ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

ВВЕДЕНИЕ В СОВРЕМЕННЫЙ ЦИФРОВОЙ МИР

РИЧАРД ФОКС

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300 Boca Raton, FL 33487-2742

© 2013 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business
No claim to original U.S. Government works
Напечатано в Соединенных Штатах Америки на бескислотной бумаге
Дата версии: 20121023

Международный стандартный номер книги: 978-1-4665-6828-0 (мягкая обложка)

В этой книге содержится информация, полученная из подлинных и высокоуважаемых источников. Для публикации достоверных данных и информации были приложены все усилия, но автор и издатель не могут взять ответственность на себя за достоверность всех материалов или последствия их использования. Авторы и издатели пытались проследить правообладателей всего материала, воспроизведенного в настоящей публикации, а также просят прощение у правообладателей, если разрешение на публикацию в этой форме не было получено. Если какой-либо материал, защищенный авторским правом, не был признан, напишите и сообщите нам об этом, чтобы мы могли исправить положение в одном из будущих новых изданий.

За исключением случаев, предусмотренных Законом США об авторских правах, ни одна из частей этой книги не может быть перепечатана, воспроизведена, передана или использована в любой форме каким-либо электронным, механическим или другими средствами, которые известны сейчас или будут изобретены позже, включая фотокопирование, микросъемку и запись в любом накопителе информации или в системе автоматического поиска без письменного разрешения от издателей.

Чтобы получить разрешение на фотокопию или использование материала в электронном виде этой работы, просьба обращаться на веб-сайт www.copyright.com (http://www.copyright.com/) или связаться с Центром по проверке авторских прав США, 01923, Массачусетс,

Данверс, Розвуд Драйв, 222, 978-750-8400. Центр по проверке авторских прав США является некоммерческой организацией, которая выдает лицензии и регистрацию различным пользователям. Для организаций, получивших фотокопию лицензии Центра по проверке авторских прав США, была организована отдельная система оплаты.

Обозначение товарного знака на изделии: Названия продукта или корпоративные названия могут быть торговыми знаками или зарегистрированными товарными знаками, и используются только для идентификации или объяснения без намерения быть нарушенными.

Library of Congress Cataloging‑in‑Publication Data

Фокс, Ричард, 1964-
Информационные технологии: введение в современный цифровой мир / Ричард Фокс.
стр. см.
Включены библиографические ссылки и индекс. ISBN 978-1-4665-6828-0 (мягкая обложка: щелочная бумага)
1. Информационные технологии. I. Название.

T58.5.F695 2013
004--dc23
2012036916

Посетите веб-сайт Taylor & Francis: http://www.taylorandfrancis.com

и веб-сайт CRC Press: http://www.crcpress.com

Содержание

ГЛАВА 1. Введение в информационные технологии 12
ЧТО ТАКОЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ? 12
Кто изучает ИТ? 17
Виды ИТ-программ 18
Программное обеспечение 30
Пользователи 32
Наш взгляд сегодня 33
Дальнейшее чтение 35

ГЛАВА 2. Организация компьютера и комплектующие 37
Структура КОМПЬЮТЕРА 39
Структура современных компьютеров 39
Что такое штифты? 41
Программа 42
Запуск программы 45
Роль центрального процессора 51
Роль памяти 54
Роль ввода и вывода 59
Аппараты и сборка компьютеров (монтаж) 66
ЦПУ 66
Память 67
Системное устройство 67
Материнская плата 68
Накопитель на жестком диске 70

ГЛАВА 3. Двоичная система исчисления 78
ДВОИЧНАЯ СИСТЕМА ИСЧИСЛЕНИЯ 82
ОТРИЦАТЕЛЬНЫЕ ЧИСЛА И ДРОБИ 91
ПРЕДСТАВЛЕНИЕ СИМВОЛОВ 97
ДВОИЧНЫЕ ОПЕРАЦИИ 103
ПОСЛАНИЕ ОТ ЗВЁЗД? 110
ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ДВОИЧНЫХ ЧИСЕЛ 111
Сетевые адреса и сетевые маски 112
Файлы изображений 114
Обнаружение и исправление ошибок 117
Дальнейшее чтение 121

ГЛАВА 4. Введение в операционные системы 123
Что такое операционная система? 123
Некоторые полезные Условия 124
Аппаратные средства 124
Программное обеспечение 125
Компьютерная система 126
Ядро 126
Драйверы устройств 127
Оболочка 127
Утилиты 127
Виртуальные машины 128
Пользовательский интерфейс 129
Управление процессом 130
Планирование 133
Управление памятью 134
Управление ресурсами 137
Управление файловой системы 140
Защита и безопасность 141
ФОРМЫ УПРАВЛЕНИЯ ПРОЦЕССОМ 142
Буттинг и инициализация системы 154
Учетная запись администратора 156
Установка ОС 158
Установка Windows 159
Установка Linux 161

ГЛАВА 5. Файлы, каталоги и файловая система 164
Файлы и каталоги 164
Файловые системы и диски 171
Дисковые утилиты 177
Файловая система LINUX 180
Файловое пространство Linux 180
Linux Разделы 186
Индексный дескриптор в Linux 188
Компьютерные Вирусы 191
Файловая система WINDOWS 192
Перемещение по системе файлов 194
Linux 194
ДОС 198
Файловая система и задачи системного администрирования 200

ГЛАВА 6. Пользователи, группы и права доступа 202
НАЛАДКА учетных записей пользователей 205
Взлом 214
Windows 215
РОЛЬ ГРУППЫ 218
Linux 222
Windows 226
Тематическое направление РАЗНОЕ учетных записей пользователей . 227

ГЛАВА 7. История компьютеров 233
Эволюция компьютерной техники 235
До Поколения 235
Второе и третье поколения 246
Четвертое поколение 250
СТАНКИ, изменившие мир 255
Эволюция программного обеспечения для компьютеров 258
Эволюции КОМПЬЮТЕР ПОЛЬЗОВАТЕЛЬ 268
ВЛИЯНИЕ НА ОБЩЕСТВО 270

ГЛАВА 8.История операционных систем 274
ПЕРЕД LINUX И WINDOWS 275
История UNIX 279
ИСТОРИЯ LINUX 281
ОТЛИЧИЯ И РАСПРЕДЕЛЕНИЯ 284
ПРОДВИЖЕНИЕ ОТКРЫТЫХ ИСХОДНЫХ КОДОВ 289
ИСТОРИЯ WINDOWS 291
ИЗУЧЕНИЕ НЕСКОЛЬКИХ ОПЕРАЦИОННЫХ СИСТЕМ 292

ГЛАВА 9. Оболочка Bash и редактирование 297
ОБОЛОЧКИ 298
ФУНКЦИИ РЕДАКТИРОВАНИЯ ОБОЛОЧКИ BASH 300
ИЗУЧЕНИЕ ИНТЕРПРЕТАТОРА BASH 308
Персонализация оболочки Bash 322
Текстовые редакторы 324
Редактор Emacs 329

ГЛАВА 10. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ 338
МЕТАСИМВОЛЫ 340
Спам-фильтры и регулярные выражения 355

Программа Grep 359
Другие методы использования регулярных выражений 367

ГЛАВА 11. Процессы и услуги 371
Запуск процесса 371
Выполнение процесса 375
Статуст процесса 380
Процесс прекращения 394
Сервисы 397
ПАКЕТЫ ОБНОВЛЕНИЯ 405
Сетевые сервисы. 406
Сервисы настройки конфигурации 408
ОПРЕДЕЛЕНИЕ СЕРВИСА ВО ВРЕМЯ ЗАГРУЗКИ 414

ГЛАВА 12. СЕТИ, СЕТЬ Программное обеспечение и Интернет 418
РАБОЧИЕ СЕТИ НА ФИЗИЧЕСКОМ УРОВНЕ 419
Сеть, ПО и интернет. 423
Сеть на логическом уровне 426
ИНТЕРНЕТ ПРОТИВ ВСЕМИРНОЙ ПАУТИНЫ 462
Краткая история Интернета 463
Будущее Интернета 466

ГЛАВА 13. Программное обеспечение 471
Типы программного обеспечения 471
ПРОГРАММНО-Сопутствующая терминология 478
Сервисы и серверы 489
Веб сервер 490
Прокси-сервер 490
Сервер баз данных 491
Сервер FTP 491
Файловый сервер 492
Сервер электронной почты 492
Система доменных имен 493

ГЛАВА 14. Программирование 499
Типы инструкций 511
Входные и выходные Инструкции 511
Операторы присваивания 513
Заявления выбора 515

Итерационные заявления 520
Подпрограммы и вызовы подпрограмм 524
Инструкции специального назначения 526
Скриптовый Язык 527
Виды Инструкции 528
Bash оболочки языка 528
MS-DOS 539

ГЛАВА 15. Информация 544
Что такое информация? 545
Базы данных 547
Информация и безопасность 563
2000 год 564
Проблемы и решения 571
Дисковые файлы 580
Криптография 583

ГЛАВА 16. Карьера в области информационных технологий 594
Сетевое администрирование 596
Администрирование системы 598
Веб-администрирование 599
Специалист Компьютерной поддержки 602
Компьютерное преступление 605
Веб-разработка 606
Программирование / инженер-программист 608
Информационные системы 609
Вирус Flame и Stuxnet 615
Другие социальные вопросы 616
Цифровой мусор 618
Непрерывное образование 619

Глава 1. Введение в информационные технологии

Этот учебник представляет собой введение в информационные технологии (ИТ), предназначенные для студентов в областях, связанных с ИТ. В этой главе представлены различные карьерные роли ИТ лица с упором на системное администрирование и типы навыков, которые требуются от ИТ-специалистов. В этой главе читатель познакомится с элементами, которые составляют ИТ-инфраструктуру — компьютер, программное обеспечение, пользователи.
Целями обучения этой главы являются:

Описание и определение различия типов ИТ карьеры.
Описание набора навыков, необходимого для достижения успеха в ИТ сфере.
Внедрение типов оборудования, найденных в компьютерной системе.
Описание и дифференциация между компонентами компьютерной системы: оборудованием, программным обеспечением и пользователями.

ЧТО ТАКОЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ?

Итак, что такое информационные технологии (ИТ)? ИТ – это термин, который используется для описания нескольких вещей, задачи сбора данных и обработки информации, способности распространять информацию с помощью технологии, сама технология, которая допускает эти задачи, а также сбор людей, которые отвечают за поддержание ИТ-инфраструктуры (компьютеры, сети, операционные системы). В общем, мы будем рассматривать ИТ как технологии, используемые при создании, сохранении и доступности информации. Другими словами, ИТ объединяет людей с вычислительными ресурсами, программным обеспечением, данными и компьютерными сетями.
ИТ-персонал, иногда именуемый как "ИТ", подразумевает категорию людей, чья работа заключается в поставке и поддержке ИТ. В эту кате-

горию входят компьютерные инженеры, которые проектируют и создают компьютерные микросхемы, компьютерные специалисты, которые пишут программное обеспечение для компьютеров, а также администраторы, которые обеспечивают организации ИТ-инфраструктурой.

Какой будет ваша роль в ИТ? ИТ-персонал имеет много различных обязанностей. В некоторых случаях ИТ-персонал в организации может представлять один человек, но в большинстве случаев такой персонал будет состоять из нескольких человек, возможно, десятков или сотен сотрудников, каждому из которых отводятся отдельные роли. Однако по большой части на ИТ-специалистах лежит две основных роли: администрирование и поддержка. Администратор - это кто-то, кто отвечает за определенную часть ИТ-инфраструктуры. Администратор может выполнять различные роли, как показано в таблице 1.1.

Давайте рассмотрим более подробно некоторые из ролей администратора в таблице 1.1. Наиболее распространенная роль ИТ - это системный администратор. Системное администрирование — это процесс сохранения операционной системы компьютера. На автономном компьютере система администрирования минимальна и, как правило, оставляется по усмотрению человека (людей), используя компьютер. Однако для сети компьютеров или компьютеров, которые совместно используют файлы или другие ресурсы, системное администрирование становится более значительным и более сложным. Системный администратор - это человек (или люди) которые выполняют системное администрирование.

Обслуживание компьютерной системы (компьютеры, ресурсы, сеть) будет включать понимание программного обеспечения, аппаратного обеспечения и программирования. С точки зрения программного обеспечения, администрирование требует установки программного обеспечения, что делает его доступным, устранение проблем, которые возникают во время использования, а также уверенность в том , что программное обеспечение работает эффективно. Кроме того, администратор(-ы) должен понимать операционную систему достаточно хорошо, чтобы настроить программное обеспечения для данной организации, создавать учетные записи, а также защищать систему от наружных атак.

С точки зрения аппаратного администрирование требует установки нового оборудования и устранения неполадок существующего оборудования. Это может включать или не включать задачи низкого уровня, такие как ремонт компонентов и прокладки сетевого кабеля. Может также потребоваться установка драйвера устройства при добавлении нового оборудования.

С точки зрения программирования операционные системы требуют "тонкой настройки", и таким образом администраторам будет приходиться часто писать свои собственные скрипты для выполнения как простых, так и сложных задач. В Linux, например, многие компоненты операционной системы зависят от файлов конфигурации. Зачастую это сценарий командного процессора. Возможно, администратору придется определить файл конфигурации и изменить его, чтобы адаптировать для работы этого компонента в пределах организации. Цель написания скриптов состоит в автоматизации процессов, поэтому администратор может вызвать скрипты, когда-то написанные, для выполнения задач, которые в противном случае были бы затруднительными. В качестве простого примера может служить написание скрипта, который будет принимать текстовый файл имен пользователей и создавать новую учетную запись для каждого имени пользователя.

Системное администрирование может ограничиваться для администрирования компьютеров, принтеров и файловых серверов. Тем не менее, система администрирования может распространяться на администрирование сети и, возможно, даже веб-сервер, FTP-сервер, почтовый сервер и администрирования сервера баз данных в зависимости от потребностей и размера компании и возможностей системного администратора (-ов). И, наконец, системный администратор может также потребоваться для обучения пользователей в системе. Таким образом, навыки, необходимые для администрирования системы, могут сильно варьироваться. Конкретные общие задачи системного администратора включают:

ТАБЛИЦА 1.1 Роли администратора в ИТ

Роль	Работа/задачи
Системный администратор	Администрирование компьютеров в Организации; установка программного обеспечения; изменение или обновление операционной системы; создание учетных записей; обучение пользователей; безопасная система; устранение неполадок системы; добавление оборудования
Сетевой администратор	Покупка, настройка и подключение компьютерной сети; поддержка компьютерной сети; устранение неполадок сети; безопасная сеть от вторжений
базы данных	Установка, настройка и поддержание базы данных и системы управления базами данных; резервное копирование базы данных; создание учетных записей; обучение пользователей
Веб-администратор	Установка, настройка и обслуживание сайта через веб-сервер; защита сайта; работа с разработчиками
Веб-разработчик	Разработка и создание веб-страниц и сценариев для веб-страниц; поддержание веб-сайтов
Администратор безопасности	Установка, настройка и администрирование брандмауэра; создание политики безопасности; устранение неполадок компьютерной системы (включая сети); активное противостояние атакам

Управление учетными записями: создание новых учетных записей пользователей и удаление устаревших учетных записей пользователей.
Управление паролями: убеждение, что все пользователи имеют пароли, которые соответствуют политике безопасности (например, пароли необходимо изменять каждый месяц, пароли должны содержать хотя бы один небуквенный символ) – вы можете удивиться, но в системах без надлежащего пароля управления многие пользователи используют в качестве пароля "" (например, их пароль может состоять из обычного нажатия клавиши Enter). Большинство организаций сегодня требуют использования *надежных паролей*: пароли, которые содержат минимум восемь символов, из которых по крайней мере один небуквенный символ и/или комбинацию прописных и строчных букв, а также сменять пароли хотя бы один раз в месяц без повторного использования паролей в течение периода, продолжительностью несколько месяцев.
Управление защитой файлов: убеждение, что файлы надлежащим образом защищены (например, убедитесь, что важные документы недоступны для записи внешним миром) и выполняется своевременное резервное копирование файловой системы.
Установка и настройка оборудования и устранение неполадок оборудования, включая сеть.
Установка и настройка нового программного обеспечения, включая обновление новых пакетов операционной системы (ОС) и устранение неполадок программного обеспечения.
Предоставление документации, поддержка и обучение пользователей компьютеров.
Выполнение программирования на системном уровне по мере необходимости (как правило, через языки скриптов, а не написание крупномасштабных приложений или систем программного обеспечения).
Безопасность: установка и техническое обслуживание брандмауэра, изучение файлов журналов на предмет наличия любых нечетных диаграмм направленности попыток войти в систему, а также изучение подозрительных процессов, которые, возможно, не должны быть запущены.
Во многих случаях администратор сети не имеет ничего общего с системным администратором. Это сетевой администратор, который отвечает за все аспекты работы компьютерной сети. Обязанности администратора сети будет включать в себя физическую прокладку кабеля, создание подключения и работу с сетевым оборудованием (например, маршрутизаторами и коммутаторами). Администратору сети также придется настроить отдельные устройства для возможности общаться

через сеть. Таким образом, как системный администратор, администратор сети будет редактировать конфигурационные файлы, устанавливать программное обеспечение (связанные с сетью) и так далее. Устранение неполадок сети будет также являться ролью администратора сети, в этом случае устранение неполадок может сочетать физическое устранение неполадок (например, является ли этот кабель плохим?) и устранение неполадок программного обеспечения. Существует также аспект безопасности компьютерной сети. Как системный администратор, так и администратор сети могут работать в системах брандмауэров. Редактирование файлов конфигурации, написание скриптов и установка программного обеспечения, а также исправления - это все будет частью задач администраторов сети.

Кроме административных задач, ИТ-персонал обеспечивают поддержку. Поддержка обычно предоставляется в двух формах: обучение и справочная служба. Обучая, ИТ-сотрудник отвечает за обучение новых и существующих пользователей, как использовать ИТ-инфраструктуру. Это может включать такие простые вещи, как войти в компьютерную систему, настройка принтеров, доступ к общим файлам и, возможно, обучение сотрудников в том, как использовать программное обеспечение, связанное с работой. Сотрудник может создавать документы, полезные веб-сайты (включая вики-страницы) и даже аудиовизуальные демонстрационные версии или вести групповые, или индивидуальные занятия. Поскольку обучение происходит только по мере необходимости (новое программное обеспечение, новые сотрудники), в большинстве случаев поддержка предоставляется в виде справочной службы. По сути, для этого требуется, чтобы кто-то был доступным для решения проблем, которые возникают в случайные моменты времени. Многие крупные организации предлагают справочные службы 24/7. Сотрудник справочной службы может просто выступать в качестве распределительного щита, маршрутизации проблемы для надлежащего специалиста ИТ. В других случаях, специалист службы поддержки может решить эту проблему напрямую, часто по телефону, а иногда по электронной почте или лично.

Где ИТ используется? ИТ сегодня является вездесущей. Почти все на планете использует некоторые формы вычислительной техники, используя сотовые телефоны и планшеты или домашние компьютеры, или в школе и на работе. Однако чаще всего ИТ-персонал нанимают для работы в ИТ-отделы организаций. Эти организации могут состоять всего из нескольких человек или корпораций, в которых работают десятки тысяч людей.

ТАБЛИЦА 1.2 Крупные пользователи ИТ

Тип организации	Типичное использование
Бизнес	Электронная коммерция, записи клиентов
Образование	Схоластической учет, поддержка обучения
Развлечения	Цифровое редактирование, специальные эффекты, музыкальная композиция, рекламы
Правительство	Учет, анализ разведывательных данных, распространение информации
Организации здравоохранения/ больницы	Делопроизводство, медицинское оборудование, страхование
Правоохранительные органы	Делопроизводство, сбор информации и распространение
Производство	Проектирование, автоматизация/робототехника
Исследования	Вычисление, распространение информации

Кто изучает ИТ?

ИТ-специалисты в прошлом часто отбирались на должность. Рассмотрим следующий сценарий. Джо получил степень бакалавра в области компьютерных наук в Университете штата Иллинойс. Он сразу же был принят в фирму по разработке и продаже программного обеспечения в Чикаго, где он занял должность программиста на языке COBOL. Тем не менее, в течение 3-х месяцев босс, будучи новичком, спросил его: "Уверен, вы знаете что-то об операционной системе Linux, не так ли?" Джо, конечно, изучал Unix как часть программы бакалаврской степени и ответил: "Конечно". Так что босс сказал Джо: "С этого момента я хочу, чтобы вы потратили 10 часов вашей недели, соединив воедино эту новую сеть компьютеров, используя Linux. Убедитесь, что она может подключиться к нашим файловым серверам и сделайте их безопасными". Джо потратил 10 часов недели для чтения руководства, установки операционной системы Linux, играя с операционной системой, и в конечном итоге сумел запустить систему.

После определенного первоначального болезненного роста проблем в использовании системы, все больше и больше сотрудников перешли на платформу Linux. Теперь, спустя 9 месяцев, половина компании перешла на Linux, но система не всегда работает гладко. Всякий раз, когда возникает проблема, Джо, как правило, является тем человеком, который должен ответить и исправить ее. Босс снова обращается к Джо: "Ваша работа с сетью была великолепна. Я хочу переместить вас на полный рабочий день, чтобы поддерживать систему." Джо не обучался этому, а поскольку в него были некоторые навыки, а также потому, что он умный, трудолюбивый индивидуум (он же должен был окончить университет в

Иллинойсе по образовательной программе в области информатики!), он был успешным в этом начинании. Вместо того чтобы нанимать кого-либо для поддержки системы, простым решением было выдвинуть Джо в должности на постоянной основе. Бедный Джо, он хотел написать код (хотя, возможно, не COBOL). Но теперь, единственный код, который он пишет, представляют собой скрипты оболочки Linux! Звучит маловероятно? На самом деле, это была очень распространенная история в 1980-х и 1990-х годах и даже в 2000-е годы. Лишь в середине 2000-х годов была разработана учебная программа в сфере ИТ, чтобы соответствовать роли ИТ-персонала. В противном случае, такие рабочие места часто заполняли компьютерные ученые или люди, которые только стали увлекаться компьютерами. Несколькими "квалифицированными" сотрудниками считались те, кто имел соответствующие степени после 2-летних технических вузов, но эти колледжи больше ориентировались на охвате таких понятий, как ремонт ПК и устранения неполадок, а не на системном и сетевом администрировании. Сегодня мы ожидаем увидеть ИТ людей, которые не только прошли подготовку по современной технологии, но понимают все аспекты ИТ-инфраструктуры, включая теоретические вопросы, математику компьютеров (бинары), роль различных компонентов, которые составляют компьютерную систему, методы программирования, операции баз данных, сетей, Интернета, и, возможно, имеют специальные знания, такие как компьютерная судебная экспертиза. Общие ИТ-программы включают в себя введение в платформы операционных систем, языков программирования и вычислительных концепций. Мы ожидаем, что студент будет иметь опыт работы как в Windows, так и Linux (или Unix). Языки программирования могут включать в себя как скриптовые языки, такие как Linux/Unix скриптов shell, Ruby или Python и JavaScript, и скомпилированные языки, такие как C, C ++, Java или Visual Basic. Концепции будут включать в себя операционные системы и сети, но могут выходить за рамки, чтобы включить веб-инфраструктуру, компьютерные архитектуры, программные приложения (например, бизнес-программное обеспечение), цифровые средства массовой информации и хранения данных и электронной коммерции.

Виды ИТ-программ

Хотя 4-летняя IT степень является относительно новой, она также не стандартизирована. Различные университеты, которые предлагают такую ИТ-программу, соглашаются реализовывать ее по разным точкам

зрения. Здесь мы рассмотрим более общие подходы.
Во-первых, это программы, которые являются ответвлениями степени компьютерных наук. Кажется естественным, объединить две степени вместе, потому что многие знания, которые обе дисциплины должны охватывать, дублируются: аппаратные технологии, программирование, проектирование баз данных, компьютерная этика, сети. Тем не менее, степень компьютерных наук всегда сильно вращалась вокруг программирования, а ИТ-степень может потребовать меньше знаний. Кроме того, математика играет существенную роль в информатике, но неясно, требуется ли такое количество математики для ИТ.
Далее, существуют вариации систем управленческой информации. Идея заключается в том, что она должна преподаваться с использованием такой точки зрения – больше приложений, хранения данных в базе данных, а не на технологии, лежащей в основе бизнес-приложений. Электронная коммерция, разработка баз данных, интеллектуальный анализ данных, компьютерная этика и закон продвинуты здесь. Кроме того, работа курса может включать концепции, связанные с управлением ИТ.
Тогда существует инженерный технологический подход, который концентрируется на аппаратуре – платах , схемах, дисковых накопителях, архитектуре ПК и устранении неисправностей, физических аспектах сетей. Тут меньше внимания уделяется программированию, хотя еще есть компонент программирования.
Другой подход – это обеспечение основы самих компьютерных систем. Этот учебник следует этой идее , представив первую аппаратную часть компьютерной системы, а затем операционные системы. Мы также рассмотрим компьютерные сети, программирование, и память ЭВМ, чтобы иметь хорошо сформированное понимание технологической стороны ИТ. ИТ-выпускник должен уметь не только работать с ИТ, скажем, как администратор, но и проектировать ИТ-системы на архитектурном уровне от аппаратных средств к сети и к программному обеспечению.
SIGITE, Специальная группа ACM по ИТ-образованию, предоставляет полезные рекомендации для построения модели ИТ-учебного плана.

Кто должен изучать ИТ? Чтобы быть ИТ-специалистом, вы не обязательно должны иметь строгое математическое или инженерное образование компьютерных ученых и компьютерных инженеров; Есть много смежных талантов. Возможно, самый важный талант – это навыки поиска *неисправностей*. Значительная часть работы ИТ-специалиста заключается в поиске того, что в вашей системе работает неправильно. Эти ди-

агностические навыки не могут быть чисто выучены. Вы должны иметь опыт, фоновые знания и инстинкт. Прежде всего, вы должны знать, как работает система, находится ли система под управлением операционной системы Linux, компьютерной сети, веб-сервера и т. д. Еще один талант – это умение писать программный код –, по всей вероятности, вы должны уметь написать небольшие программы или скрипты, в отличие от инженера-программиста, который будет участвовать в крупномасштабных проектах.

Вы также должны быть в состоянии общаться с другими людьми, чтобы вы могли понять проблемы, о которых сообщили ваши коллеги или клиенты, и в свою очередь, описывать пути решения этих проблем. Это взаимодействие может происходить по телефону, а не лично. Вы также должны быть в состоянии писать технически. Вам часто может быть предложено подготовить документацию и отчеты. И, наконец, вам потребуется способность учиться самостоятельно, поскольку технология постоянно меняется. То, что вы узнали в период обучения или путем самообразования раньше, может быть устаревшим в течение года или двух. Тем не менее, то, что вы узнаете, должно формировать фундамент, из которого вы можете продолжать учиться. Посмотрите таблицу 1.3, которая выдвигает на первый план навыки, ожидаемые или требуемые

Таблица 1.3 ИТ Навыки

Навыки	Описание	Пример(ы)
Поиск и устранение неисправностей, решение проблем	Обнаружить проблему Диагностировать ее причину Найти решение (средства его фиксации) Операционная система приложений создание учетных записей пользователя установки системы Системный мониторинг скрипты Shell для автоматизации процессов управления файлами конфигурации для системных служб	Дисковое пространство Низкая производительность процессора полный Вирусы или троян инфекции Версии Linux Версии Unix Windows Mac OS
Знание операционных систем		Bash, CSH скрипты DOS скрипты Рубин скрипты C / C ++ программ
Программирование на уровне системы		
Система безопасности	Обеспечение надлежащей безопасности системы в месте После или до разработки политики для пользователей Мониторинг угроз	Настройка брандмауэра системы Установка противовирусного / программного обеспечения защиты от вредоносных программ экспертизы файлы журналов для доказательства вторжений и системных дыр в безопасности идти в ногу с последними обновлениями безопасности
Аппаратура	Установка и настройка нового оборудования Поиск и устранение неисправностей, замена или ремонт неисправного оборудования	Замена процессоров и дисковых накопителей Подключение сетевых кабелей к сетевым концентраторам, коммутаторам, маршрутизаторам

от ИТ-персонала.
Это, вероятно, не прототип ИТ-студента. Но ИТ-студент должен:
1. Наслаждаться , играя с компьютером, не просто использовать его, но и узнать, как он работает, учится, как делать вещи на нем на системном уровне
2. Наслаждаться самообучением - задача выяснить вещи, особенно новые вещи
3. Думать , что технология – это круто, не бояться технологии, но принять ее во всех его формах
4. Наслаждайтесь поиском и устранением неисправностей.
Это не обязательно так, что ИТ-студент будет получать удовольствие от программирования. На самом деле, многие студенты, которые выбирают ИТ в качестве карьеры делают этот выбор, потому что они сначала начинают с компьютерной науки, но вскоре устают от тяжелых программных требований этой дисциплины. Это не означает, что ИТ-студент не программирует, а что программирование осуществляется менее интенсивно, что требует в основном написания небольших скриптов. Например, студент информатики может в ходе своих исследований написать текстовый процессор, систему управления базами данных или переводчик языка, в то время как ИТ-студент может писать сценарии для автоматизации создания учетной записи пользователя или написать сценарии на стороне клиента, чтобы гарантировать, что веб-формы были заполнены правильно, или писать серверные сценарии для обработки веб-форм. Есть много граней должности системного администрирования, не охваченные ранее, которые стоит отметить. Студенты могут думать, что, ступая на путь ИТ-карьеры, они получат работу, где они получат возможность "поиграть" с технологией. Это смешно, но это также является проблемой, ведь это почти то же, что хобби. И все-таки студент, начиная работать, может нести ответственность за поддержание ИТ-инфраструктуры в организации десятков или сотен сотрудников. Оборудование может стоить сотни тысяч долларов, но сам бизнес может приносить миллионы долларов. Таким образом, ИТ-специалист должен принять свою работу серьезно – простои, системные ошибки, вторжения и т.д. могут дорого стоить организации. Специалист ИТ имеет обязанности, выходящие за рамки простого системного администратора. Некоторые из этих ожиданий прорабатываются ниже.
Для начала системный администратор должен быть в курсе событий в этой области. Как минимум, системный администратор должен знать о проблемах безопасности, которые возникают и как защититься от них.

Они могут включать в себя обеспечение системы от вирусов, сетевых вторжений, отказы сервисных атак, а также атак с внедрением SQL. Кроме того, системный администратор должен не отставать от новых версий операционной системы и / или программного обеспечения сервера, который он / она поддерживает. Тем не менее, системному администратору, возможно, придется выйти далеко за рамки, читая о новом оборудовании, новом программном обеспечении, а также других подобных разработках в этой области.

Непрерывное образование имеет важное значение для того, чтобы системный администратор шел в ногу с новыми технологиями, новыми тенденциями и новыми исправлениями безопасности. Системный администратор должен быть пожизненным учеником и начинателем. Администратор системы может смотреть интернет-форумы, но и должен регулярно читать новости о технологиях и быть готовым следить за статьями путем собственных исследований. Администратор системы также должен быть готов погрузиться в новое программное обеспечение и экспериментировать с ним, чтобы определить его потенциальное использование в рамках организации. Системный администратор часто будет "на вызове" в нерабочее время. Когда приходит беда, системный администратор должен быть доступен. Экстренный вызов в 3 утра или во время отпуска вполне возможен. Хотя каждый работник заслуживает на личное время отдыха, контракт администратора системы может включать в себя положения о том, куда можно обращаться 24/7. Без такой уверенности, организации могут оказаться недоступными к файлам данных или не иметь возможности выполнить транзакции в течение нескольких часов, что может привести к ущербу на миллионы долларов. Через атаки с целью нарушения нормального обслуживания пользователей и неспособность быстрого восстановления репутация некоторых компаний может пострадать.

Системный администратор должен также вести себя этично. Тем не менее, это часто для студента стает неожиданностью, что этика воспринимается как вопрос. И все же, что бы вы сделали, если бы столкнулись с какой-то моральной дилеммой? Например, ваш работодатель обеспокоен тем, что слишком многие сотрудники используют электронную почту компании для личных вещей, и поэтому босс просит вас смотреть электронную почту каждого. Как бы вы себя чувствовали? Теперь представьте, что существует политика в компании, которая заявляет, что сотрудники могут использовать электронную почту компании для личных целей до тех пор, пока электронная почта не раскрывает никаких секре-

тов компании. В этом случае, если вас просят смотреть почту сотрудника, как вы почувствуете себя в этой ситуации?
Неэтичное поведение может включать в себя:
1. Шпионаж за другими (электронная почта, веб-браузер привычки, рассматривая файлы)
2. Настройка бэкдор счета для осуществления несанкционированного доступа компьютерных систем
3. Незаконная загрузка программного обеспечения или файлов, или поощрение / выдача разрешений другим делать это
4. Выполнение кражи или саботаж из-за вашего доступа системного администрирования

ИТ-инфраструктура

ИТ вращается вокруг компьютера. Вы использовали компьютер сегодня? Даже если вы не коснулись вашего настольного компьютера (или ноутбука), чтобы проверить электронную почту, есть вероятность, что вы использовали компьютер. Ваш мобильный телефон – это компьютер, как и ваше устройство для чтения электронных книг. Они гораздо менее мощные, чем настольные устройства, но они, тем не менее, компьютеры. Компьютерные компоненты присутствуют в вашем автомобиле и на городских улицах, по которым вы ездите. Здание, в котором вы работаете или учитесь, может использовать компьютеры для управления освещением и кондиционером. Да, компьютеры нас окружают везде, даже если мы не распознаем их.

Мы будем определять компьютер как часть электронного оборудования, которое способно запускать программы, взаимодействуя с пользователем (с помощью устройств ввода-вывода), и хранить данные. Эти задачи часто называют циклом ВОВХ (ввод, обработка, вывод, хранения). Компьютер общего назначения представляет собой устройство, которое может запустить любую программу. Многие устройства сегодня – это компьютеры, но они могут быть не только общего назначения, так и другого назначения. Например, ваш IPod способен воспроизводить музыку; он имеет пользовательский интерфейс, и может содержать небольшое количество приложений, загруженных в него, чтобы управлять календарем, показать время суток, и предложить несколько игр. Ваш мобильный телефон содержит еще большее число приложений, но он не может запускать большинство программ. Степень, при которой компьютер относится к устройству общего назначения, в значительной степени основана на емкости и том, будут ли программы специально написаны для процессора.

Компьютеры различаются по размеру и возможностям – от суперкомпьютеров, которые могут заполнить комнату, до настольных устройств, которые не являются очень тяжелыми, но не предназначены для портативных устройств, лэптопов, которые, так легки, как, возможно, тяжелый учебник, для мобильных устройств, таких как сотовые телефоны и mp3-плееры. Общая разница между мобильным устройством и настольным или портативным устройством заключается в типах доступных периферийных устройств (полноразмерная клавиатура и мышь по сравнению с сенсорным экраном, 20-ти дюймовый монитор по сравнению с 2 дюймовым экраном), объеме оперативной памяти и пространстве для хранения жесткого диска, а также в доступности внешнего запоминающего устройства, например, флэш-накопителей через USB-порты или оптические диски через дисковод оптических дисков.

Компьютеры

Мы будем изучать, из чего состоит компьютер более подробно в следующей главе. На данный момент, мы будем смотреть на компьютер в более общих чертах. Компьютер представляет собой электронное, программируемое устройство. Чтобы запустить программу, устройство нуждается в процессоре [центральный процессор (ЦП)], памяти для хранения программ и данных, возможности ввода и вывода, а также возможности долгосрочного хранения и сетевых возможностях (последние два не являются обязательными). Исходя из этого определения, компьютеры не только включать в себя настольные и портативные устройства, серверы, универсальные компьютеры и суперкомпьютеры, но и нетбуки, мобильные телефоны, компьютерные игровые приставки, mp3-плееры и книги (например, устройство для чтения электронных книг). В двух последних случаях, устройства являются специализированными – они запускают лишь несколько избранных программ. Понятие исторического компьютера ушло. Сегодня мы живем с компьютерами во всем мире.
Рисунок 1.1 иллюстрирует некоторый диапазон в компьютерах. Настольные блоки с большими мониторами и системными блоками являются общими так же, как и современные портативные компьютеры с большими мониторами. Еще более популярными являются портативные устройства, включая карманные персональные компьютеры (КПК), сотовые телефоны и электронные книги. Мониторы – это плоские экраны. Мы больше не ожидаем обнаружить громоздкие мониторы на наших настольных компьютерах. Тем не менее, системный блок, который позво-

ляет нам иметь множество дисководов и других компонентов, является громоздким. Мы жертвуем некоторыми из периферийных устройств, когда используем портативные компьютеры. Мы жертвуем большим количеством доступности, когда переходим к переносным устройствам. В случае КПК, лэптопов, ноутбуков, чипов и материнской платы, а также всех других форм хранения, они должны быть размещены на довольно небольших площадях. Для КПК, скорее всего, это просто беспроводная карта, чтобы разрешить доступ к сети сотового телефона (и, возможно, Wi-Fi). Для портативных компьютеров и ноутбуков, это вероятно, жесткий диск. Лэптоп будет также, вероятно, иметь накопитель на оптических дисках.

Основным компонентом компьютера является *процессор*. Роль процессора заключается в процессе, то есть, он выполняет программы, которые мы запускаем на компьютере. Для запуска программы на данном ком-

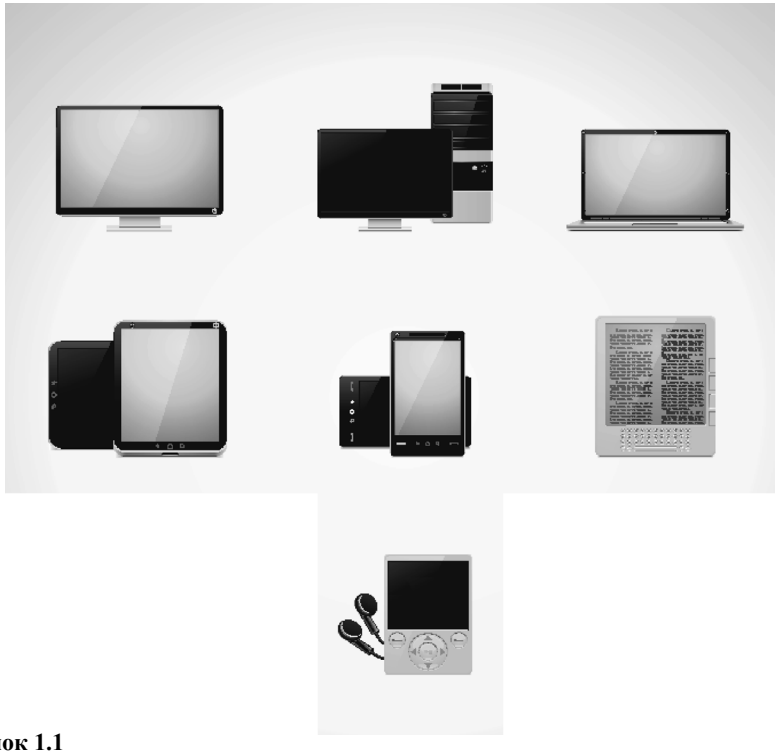


Рисунок 1.1
Типы компьютеров. (Адаптированы из Shutterstock/tele52.)

пьютере программа должна быть создана для этого компьютера. Компиляция – это процесс трансляции языка, который преобразует программу из более читаемого вида (скажем, Python или Java) в машинный язык компьютера. Компьютер может запускать только те программы, которые написаны на языке этой машины. Мы обсудим эту концепцию более подробно, а также особую роль процессора в главе 2. Позже в учебнике мы рассмотрим языки программирования. Помимо процессора, компьютеры нуждаются в хранилище. Есть два типа хранилища – кратковременное и долговременное. Кратковременное хранилище – это чаще всего оперативное запоминающее устройство (ОЗУ). К сожалению, оперативная память может описать несколько различных типов. Наши современные компьютеры, как правило, имеют три формы памяти: динамическое ОЗУ (то, что мы обычно называем основная память), статическая оперативная память (кэш-память и регистры) и ПЗУ (постоянное запоминающее устройство). Мы приведем различия между этими типами памяти в главе 2. На данный момент, прямо сейчас рассмотрим все три для "запоминания".

Память (динамическое ОЗУ) состоит из микросхем. Динамическое ОЗУ обеспечивает быстрый доступ и часто большую емкость. Тем не менее, некоторые портативные устройства не имеют большую оперативную память ОЗУ. Вместо этого они используют для хранения флэш-память, которая является более ограниченной по объему. Длительное хранение наиболее часто использует жесткие диски, но также может включать в себя оптический диск, флэш-память и магнитную ленту. Длительное хранение обладает гораздо большей вместимостью, чем кратковременное хранение оперативной памяти, а также потому, что дополнительное место для хранения всегда можно приобрести; мы могли бы рассмотреть долгосрочную емкость как неограниченную. Типичное кратковременное хранение десктопа и лэптопа находится в диапазоне 1-8 Гб. 1 Гбайт означает 1 гигабайт, что примерно 1 миллиард байтов. Подумайте о байте как 1 символе (буква), так что 1 Гб будет хранить 1 миллиард символов. Типичная книга составляет, вероятно, порядка 250 тысяч до 1 миллиона символов. 1 Гб будет хранить не менее 1000 книг (без картинок). Жесткие диски могут теперь хранить 1 ТБ, или 1 терабайт, который составляет 1 триллион байтов. Очевидно, что длительный срок хранения гораздо больше, чем в емкости кратковременного хранения. Некоторые общие размеры хранения приведены в таблице 1.4. Мы будем изучать размеры хранения более подробно в главах 2 и 3.

Таблица 1.4. Размеры хранения

Размер	Значение	Пример
1 бит	0 или 1	Наименьшая единица хранения, может хранить 1 черно-белый пиксель или 1 истинное / ложное значение, как правило, мы должны объединить много битов, чтобы создать что-либо значимое правило, мы должны объединить много битов, чтобы создать что-либо значимое
1байт (1B)	8 бит	Мы могли бы хранить число от 0 до 255 или -128 до 127, или один символ (букву алфавита, цифру, знак препинания)
1 слово	32 или 64 бит	Одна часть данных, таких как числа или команды программы
1 кб	1024 байт	Мы можем хранить блок памяти в этом размере
1 мб	~1 млн байт	Небольшое изображение или большой текстовый файл, mp3-файл песни может занять от 3 до 10 МБ, 50-минутный ТВ-шоу с высокой степенью сжатия может потребоваться 350 МБ
1 гб	~1 блн байт	Библиотека песен или изображений, десятки книг, DVD требуется несколько гигабайт дискового пространства (4-8 ГБ)
1 тб	~1 тлн байт	Библиотека фильмов

Нужны ли нам обе формы хранения? Это зависит от типа устройства и вашего предполагаемого использования, но в общем, да мы нуждаемся в них обоих. Зачем? Чтобы запустить программу, нам нужно загрузить эту программу в пространство для хранения, которое быстро реагирует. Длительное хранение гораздо медленнее, чем ОЗУ, поэтому, если вы не желаете иметь очень медленный компьютер, то вам нужно кратковременное хранилище. С другой стороны, кратковременное хранилище гораздо более ограничено в емкости и программы, которые мы запускаем, как правило, очень большие. Мы также часто имеем очень большие файлы данных (музыкальные файлы, фильмы и т.д.), поэтому мы не можем полагаться только на кратковременное хранение. Портативные устройства предлагают компромисс, они часто используют флэш-память вместо оперативной памяти, что приводит к замедлению времени доступа по сравнению с настольными / портативными компьютерами, и они име-

ют ограниченное пространство долгосрочного хранения (если таковые имеются), требуя, чтобы пользователь довольно часто перемещал файлы между карманными устройствами и местами постоянного хранения (скажем, на настольном компьютере).

Помимо разницы в скорости и емкости для хранения между памятью и долговременным хранилищем, другим различительным фактором является их изменчивость. Термин летучий, при описании памяти, указывает на способность типа памяти сохранить свое содержимое при отключении источника питания. Оперативная память (ДЗУПВ) и кэш / память регистра (СОЗУ) представляют собой летучие формы памяти. После того, как вы выключите питание, содержимое теряется. Вот почему, когда вы включаете компьютер, память изначально пуста, а устройство должно пройти через процесс "загрузки". Энергонезависимые памяти включают в себя ПЗУ, флэш-диски, жесткий диск и оптический диск. Энергонезависимая память сохраняет свое содержимое неограниченно долго. В случае ПЗУ, содержимое никогда не теряется. В случае флэш-накопителей и дисков, содержимое сохраняется до того момента, пока вы не решите удалить его.

Компьютеры также требуют периферийных устройств (хотя "требуют" не является правильным словом; возможно, мы должны сказать, что для удобства пользователя мы добавим периферийные устройства). Слово "периферийный" означает "на окраине", но в случае компьютера, мы обычно ссылаемся на периферийные устройства, как устройства, которые находятся вне компьютера или, более конкретно, за пределами системного блока. Системный блок – это коробка, которая содержит материнскую плату (на которой находится центральный процессор и память) и дисковод. Периферийные устройства являются устройствами, которые или слишком велики, чтобы поместиться внутри системного блока или устройства, которые должны быть доступны пользователям (Рисунок 1.2). Этими устройствами являются наши устройства ввода и вывода-клавиатуры, мыши, трекпоинт, трекбол или джойстик, монитор, принтер, звуковые колонки, перо и планшет (область записи) или световое перо и т.д. Без этих устройств ввода и вывода (известных коллективно как устройство ввода/вывода) люди не смогут взаимодействовать с компьютером. Если все входные данные поступают из файла на диске и все выходные данные аналогично будут сохранены в файл на диск, то нет никакой необходимости взаимодействия компьютера с человеком. Но человек, в конечном счете, хочет знать, что сделала программа. Среди периферийных устройств есть устройство (-а) связи. Устройство свя-

зи – это то, что позволяет компьютеру обмениваться данными с другими компьютерами. Эти устройства, как правило, модемы, которые могут либо требовать подключения к телефонной линии (или, возможно, кабельному телевидению коаксиальной линии) или могут быть беспроводными. Почти все портативные компьютеры сегодня приходят с беспроводными модемами, в то время как настольные единицы могут прийти с проводным или беспроводным модемом. Однако в тех случаях, когда компьютеры подключены к локальной сети (LAN), компьютер требует подключения к сети вместо или в дополнение к модему. LAN соединение осуществляется с помощью сетевой карты, часто Ethernet-плата. Если говорить о высокоскоростных сетях, то сетевая карта предлагает гораздо более высокую пропускную способность (скорость передачи данных), чем модем.

Мы будем изучать проводные и беспроводные модемы и сетевые карты в учебнике далее, когда рассмотрим компьютерные сети.

Давайте кратко изложим суть знаний о нашем компьютере. Компьютер, в сущности, представляет собой совокупность различных устройств, каждое из которых выполняет различные типы задач. Типичный компьютер будет включать следующее:

Рисунок 1.2
Периферийные компьютеры. (Представлено Shutterstock/Nevena.)

Системный блок, в котором находится
а. материнская плата, которая содержит
Процессор
Блок охлаждения для CPU
Возможно, дополнительные процессоры (например, для работы с графикой)
Микросхемы памяти для ОЗУ, ПЗУ
Разъемы для подключения периферийных устройств (иногда называемые портами)
Слоты расширения для других карт периферийного устройства
ROM BIOS для загрузки и базовые инструкции системы ввода и вывода
Соединитель электропитания
б. Дисководы
в. Вентиляционная установка
г. Питание
Монитор и клавиатура
Как правило, некоторая форма указательного устройства (мыши, трекпоинт, трекболом)
Колонки (по желанию)
MODEM или сетевая карта (они обычно располагаются внутри системного блока, подключены к одному из слотов расширения)
Принтер (по желанию)
Внешние устройства хранения данных, такие как внешний жесткий диск и ленточный накопитель.
В главе 2 размещены фото, иллюстрирующие многие из вышеуказанных компонентов.
Теперь мы разобрались со значением слова "компьютер ". Но компьютер – это только часть истории. Без программного обеспечения компьютер не будет способным выполнять работу. А без людей компьютер не будет знать, ни какую программу запускать, ни на какие данные использовать. Таким образом, наша компьютерная система включает в себя эти компоненты.

Программное обеспечение

Какой смысл компьютера? Для запуска программы. Без программного обеспечения компьютер не будет способным выполнять работу. Программа, известная также как программное обеспечение (чтобы отличать ее от физических компонентов компьютера, аппаратного обеспечения),

представляет собой список инструкций, которые конкретизируют компьютеру его действия. Эти инструкции написаны на языке программирования, таких как Java или Python. Инструкции языка программирования должны быть очень описательного характера. Например, если вам нужно, чтобы компьютер ввел 2 пользовательских числа и определил, которое из них является более длинным, вы не можете просто сказать: "Введи два числа и выведи, которое более длинное из них." Вместо этого, вы должны описать действия, которые должны иметь место как алгоритм, разбитый на пошаговые инструкции. Инструкции должны быть написаны на языке программирования. Например, проблема, описанная в этом пункте, может быть разбита на четыре этапа:
ввод числа 1
ввод числа 2
сравнение этих двух чисел, и если первое больше, чем второе, вывод числа 1
в противном случае выводится число 2.

На таком языке, как C, это будет выглядеть следующим образом:

```
scanf(“%d”, &number1);
scanf(“%d”, &number2);
if(number1 > number2) printf(“%d is greater”, number1);
else printf(“%d is greater”, number2);
```

Команда scanf вводит значение, команда PRINTF выводит значение или сообщение. Команда if используется для сравнения двух значений и принятия решения. Некоторый синтаксис в C является специфическим, например, & перед "number1" и "number2" в Scanf операторах, использование точки с запятой для завершения инструкции, а также использование %d. Каждый язык будет иметь свой собственный синтаксис и во многих случаях, синтаксис может оказаться очень странным для кого-то, кто не является программистом или не изучал этот язык.
Программы – это не только список исполняемых операторов. Программы также требуют различных определений. Они могут включать объявления переменных, функции или методы и классы. На языке C, например, мы должны определить number1 и number2 как переменные, которые будут использоваться в коде выше. В этом примере, они будут объявлены как целые числа.
Есть много форм программного обеспечения, но мы в целом делим их на две категории: системное программное обеспечение (операционная

система) и прикладное программное обеспечение (программы, которые мы запускаем, чтобы выполнить наши задачи, такие как текстовый процессор, интернет-браузер или компьютерная игра). Как правило, наше программное обеспечение написано инженерами-профессионалами программного обеспечения. Однако, как только вы научитесь программировать, вы можете написать свое собственное программное обеспечение, если вы желаете. Как ИТ-студент, вы научитесь писать короткие куски кода, скрипты. Скрипты могут быть использованы для поддержки или операционной системы, или приложения. Например, вы могли бы написать скрипт оболочки Bash для поддержки деятельности, такой как автоматическое создание учетных записей пользователей для новой группы пользователей. Или вы можете написать серверный скрипт в Perl, чтобы проверить URL на наличие угроз безопасности в веб-сервере.

Пользователи

Без человека компьютер не будет обладать какими-либо ресурсами для работы. Это пользователь, который инициировал процессы на компьютере. "Делайте это сейчас, сделайте это позже." Нам может понадобиться взаимодействие с программами в процессе их выполнения. Эта интерактивность осуществляется с помощью устройств ввода/вывода. Сегодня мы настолько привыкли к интерактивности, что мы, вероятно, не можем себе представить использование компьютеров без нее. Но в прежние времена (1940-1970-е годы), большинство, если не все, обработок были сделаны вообще без вмешательства человека. Пользователь указал программу, источник входных данных, местонахождение выхода и отправил программу на выполнение. Пользователь мог видеть результаты, как только компьютер запустил программу – немедленно или много часов спустя!

Пользователи прогрессировали с течением времени, так же, как технология продвигалась. Самые ранние пользователи компьютера были инженерами, которые создавали и программировали их. Компьютеры были настолько сложны и дороги, что никто не имел доступ к ним. Так, как компьютерные расходы позволяли организации приобретать их (за миллионы долларов), то пользователями компьютеров были те сотрудники, которые получили специальную подготовку для их использования. Ситуация начала меняться с появлением персональных компьютеров, впервые выпущенными в 1970-е годы. Но это было до тех пор, пока не появились оконные операционные системы, так что пользователи ком-

пьютеров могли научиться использовать компьютеры при минимальной подготовке или практически без подготовки. И вот сегодня, он является распространенным, так что любой и каждый может использовать компьютер. На самом деле, компьютеры являются настолько обычным явлением, что люди не могут понять, что они используют компьютер, когда они программируют свой GPS или запускают приложение на своем смартфоне.

Наш взгляд сегодня

Компьютеры обычно легко идентифицировать. Они были чудовищно дорогими устройствами, которые весили тонны, заполняли комнату или больше пространства, требовали чистой среды в помещении и кондиционирования воздуха. Люди фактически не прикасались к компьютеру; они взаимодействовали с компьютером через терминалы и сети. С появлением персональных компьютеров, стали доступными вычислительные машины для физических лиц, и многие люди стали иметь компьютеры в своих собственных домах. Телекоммуникация через локальные сети или по телефонным сетям позволила людям соединить свои компьютеры вместе, общаться друг с другом и обмениваться сообщениями электронной почты, файлами, программами и т.д. Интернет, который впервые был включен в его ранней форме в 1969, стал коммерчески доступным для пользователей домашних компьютеров в середине 1990-х годов. В начале этого периода, люди подключались к Интернету через медленный модемный доступ по их телефону. Но за последние 15 лет, телекоммуникации полностью изменились. Теперь у нас есть беспроводной доступ, высокоскоростные Интернет-соединения, сотовые телефоны и многое другое.

Сегодня компьютеры нелегко идентифицировать. Они больше не ограничены компьютерами обычных размеров или настольными устройствами. Вы можете иметь сетевой компьютер или лэптоп, ноутбук, планшетный компьютер, карманный компьютер. У нас даже есть устройства меньшего размера, чем ручные устройства, использующие процессоры и память. И наше подключение изменилось соответственно. Ваш доступ к средствам связи уже не ограничивается телефонным портом в вашем доме. С помощью беспроводной связи, вы можете получить доступ в любом месте в вашем доме или где-нибудь в отдаленном месте, которое имеет точки доступа. Хотите кофе-брейк? Нет проблем, идите в кафе Старбакс, и вы все еще можете получить доступ к Интернету через но-

утбук. Или желаете на прогулку? Вы все еще можете получить доступ к Интернету через мобильный телефон (до тех пор, пока вы находитесь в разумной близости к башне сотового телефона). Мы представляем объединенный мир через почти мгновенную связь независимо от того, где мы находимся. И мы представляем мир миллиардов на миллиарды процессоров. Мы привычно считали компьютеры миллионами, но сегодня, есть десятки миллиардов процессоров и большинство из них могут взаимодействовать друг с другом.

Такое постепенное введение в ИТ будет служить в качестве отправной точки в этом тексте. На протяжении последующих глав мы будем изучать множество концепций, связанных с ИТ. Сначала мы посмотрим на компьютерные компоненты, получим понимание того, что они делают, как они работают, и как мы подключаем их к компьютеру. Мы также изучим тему, связанную с двоичными числами, и как мы используем двоичный файл. Последующие две главы по компьютерной организации и двоичному счислению часто являются материалом, подпадающим в учебные программы по информатике. Они включены здесь, поскольку, как ИТ-специалист, вы знаете больше, чем процессор и материнская плата, когда дело доходит до компьютерной техники. Имея прочный фундамент, что делают компьютерные компоненты и как они работают, вы должны быть в состоянии понять необходимость, когда увеличить оперативную память, или как отличить процессор. Двоичное счисление в этом тексте включено в основном для поддержки концепций, обнаруженных в компьютерных сетях.

Фокус смещается в сторону системного программного обеспечения, то есть операционных систем. Рассмотрим две наиболее распространенные платформы операционных систем: Windows (Windows 7) и Unix (Red Hat Linux).

Мы будем сравнивать и противопоставлять их, как они выглядят, как мы их используем, и как мы их настраиваем. Темы операционных систем включают в себя файловые системы, пользователей, учетные записи и разрешения, процессы и управление процессами и услугами. Мы также рассмотрим две конкретные темы относительно Linux: оболочки Bash и использование регулярных выражений в Linux.

В тексте рассматриваются несколько различных историй. Эволюция компьютерных аппаратных средств, эволюция операционных систем, эволюция компьютерного программирования, истории как Linux и Windows, так и истории Интернета все было охвачено (хотя и не в той же главе). Хотя это, возможно, не является необходимым для ИТ-специа-

листа, оно помогает установить фон к тому, как изменилась технология так, что вы будете иметь представление о быстроте изменений в ИТ. Кроме того, понимание прошлого может помочь вам понять, куда ИТ может привести.

Окончательный набор глав охватывает другие темы ИТ. Компьютерные сети рассматриваются из нескольких различных точек зрения. Логическая структура сети, физическая природа сети, сетевые протоколы, которые предписывают, как переданные данные должны быть обработаны, и некоторое из наиболее распространенного сетевого программного обеспечения – все рассматриваются в одной главе. Управление программным обеспечением описывает типы доступного программного обеспечения и содержит подробную информацию о том, как установить программное обеспечение в компьютерной системе с открытым исходным кодом, в частности, в Linux. Еще одна глава посвящена программированию, предлагая примеры написания скриптов как в оболочке Linux, так и в DOS. Предпоследняя глава текста охватывает информационную сторону. В этой главе мы рассмотрим такие понятия, как управление информацией и информационное обеспечение, и безопасность. Заключительная глава оборачивает текст, рассматривая карьеру в области ИТ и различные темы, связанные с ИТ-специалистами.

Дальнейшее чтение

Существует целый ряд веб-сайтов, предоставляющих информацию об ИТ-карьере, некоторые из них перечислены ниже.

http://www.wetfeet.com/careers-industries/careers/information-technology
http://www.cio.com/article/101314/The_Hottest_Jobs_In_Information_Technology
http://www.careeroverview.com/technology-careers.html
http://www.techcareers.com/
http://information-technology.careerbuilder.com/

Лучший источник для ИТ-образования можно найти с помощью специальной группы интересов по ИТ-образованию (SIGITE) в http://www.sigite.org/it-model-curriculum.

Общие сведения по компьютерной технике, программному обеспечению и пользователям могут быть найдены в любом количестве текстов

компьютерной грамотности, таких как эти:

• Beekman, G. and Beekman, B. Tomorrow's Technology and You. Upper Saddle River, NJ: Prentice Hall, 2008.
• Fuller, F. and Larson, B. Computers: Understanding Technology. St. Paul, MN: ECM Paradigm Publishing, 2010.
• Meyer, M., Baber, R., and Pfaffenberger, B. Computers in Your Future. Upper Saddle River, NJ: Prentice Hall, 2007.
• Laberta, C. Computers Are Your Future. Upper Saddle River, NJ: Prentice Hall, 2011.
• Williams, B. and Sawyer, S. Using Information Technology. New York: McGraw-Hill, 2010.
• Snyder, L. Fluency with Information Technology: Skills, Concepts and Capabilities. Upper Saddle River, NJ: Prentice Hall, 2010.

Тем не менее, как лицо, которое хочет сделать карьеру в ИТ, вам бы лучше ознакомиться с более подробным материалом. Такие тексты будут перечислены в последующих главах, где мы рассмотрим материал большее подробно. Смотрите дополнительные показания в главе 2 для получения дополнительной информации о компьютерных аппаратных средствах, в главе 4 для получения дополнительной информации об операционных системах, главе 14 для получения более подробной информации о программировании, а в главе 16 для получения дополнительной информации об ИТ-карьере.

Глава 2. Организация компьютера и комплектующие

В этой главе рассматриваются аппаратные средства компьютера: центральный процессор (ЦП), память, подсистема ввода / вывода, а также шину . Каждый из этих компонентов описан подробно, чтобы обеспечить понимание его роли. При рассмотрении ЦП, глава подчеркивает цикл выборки-выполнения. Пример программы используется для иллюстрации цикла выборки-выполнения. Компоненты процессора исследованы отдельно: блок управления, арифметическое логическое устройство и регистры. При изучении памяти вводится иерархия памяти, а также обсуждаются различные формы памяти: ОЗУ, СОЗУ, ДОЗУ, ПЗУ, виртуальная память. При обсуждении устройств ввода и вывода, подчеркнуты темы взаимодействия человека с компьютером (HCI) и эргономика. Глава включает в себя обсуждение вопроса о том, как собрать различные компоненты компьютера. Цель этой главы заключается в том, чтобы обеспечить ИТ-студента прочным фундаментом в компьютерной технике.
Целями обучения данной главы являются
Определить компоненты компьютера и их роли.
Охарактеризовать цикл выборки-выполнения.

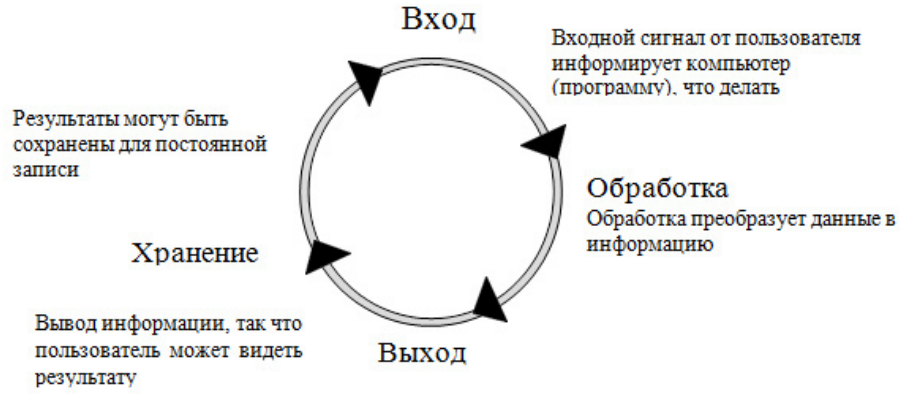


Рисунок 2.1
Цикл BOBX.

Обсудить характеристики, которые влияют на производительность процессора.
Описать различные уровни иерархии памяти.
Описать роль различных типов устройств ввода и вывода.
Обсудить влияние, которое устройства ввода / вывода имеют на тело человека и важность HCI.
Проиллюстрировать, как компьютер можно собрать из составных частей.

Компьютерная организация – это изучение компонентов компьютера, их функции, их структуры (как они работают), и как они связаны друг с другом. Эта тема является общей в программах компьютерных наук как на втором курсе, так и на должности младшего уровня.

Здесь, в то время как мы будем рассматривать эти компоненты, мы будем рассматривать больше их функции, и как они связаны друг с другом. Изучая вычислительную организацию, вы получите представление о том, как работают компьютеры, а также узнаете важность каждого из основных компонентов, введенных в главе 1. Эта глава завершается разделом, посвященным компьютерной технике и процессу сборки (построения) компьютера из составных частей.
Компьютер состоит из центрального процессора (ЦП), памяти и хранилища данных, а также периферийных устройств (включая некоторые формы подключения к сети). Компьютер выполняет четыре операции:

1. Ввод
2. Обработка
3. Вывод
4. Хранение

BOBX цикл ввода-обработки-вывода-хранения описывает примерно, как мы используем компьютер. Мы вводим данные, обрабатываем их, выводим результаты и сохраняем любую информацию, которую мы хотим сохранить навсегда. Рисунок 2.1 иллюстрирует цикл BOBX. Описание компьютера на этом уровне на самом деле не говорит нам, что происходит, например, каким образом хранение отличается от памяти, а также как происходит обработка? Таким образом, мы будем более внимательно обсуждать это здесь.

Структура КОМПЬЮТЕРА

На рисунке 2.2 показана общая структура большинства компьютерных систем. Как вы можете видеть, есть четыре компонента: центральный процессор (или процессор), память, подсистема ввода/вывода (В/ВЫВ), и шина. Процессор представляет собой устройство, которое не только выполняет команды своих программ, но и команды различных компонентов в компьютере. Память хранит запущенную программу(-ы) и данные, которые использует каждая программа. Подсистема ввода/вывода включает в себя все периферийные устройства (ввод, вывод, хранение, сеть), где хранения состоит из долгосрочных устройств хранения данных (жесткие диски, ленточные).

Структура современных компьютеров.

Шина представляет собой устройство, которое позволяет перемещать информацию между каждым компонентом.
Прежде чем мы переведем наш взгляд на компоненты компьютера, мы должны понимать, что устройства на компьютере представляют собой цифровые устройства. Внутри компьютера, цифровые данные представлены в виде электрического тока, существующего в одном из двух состояний: высокий или низкий. Высокий означает, что существует заметный ток, протекающий через компонент, а низкий означает, что протекающего тока нет, или почти нет. Мы назначим номер 1 на высоком уровне тока

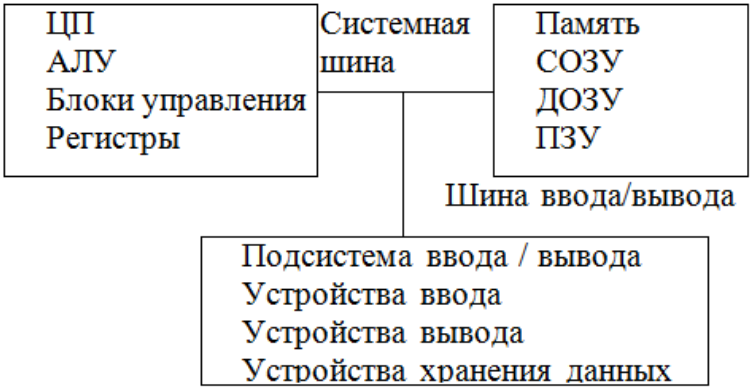


РИС 2.2
СТРУКТУРА СОВРЕМЕННОГО КОМПЬЮТЕРА

и 0 при низком (или при отсутствии) токе. Шина тогда, возможно, будет легким устройством для понимания. Шина состоит из нескольких проводов, каждый провод позволяет 1 биту (один 1 или 0, высокий или низкий ток) течь через него. Мы обсуждали емкость запоминающего устройства еще раньше, в главе 1 (таблица 1.4), но на данный момент, мы определим три термина: бит (одиночный 1 или 0), байт (8 бит, используя восемь проводов на шине, как правило, наименьшая единица передачи данных), а также слово (сегодня, компьютеры имеют либо 32-битные или 64-битные слова, размер слова — это типичный размер данных).

Шина на самом деле состоит из трех частей: шина адреса, шина управления и шина данных. Эти три части шины выполняют следующие операции:

Шина адреса используется ЦП для отправки адреса либо в память или в подсистемы ввода/вывода. Адрес — это место, из которого процессор хочет извлечь данные, или место, в котором процессор хочет сохранить данные. Процессор более часто обращается к памяти, чем к В/ВЫВ. Обратите внимание, что данная величина может быть либо значением, которое должно быть использовано в программе, либо командой программы.

Шина управления используется процессором для рассылки команд. Команды могут быть "чтение" или "запись" в память или "ввод", "вывод", или "ты доступный" для устройства ввода/вывода. Шина управления также используется различными устройствами в компьютере, чтобы посылать сигналы обратно в ЦП. Первичный сигнал известен как прерывание, которое представляет собой запрос с помощью устройства (например, дисковод, принтер), чтобы прервать ЦП и попросить его внимание.

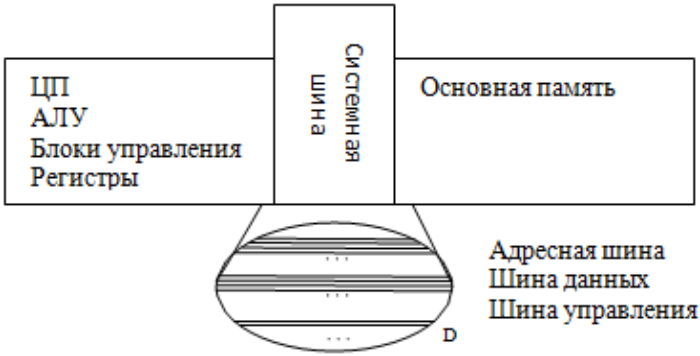


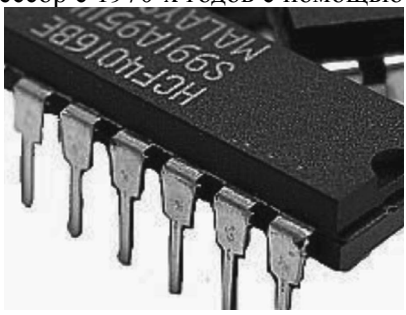
РИСУНОК 2.3
Системная шина подключения процессора и памяти.

Шина данных используется для передачи данных (в том числе программных команд) из памяти в процессор, или данных от процессора к памяти, или между устройствами ввода/вывода и процессором или памятью.

Размер шины данных – это, как правило, размер слова компьютера. Размер адресной шины основан на размере адресуемой памяти. Размер шины управления основан на количестве различных команд, которые процессор может послать вне. Рисунок 2.3, где шина данных, вероятно, будет в размере 32 бита, а также адрес и размер шины управления будет зависеть от целого ряда различных вопросов. Шина, показанная на этом рисунке, является системной. Другая шина, местная шина, соединяет компоненты ЦП [АЛУ (арифметическо-логическое устройство), блок управления, регистры] и еще одна шина соединяет устройства ввода / вывода вместе.

Что такое штифты?

Взгляните на ЦП. Вы увидите несколько металлических штифтов, которые выходят из нижней части. Штифты используются для подключения процессора в сокет. Гнездо подключается к системной шине. Таким образом, каждый штырь подключается к одной линии шины. Более ранние процессоры имели несколько штифтов, в то время как сегодня процессоры могут содержать больше тысячи штифтов! Таким образом, штифты должны быть очень маленькими, чтобы соответствовать. Кроме того, каждый штифт требует определенного количества энергии, так что ток может течь через любой или все контакты в любое время. Таким образом, наши современные процессоры требуют большего потребления энергии. Это, в свою очередь, дает больше тепла, так что нам нужно более мощное охлаждение центрального процессора. Ниже приведены два процессора Intel, ранний процессор с 1970-х годов с помощью всего



лишь нескольких десятков булавок, и современный процессор Pentium с почти 500 штифтами.

Программа

Для того чтобы понять ЦП, мы должны понимать компьютерную программу. Это объясняется тем, что процессор выполняет программы. Таким образом, мы сначала рассмотрим простую программу. Следующая программа написана на С, хотя она может быть написана на почти любом языке. Единственное, что изменяется это синтаксис. Знаки // указывают комментарии. Комментарии помогут вам разобраться в программе.

```
#include <stdio.h>        //input/output library
void main( )      //start of the program
{
int a, b, c;          //use 3 integer variables scanf("%d", &a);          //input a
scanf("%d", &b);          //input b
if(a < b)          //compare a to b, if a is less then b
c = a + b;          //then set c to be their sum
else c = a-b;       //otherwise set c to be their difference
printf ("%d", c);          //output the result, c
```

Эта программа вводит два целочисленных значения от пользователя, и когда первое меньше второго, она вычисляет c = a + b (c – это сумма двух), в противном случае, она вычисляет c = a - b (c минус b). Затем она выводит c. Если, например, пользователь вводит в 5 и 10, он вычисляет c = 15 и выводит 15. Если вместо этого пользователь вводит 5 и 3, она вычисляет c = 2 и выводит 2. Это довольно бессмысленная программа, но она будет служить нашим примером.
После написания, что происходит с программой С? Могу ли я запустить эту программу? Нет. Ни один компьютер не понимает языка программирования С. Вместо этого я должен запустить специальную программу под названием компилятор. Компилятор переводит программу на более простой язык, называемый машинный язык. Версия машинного языка программы может быть выполнена. Машинный язык является архаичным и очень трудно читаемым, поэтому мы будем смотреть на промежуточную форму программы, на *языке ассемблера*. Опять же, комментарии будут происходить после // отметок.

```
Input 33          //assume 33 is the keyboard, input a value
//from keyboard
Store a  //and store the value in the variable a Input 33    //repeat   the   input
for b
Store b
Load a  //move a from memory to CPU, a location
//called the accumulator
Subt b   //subtract b from the accumulator
//(accumulator = a – b)
Jge else //if the result is greater than or equal
//to 0, go to location “else”
Load a  //otherwise, here we do the then clause,
//load a into accumulator
Add b   //add b (accumulator is now a + b) Store c          //store  the  result  (a
+ b) in c

            Jump next       //go to the location called next
else:       Load a          //here is the else clause, load a into the
                            //accumulator
            Subt b          //subtract b (accumulator is now a – b)
            Store c         //store the result (a – b) into c
next:       Load c          //load c into the accumulator
            Output 2049     //send the accumulator value to the output
                            //device 2049, assume this is the monitor
            Halt            //end the program
```

Версия программы на языке ассемблера состоит из более примитивных инструкций, чем версия C. Например, один оператор scanf в C, который на самом деле может быть использован для ввода нескольких значений, в то время, разбивается на ввод, и хранение пар для каждого ввода. Утверждение C if разбивается на множество меньших утверждений. Теперь вспомним, что программа на ассемблере должна быть проще, чем программа на языке высокого уровня. Она проще в том, что каждая команда на ассемблере по существу делает одну примитивную вещь. Тем не менее, для наших программ для достижения чего-нибудь полезного, нам нужно гораздо больше инструкций на языке ассемблера, чем инструкции языка высокого уровня. Вы могли бы думать о двух типах языков таким образом: инструкция на языке ассемблера подобно гайкам

и болтам конструкции, в то время как обучение языку высокого уровня подобно готовым компонентам, таких как петли и замки. Компьютер не может выполнить что-то вроде scanf, или if–else утверждение непосредственно за одну команду, но он может сделать загрузку, добавление, хранение, или прыжок за одну операцию.

Как и в программе C, компьютер не может выполнить программу на языке ассемблера. Таким образом, наша программа затем должна быть преобразована из языка ассемблера в машинный язык. Причина, по которой мы показали программу на ассемблере, это то, что ее легче понять, чем программы на машинном языке.

Компилятор представляет собой программу, которая преобразует исходную программу, написанную на языке высокого уровня, таких как C, в машинный язык. Ассемблер — это программа, которая переводит программу на языке ассемблера в машинный язык. Итак, представьте, что мы перевели предыдущую программу ассемблера в машинный язык. Часть окончательной программы может выглядеть следующим образом:

```
1000100    00000000000000000000100001    —input  (from keyboard)
1000111    00100110001001011010100 01    —store  the datum in a
1000100    00000000000000000000100001    —input  (from keyboard)
1000111    00100110001001011010100 10    —store  the datum in b
```

Остальная часть программы на машинном языке отсутствует, поскольку, оно не имело бы никакого смысла больше, чем вышеуказанный перечень 1 и 0. Если вы посмотрите на 1 и 0, или двоичные числа, можно заметить, что они отформатированы на две части. Эти две части представляют собой работу каждой команды (первые 7 бит) и операнд или данные. Здесь операция означает один из, возможно, 100 различных инструкций. 1000100 представляет "ввод" и 1000111 представляет собой "хранение". Там так же было бы операции для "сложить", "SUBT", "JGE", "нагрузка", и "прыгать". Операнд обозначает ячейку памяти, написанную в двоичной системе (например, Элемент данных хранится в месте 00100110001001011010100 01, который является адресом 5,000,017, а значение00000000000000000000100001 обозначает номер устройства, 33, для клавиатуры). Хотя этот код машинного языка приведен для примера, он не слишком отличается от того, который мы могли бы найти, если бы мы изучали конкретные машинные языки.

Запуск программы

Ладно, вернемся к компьютерной организации. Мы хотим запустить эту простую программу. Первое, что происходит, когда вы хотите запустить программу это то, что операционная система загружает программу, которая хранится на жестком диске в свободную часть памяти. Будем считать, что операционная система разместила ее на ячейку памяти 5000000. Потому что есть 17 инструкций, программа будет храниться последовательно в ячейках памяти 5,000,000 до 5,000,016. Мы также будем считать, что переменные, a, b, c, сохраняются сразу после программы, в ячейках памяти 5000017 до 5,000,019, соответственно, для a, b и c. После загрузки операционная система передает управление на процессор (ЦП), чтобы начать работу этой программы. Теперь мы должны понять более подробно, что процессор и память делают. Процессор состоит из блока управления, АЛУ и регистров. Один из этих регистров называется программный счетчик, или ПК (не следует путать с общим названием домашнего компьютера). Персональный компьютер получает адрес первой команды программы в памяти, 5000000. Другой регистр называется регистр команд (РК). Он хранит текущую инструкцию. Другой регистр называется флаги состояния (ФС); он сохраняет результат самого последнего вычисления АЛУ с точки зрения того, был ли результат положительный, отрицательный, нулевым, вызвал переполнение, вызвал прерывание, имел значение четного паритета, и так далее. Каждый из этих элементов хранится в 1 бите, поэтому ФС будет хранить несколько результатов, хотя большинство из битов будет равно 0. Другие регистры данных регистров, они хранят данные, которые мы в настоящее время используем. Один специальный регистр данных, называется аккумулятор (А), используется для хранения.
Мы готовы смотреть на то, как процессор запускает программу. Процес-

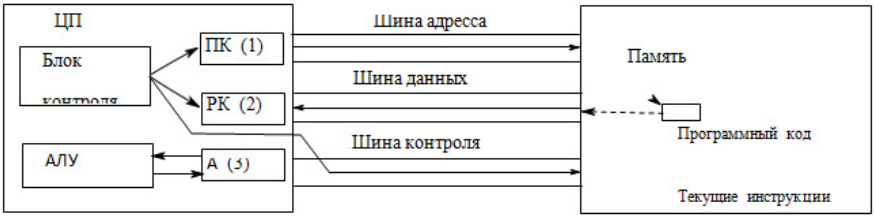


Рисунок 2.4
Три части выборки-выполнения цикла.

сор выполняет то, что известно, как цикл выборки-выполнения. Суть этого цикла заключается в том, что процессор сначала извлекает команду из памяти, а затем выполняет ее. На самом деле, есть больше, чем это. Типичный цикл выборки-выполнения (цикл отличается в зависимости от процессора) будет иметь четыре или пять, или может быть, больше, этапов (рисунок 2.4).

Блок управления перемещает ПК к адресной шине и сигналы памяти "читать" команду над управляющей шиной, память возвращает инструкция по шине данных, которые будут храниться в РК
Блок управления декодирует инструкцию в РК
Выполните инструкции в использовании опорной точки АЛУ в сети переменного тока, положить результат обратно в А

Для нашего примера, мы будем считать, цикл из пяти частей[1] со следующими этапами:

1. Получить инструкцию из памяти.
2. Расшифровать инструкцию с помощью машинного языка в микрокод.
3. Получить операнды из регистров.
4. Выполнить команду.
5. Сохранить результат обратно в регистр.

На рисунке 2.4 пункты 3-5 все указаны как шаг 3, выполняя команду с использованием регистра данных. Тем не менее, во многих процессорах есть несколько, даже десятки регистров. Это требует, чтобы шаги 3 и 5 будут отделены от шага 4.
Шаг 4 – это шаг из нашего цикла, состоящего из пяти частей, где выполняется команда. Однако, как вы видите, это не единственный шаг, необходимый для выполнения операций на языке ассемблера. Без всех этапов, программа не работает правильно. Микрокод, упомянутый в пункте 2, будет обсуждаться далее в этой главе.
Итак, давайте посмотрим, как программа, обсужденная в предыдущем разделе, будет выполняться в этом пяти-частичном цикле. Первое, что происходит, является то, что процессор осуществляет выборку первой инструкции программы. Это происходит путем отправки значения, хранящегося в ПК (ячейку памяти первой команды) к памяти по адресной

1 Разные процессоры имеют различные длины для их выборки-выполнения циклов, от нескольких до десятков

шине, и отправка сигнала "read" по шине управления. Память принимает как адрес и команду "read", и выполняет доступ для чтения к данной ячейке памяти. Чтобы ни нашлось, затем отправляется обратно в центральный процессор по шине данных. В этом случае то, что отправляется обратно в центральный процессор, не являются данными, а инструкцией программы, и в случае нашей программы, это команда "Input", написанная в двоичном виде:

1000100 +00000000000000000000100001

Как только команда получена с помощью ЦП, она хранится в РК. Для завершения этой стадии, процессор увеличивает ПК к 5,000,001, так что теперь указывает местоположение для следующей извлекаемой инструкции.

Следующим этапом является декодирование инструкции. Блок управления прерывает инструкцию на две или более частей-операций (в данном случае первые 7 битов) и операнда (-ов). По существу, блок управления учитывает таблицу операций, чтобы найти 1000100, команду "Input". Это информирует процессор для выполнения инструкции, он должен выполнить операцию ввода от устройства ввода, заданной по адресу в операнде. Операнд является двоичным числом 33, которым (в нашем фиктивном компьютере) является клавиатура.

Поскольку нет операндов для извлечения, этап 3 пропускается. Этапом 4 является выполнение команды. Инструкция ввода требует, чтобы процессор обменивался данными с устройством ввода (клавиатуры) и получить следующие введенные данные. Выполнение команды ввода не является типичным в том, что процессор не продолжит, пока пользователь не введет что-то. В тот момент, центральный процессор принимает данные по шине данных и приводит их в ЦП. Пятая и последняя стадия требует перемещения значения из шины данных в регистр А.

Теперь мы увидели полное выполнение первой команды нашей программы. Что будет дальше? Весь цикл выборки-выполнить повторяется. На самом деле, он будет продолжать повторять, пока программа не завершит свою работу.

Для второй команды, первый шаг должен выбрать команду из памяти. Теперь компьютер имеет адрес 5,000,001, поэтому процессор извлекает команду в этом месте из памяти. Значение ПК помещается на шину адреса, блок управления сигнализирует о считывании памяти через шину управления, память просматривает этот адрес и возвращает содержи-

мое по шине данных, и элемент будет сохранен в ИК. Последним шагом фазы выборки является приращение ПК к настоящей точке 5,000,002. Инструкция в ИК

1000111 +00100110001001011010100001

является "сохранить а".

Стадии декодирования определяет, что операция является операцией сохранения, которая требует перемещения данных из АС в основную память. Адрес, который получит данные из АС хранится в качестве второй части инструкции, 5000017 (это адрес).

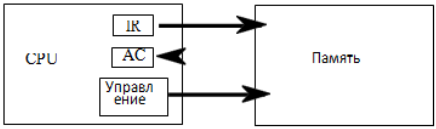
Для выполнения инструкции, последняя часть ИК перемещается на шину адреса, значение в АС перемещается к шине данных, а блок управления сигнализирует памяти о "записи" по шине управления. Выполнение команды теперь в руках основной памяти, которая принимает значение от шины данных и сохраняет его в ячейку памяти, полученной по адресной шине (5,000,017). Эта команда не требует пятого шага, так как процессору саму по себе не нужно хранить что-нибудь.

Следующие две команды встречаются в почти идентичном образе, за исключением того, что эти инструкции на 5,000,002 и 5,000,003, соответственно, а вторые данные хранятся в месте памяти 5,000,018 (б). К тому времени эти две команды были выполнены, ПК будет указывать на местоположение 5,000,004 и ячейки памяти 5,000,017 и 5000018 будут хранить первые два входных значения, а, б, соответственно.

Пятая команда извлекается и хранится в ИК. Эта инструкция "загрузки а". Эта инструкция посылает адрес, 5,000,017, по адресной шине и сигнал чтения памяти по шине управления. Память ищет данные, хранящиеся в этом месте, и посылает их обратно по шине данных. Заключительным этапом этой инструкции является хранение полученных данных в АС. Обратите внимание на то, что в отличие от предыдущей операции "чтения из памяти", описанные в данном разделе, это чтение истинных данных, а не инструкции. Рисунок 2.5 иллюстрирует разницу между инструкциями "загрузка" (верхняя часть рисунка) и "хранение" (нижняя часть рисунка), то есть, между памятью для чтения и записи в память.

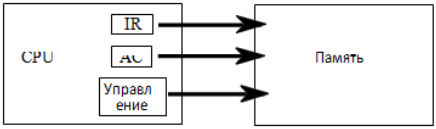
Шестая инструкция, "SUBT B", начинается похожей на пятую "загрузку". Команда выбирается, как и прежде, с "Subt B", помещенной в ИК и ПК увеличивается на единицу. Блок управления затем декодирует эту инструкцию. Тем не менее, этап выполнения для вычитания отличается от загрузки, так как эта команда требует двух отдельных шагов выполне-

ния. Во-первых, память читается, похоже на "загрузку", но в этом случае адрес является 5000018 (переменная б). Данные, которые возвращаются из памяти не хранятся в AC, хотя вместо этого содержимое из AC и данные, возвращенные из памяти и отправлены в АЛУ. Блок управления сигнализирует АЛУ на выполнение вычитания. Схема вычитания работает на двух данных и результат, выходящий из АЛУ, является значением из AC минус значение из памяти (т.е. A - B). Шагом сохранения результата данной инструкции является принять этот результат из АЛУ и переместить его в AC.
АЛУ также устанавливает флаг состояния на основе результата вычитания. SF используется для обозначения результатов операций ALU, например, был ли результат положительным, нулевым или отрицательным,



CPU - центральный процессор
IR - регистр интерфейсов
AC - регистр управления выборкой
Control Unite- устройство управления (блок управления)

1. Адрес передается из регистра интерфейсов в блок управления шиной, сигналы памяти считывают блок управления шиной.
2. Память получает доступ к адресу, возвращает данные через шину.
3. Данные сохранены в регистре управления выборкой.



1. Адрес из регистра интерфейсов передается в шину данных из регистра управления выборкой через блок управления шиной, сигналы памяти записываются через блок управления шиной.
2. Память получает доступ к адресу, хранящему данные, переданные из шины данных в определенное место памяти.

Рисунок 2.5
Чтение из памяти (вверху) в зависимости от записи в память (внизу).

или вызван ли в результате переполнения или выполнения, или прерывания. Флаги, как правило, обозначаются как PF (положительный флаг), ZF (флаг нуля), NF (отрицательный флаг), OF (флаг переполнения), и так далее. Любая операция АЛУ приведет к по меньшей мере, одному набору флагов (меняются от 0 до 1). Мы увидим, как эти флаги используются в следующей инструкции.

Седьмая инструкция (на 5,000,006) является первой инструкцией, путем чтения кода на ассемблере. Она говорит: " jge else ". Это означает, что " прыгать на большее или равное место другому ". Расположение "else" это другое место в памяти. Поскольку первая команда в программе в ячейке памяти 5000000, то "else" положение находится в 5,000,011 (это 12-я инструкции в программе). Вернитесь к программе в предыдущем разделе. Как и все предыдущие инструкции, эта команда (" jge else ") извлекается из памяти в ИК и декодируется. Этап выполнения работает следующим образом:

Если какой-либо PF или ZF устанавливаются (вычитание привело к положительному или нулевому результату), а затем сбрасывает ПК к месту " else ". Если NF установлен, то ничего не делать.

Это означает, что, если предыдущая инструкция (вычитание) привела к положительному или нулевому результату, то ответвление на место " else ". Мы используем PF и ZF здесь, так как команда определяет "больше или равно". Если бы команда была "меньше чем", мы будем использовать только NF. Другие сравнения включают "равно", "не равно", "больше чем" и "меньше или равно". В конце этой инструкции, компьютер будет либо 5,000,007 (следующая последовательная инструкция) или 5,000,011 (расположение else).

Восьмая инструкция затем зависит от того, что произошло в седьмой. Либо процессор получает инструкцию на 5,000,007 или 5,000,011. В любом случае следующие три инструкции почти идентичны. Они являются "загрузка", "добавить / Subt B", и "сохранить c". То есть, группа из трех инструкций выполняет либо c = a + Ь или c = a - Ь в зависимости от того, какой из двух наборов кода выполняется. Если предположить, что мы выполнили инструкции, начиная с 5,000,007, то инструкция по 5,000,010 является " jump next ". Эта команда изменяет значение ПК, чтобы быть 5000012 (расположение рядом). Какой бы путь не привел нас сюда, последними тремя инструкциями являются "загрузка c", "вывод", "стоп". Команда вывода принимает все, что находится в АС и перемещает его на устройство вывода из списка (2049, предположительно, монитор). Когда процессор выполняет команду "Стоп", программа завершается и управ-

ление возвращается к операционной системе.

Роль центрального процессора

Процессор обрабатывает наши программы. Он делает это с помощью двух различных частей аппаратного обеспечения. Первым из них является АЛУ, которое выполняет все арифметические и логические операции. АЛУ имеет индивидуальные схемы для выполнения различных операций. Сумматор используется для выполнения как сложение, так и вычитание. Множитель используется для умножения. Делитель используется для деления. Компаратор используется для сравнения двух значений (например, чтобы определить, есть ли а > б). Другие схемы выполняют сдвиг, вращение и вычисления четности (см главу 3).

Второй частью оборудования является блок управления. Блок управления отвечает за контролирование (командование) всех компонентов компьютера. Как мы видели в примере выполнения программы, блок управления посылает такие сигналы, как чтение памяти или запись в память. Он также посылает сигналы на АЛУ, такие, как выполнить вычитание или сравнение. Блок управления контролирует цикл выборки-выполнения. Во-первых, он выполняет этап выборки. Затем он декодирует сгруженную инструкцию в микрокод. Это, в свою очередь, дает команду блоку управления о том, как выполнить команду. Таким образом, блок управления будет иметь разделенные наборы команд для этапа выборки команды, и для каждой машинной команды. Некоторые инструкции требуют операнда, и поэтому блок управления обрабатывает как получит операнд (этап 3 из цикла выборки-выполнения, описанный в выполнении программы). Некоторые инструкции требуют ALU, поэтому блок управления информирует управлять надлежащую схему АЛУ. Некоторые инструкции требуют, чтобы результат был сохранен (этап 5 из цикла выборки-выполнения из выполнения программы), так что блок управления перемещает данные из источника в АС (или другой регистр).

Микрокод — это конкретные операции, которые должны проходить в рамках данного такта. Например, в начале выборки, первый шаг, чтобы переместить значение из ПК в память адресной шины и сигнал памяти для выполнения чтения. Этими двумя командами являются только два действия, которые имеют место в течение этого такта. Микрокод для этого шага будет бинарный перечень фактические компоненты, которых должны получить команду. Бинарный листинг почти полностью состоит из нулей, потому что, как мы видим с выборки команды, только два

компонента из всего ПК делают что-нибудь. Так что микрокод для этого шага имеет две 1, а остальные являются 0. Таким образом, микрокод выглядит так же, как машинный язык. Тем не менее, хотя наш пример машинного языка в программе А состоял из 32-битных инструкций, наши инструкции микрокода являются такой длины, какой существуют компоненты в компьютере. Например, если есть 50 различных компонентов для команды, инструкции микрокода будут длиной 50 бит. Вот, давайте представим, что компьютер имеет 50 компонентов к команде, и что команда переместить значение ПК к шине адреса является управляющим сигналом 0 и сигнал в память для выполнения чтения является управляющий сигнал 15. микрокод для этого шага должен был бы:

10000000000001000000000000000000000000000000000000

Одну команду микрокода иногда называют микро-операция. Существует различные микро-операции для каждого шага цикла выборки-выполнения, плюс одна или несколько микроопераций для каждой инструкции на машинном языке. Напомним, что шаг 2 из цикла выборки-выполнения должен был преобразовать машинную команду (например, "загрузка") в микрокод. После того, как это делается, выполнение шага является лишь вопросом блока управления отправляя сигналы, где каждый бит в микро-операции обозначает бит, который будет разослан через другую линию шины управления. Эта тема, очень продвинутая , и если вы этого не понимаете, не беспокойтесь, потому что это не имеет значения для материала в данном тексте.

Мы уже говорили о различных регистрах. Существуют два различных класса регистров: те, которые используются блоком управления и те, которые используются для хранения данных для АЛУ. Блок управления использует ПК (программа счетчик), ИК (регистра команд), SFs, указатель стека, и, возможно, другие. Хотя наш пример в программе и выполнения программы ссылается на один регистр данных, АС (аккумулятор), современные компьютеры имеют несколько, возможно сотни, регистров. Процессор Intel 8088 (используется на ранних компьютерах IBM PC), а затем процессоры Pentium используют четыре регистра целочисленных данных, которым даны имена EAX, EBX, ECX и EDX. Другие компьютеры могли бы назвать свои регистры как R0, R1, R2, и так далее. Эти регистры хранят данные для использования во время вычислений в АЛУ. Современные компьютеры имеют тактовую частоту в несколько ГГц (гигагерц). Что означает этот термин на самом деле? 1 ГГц означает 1 миллиард тактовых циклов в секунду, или что часы работают со скоростью 1 миллиардной доли секунды. Это делает его звучащим, как 1-ГГц

процессор выполняет 1 миллиард инструкций за секунду. Это неправда. Напомним, что цикл выборки-выполнить может состоять из пяти этапов или более. Оказывается, что каждый этап выборки-выполнения цикла требует, как минимум 1 такта, чтобы этот этап был выполнен.

Рассмотрим этап выборки, как описано в выполнении программы. Это требуется по крайней мере трех тактов. В первом цикле, значение ПК посылается по шине адреса в памяти и блок управления сигнализирует о считывании памяти. Во втором цикле, память возвращает команду по шине данных. В третьем цикле, инструкция хранится в ИК, и ПК увеличивается. Если мы предположим, что декодирование занимает один такт, если операнд выборка занимает один цикл, что выполнение команды занимает от одного до трех циклов, и сохранение результата занимает один цикл, то один цикл выборки-выполнения, что эквивалентно одной инструкции машинного языка, займет от шести до восьми тактовых циклов. Обратите внимание на то, что разные процессоры будут иметь различную длину циклов выборки-выполнения. Процессор 1 ГГц может иметь пятиступенчатый цикл, требующий восемь тактов на инструкцию. Процессор 2,5 ГГц может иметь 12-этапный цикл, требующий около 20 тактов на инструкцию. Что быстрее? Процессор 2,5 ГГц имеет более быстрые часы, но занимает большее количество циклов, чтобы выполнить команду. На самом деле, в данном случае, эти два варианта имеют одинаковую производительность (20 / 2,5 млрд = 8/1 млрд).

Другой способ измерить скорость процессора - это подсчитать, сколько команд может быть выполнено в течение 1 секунды. Это значение часто выражается MIPS (миллионов инструкций в секунду). Так как компьютерная графика и многие вычисления требуют использования значений с плавающей точкой (числа с десятичной точкой), то существует еще один термин мегафлопс (миллионы операций в точечных флоат ИНГ секунду с десятичной точкой).

Ни один из них, однако, не говорит всю историю нашей скорости процессора. Скорость процессора также находится под влиянием следующего:

1. Размер слова, который ограничивает размер данных перемещен в любой момент времени (размер слова, как правило, это размер шины данных и размер регистров; меньшие размеры слов обычно имеют в виду больше передачи данных по шине и продолжительные времена исполнения для вычисления больших значений)
2. Производительность кэша (см. следующий раздел), скорость памяти и размер
3. Сама программа (некоторые программы требуют ресурсов, которые

медленнее или больше время затратные, чем другие)
4. Является ли компьютер работающим с нескольким программами, по сравнению с работающим на незагруженной системе
5. Влияние операционной системы на производительность
6. Влияние виртуальной памяти на производительность
7. Многие другие вопросы
Для того, чтобы получить реальную картину производительности процессора, компьютерные архитекторы будут тестировать процессоры по эталонным программам. Производительность на какой-либо одной программы может вводить в заблуждение, поэтому тесты, которые используются, чтобы проверить процессоры - это набор программ, которые проверяют различные аспекты процессора. Только глядя на общую производительность эталона, мы можем получить хорошее понимание того, как работает процессор. Но важно здесь не быть одураченным ГГц рейтингом процессора, он говорит вам что-то о процессоре, но не так много, как вы могли бы подумать. Если вы должны были заказать компьютеры для вашей организации, вы бы лучше обслужились не просто читать упаковку и видя рейтинг ГГц, но и прочитав о производительности каждого процессора в публикациях (например, Consumer Reports, PC World) и веб-сайты (www.cpubenchmark.net, www.geek.com).

Роль памяти

В начале истории компьютеров, считалось, что память будет хранить только данные. Компьютерная программа выполняла бы ввод одной команды за один раз с перфокарт. Тем не менее, с компьютерами, такими как ENIAC (электронный цифровой интегратор и компьютер), которые способны выполнять 5000 операций в секунду, ввод команды программы с перфокарт снизит производительность ENIAC до той, которая у считывателя перфокарт. Джон фон Нейман был первым, кто произвел компьютер, хранящий программы. Идея заключается в том, что память компьютера будет хранить как программный код, так и данные программы. Таким образом, процессор не будет обременен медленным вводом, определенным считывателем перфокарты. Тем не менее, существует значительная проблема с памятью (RAM), и что в том, что скорость процессора по-прежнему существенно будет улучшаться каждый год, а доступ к оперативной памяти будет улучшаться постепенно на протяжении многих лет. Рисунок 2.6 иллюстрирует огромное улучшение скорости про-

цессора по сравнению с умеренным улучшение во времени доступа к памяти. Обратите внимание на то, что увеличение производительности, отображаемое на оси Y, находится в экспоненциальной шкале, увеличивая частями по 2. Например, в период с 1982 по 1986 год, время доступа к памяти едва увеличилось , в то время как производительность процессора выросла в четыре раза. Тогда как производительность процессора удваивается примерно каждые два года, а время как основное время доступа к памяти едва возрастает на протяжении многих лет, основная скорость доступа к памяти отстает от скорости процессора все больше и больше с каждым годом.

Процессор полагается на память, по крайней мере один раз в инструкции, чтобы выбрать команду из памяти. Некоторые инструкции требуют либо чтения из памяти (команды загрузки) или записи результатов в память (инструкции сохранения). В такой инструкции будет два доступа в память на инструкцию, одна инструкция выборки, другая загрузки или хранения. На самом деле, некоторые процессоры позволяют читать несколько операций и записывать в память каждую команду.

Перед тем как продолжить, нам необходимо определить ОЗУ. ОЗУ обозначает оперативное запоминающее устройство. Идея заключается в том, что мы посылаем информацию в блок памяти и команду чтения. Память ищет элемент и возвращает его. С другой стороны, мы посылаем блоку памяти адрес, данные и команду записи. Затем память сохраняет эти данные в этом месте. Термин "случайный" означает, что адреса будут отправлены , казалось бы, случайным образом. Этот термин также озна-

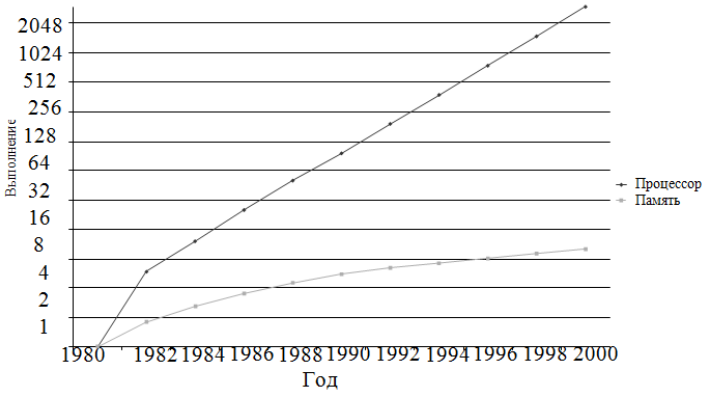


Рис.2.6

Улучшение скорости процессора с течением времени доступа к памяти.

чает, что доступ к любому адресу должен занимать такое же количество времени, независимо от того, какой адрес отправлен. Это отличает оперативную память от форм хранения, которые имеют различное количество времени в зависимости от расположения разыскиваемого пункта (ленте, например, может потребоваться немало быстрой перемотки вперед или назад, а диск занимает много времени для перемещения головки считывания / записи соответственно). К сожалению, термин ОЗУ несколько неоднозначен - есть три формы памяти, которые все квалифицируют как ОЗУ.

Мы различаем эти формы ОЗУ в качестве динамической оперативной памяти (DRAM), статическое ОЗУ (SRAM), и ПЗУ. ПЗУ память только для чтения. Это память, где информация хранится постоянно слитая в месте, так что это может быть только для чтения, а не записи. Большинство компьютеров хранят программу загрузки в ПЗУ наряду с основной системой ввода / вывода (BIOS), но немного другое. Мы не будем беспокоиться о ПЗУ в этом тексте. Из двух других форм оперативной памяти, DRAM является старшей. DRAM состоит из конденсаторов, которые могут быть миниатюрными до такой степени, что мы можем разместить буквально миллиарды на одном чипе. DRAM также очень недорога. DRAM, однако, является относительно медленной по сравнению с CPU (см обратно Рисунок 2.6). Таким образом, с одной стороны, мы имеем недорогое средство, обеспечивающее больше памяти для хранения, но с другой стороны, время доступа значительно медленнее, чем процессор (с коэффициентом от 25 до 100!).

SRAM построена с использованием нескольких транзисторов, это блоки, называемые триггерами. Мы используем SRAM для создания обоих регистров в CPU и кэш (произносится как "кэш") память. Кэш является новой технологией в вычислениях, чем DRAM, но очень затратна. SRAM намного дороже, чем DRAM, так что мы, как правило, используем ее намного меньше в наших компьютерах. Тем не менее, SRAM примерно так же быстра, как центральный процессор, и поэтому, когда процессор получает доступ к SRAM для инструкций и данных, процессору не приходится ждать, как он делает при доступе к памяти DRAM.

Это оставляет нас с интересной дилеммой. Поскольку у нас есть небольшое количество SRAM, то как мы можем гарантировать, что, когда нам нужно что-то из памяти, то оно будет перемещено (скопировано) в SRAM из памяти DRAM? Для этого мы создаем память нашего компьютера в том, что мы называем иерархия памяти. Она включает в себя несколько слоев, где каждый слой, идущий вниз будет более медленный

, менее дорогой памятью, но и гораздо большей емкости (Рисунок 2.7). В иерархии памяти, регистры являются частью процессора. Есть где-то от нескольких до нескольких десятков регистров, каждый из которых способен хранить слово, [2] поэтому процессор может иметь от нескольких десятков до нескольких сотен байт регистра памяти. Они работают примерно с той же скоростью, что и системные тактовые сигналы. Есть два разных уровня кэш-памяти: на чипе и вне чипа. Кэш на кристалле находится на процессоре, хотя не в той же самой области ЦПУ как регистры. Несмотря на то что, на чипе кэш имеет ту же технологию, как регистры, но тем не менее требуются дополнительные аппаратные средства адресации, что приводит кэш на чипе к более медленной работе, возможно, в два раза медленнее доступа к регистру. Кэш офф-чип, расположенный отдельно на материнской плате , использует ту же технологию, как кэш на чипе. Тем не менее, так как кэш вне чипа расположен на большем расстоянии от процессора, то он требует больше времени для доступа к нему, возможно, от 3 до 10 раз медленнее, чем кэш на чипе.

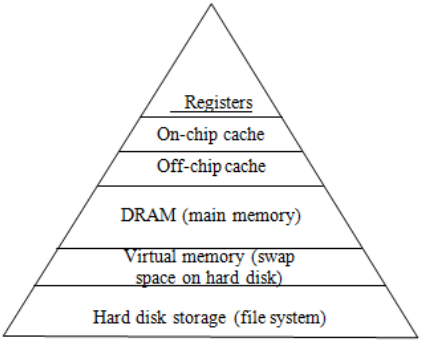


Рис. 2.7.
Иерархия памяти.

Перевод:
Registers – Регистры
On-chip cache – внутрисхемный кэш
Off-chip cache – внесхемный кэш
DRAM (Main memory) – динамическое ОЗУ (Основная память)
Virtual memory (swap space on hard disk) – Виртуальная память (пространство свопинга на жестком диске)
Hard disk storage (file system) – хранилище жесткого диска (файловая система)

2 Некоторые процессоры имеют двойные регистры, способные хранить два слова.

Поскольку кэш на кристалле находится на самом процессоре, он должен отображаться совместно с остальной частью процессора. Кэш-офф-чип может быть гораздо больше, и, следовательно, иметь гораздо больший объем памяти. Кэш на чипе может состоять только из 32 или 64 КБ, в то время как кэш офф-чип может быть от 1 до 8 МБ. Даже в этом случае, кэш вне чипа намного меньше, чем емкости DRAM, которые сегодня являются по крайней мере, 4 ГБ. Тем не менее, скорость DRAM, гораздо медленнее, до сих пор, может быть от 25 до 100 раз медленнее, чем сам процессор.

Обратите внимание на то, что иерархия памяти не заканчивается. Программы, которые мы выполняем сегодня часто намного больше, чем размер самой памяти! Как же тогда мы можем сжать их в память? Также считаю, что мы, как правило, можем запускать несколько программ одновременно. Для того, чтобы использовать большее количество памяти , чем у нас, мы расширяем основную память, используя концепцию, называемую виртуальной памяти. В виртуальной памяти, все что в настоящее время не используется отправляется в область, известную как пространство подкачки. Пространство подкачки хранится на жестком диске в отдельной области, известной как раздел подкачки. Этот раздел создан отличным от остальной части файловой системы дискового пространства - она разделена, чтобы быть быстрее, даже если разделение не является столь эффективным с точки зрения использования пространства. Но, имея ее быстро действующую , мы можем поменять местами куски памяти и по мере необходимости.

Каждый уровень иерархии памяти выступает в качестве "обратного хода" для следующего более высокого уровня. Мы начинаем путем загрузки всей программы и пространства для данных с жесткого диска (или съемных носителей) в виртуальную память и копируем в память DRAM только те части, которые необходимы немедленно (например, запуск программы). Тогда, как мы начинаем инструкции доступа и данных, они копируются в кэш на чипе и вне чипа. После того, как кэш на чипе заполнен, любые дополнительные элементы, загруженные в кэш первыми, требуют, чтобы что-то было выброшено.

По мере продвижения вверх по иерархии, пространство ограничивается еще больше, и поэтому выбраковка случается чаще. Выборка подает элементы, находящиеся в кэше, когда процессор нуждается в них. Выборка редко приводит к ухудшению качества работы. Мы имеем в виду эффективность кэш-памяти по скорости попадания (как часто то, что мы хотим найти в кэше). Удивительно, но даже при наличии небольших кэшей на

чипе, наивысшие коэффициенты могут достигать 98% до 99%. По мере продвижения вниз по иерархии, наивысшие коэффициенты улучшаются. Эти вариации, хотя и более дороги, как правило, направлены на повышение производительности.

Роль ввода и вывода

Подсистема ввода / вывода состоит из входных и выходных устройств (включая устройства хранения данных), шину для подключения устройств друг к другу и системной шине , а также интерфейсных устройств, таких как карты и порты расширения. Строго говоря, компьютер не требуется для работы устройств ввода или вывода. Компьютерная программа хранится в памяти вместе с данными программы, и результаты сохраняются обратно в память. Тем не менее, без устройств ввода и вывода, мы не в состоянии взаимодействовать с компьютером. Мы не можем сказать компьютеру, что мы хотим сделать, и мы не можем просмотреть результаты.

Самые ранние формы ввода и вывода были ограничены магнитной лентой, перфокартами и компьютерной распечаткой. Там не было никакого прямого взаимодействия; вместо этого, пользователь должен был подготовить как программный код, так и данные, используя устройство телетайп, результаты которого будут появляться на перфокартах (одну карту на инструкцию или данные). Стек перфокарт вводился через считыватель перфокарт и хранился на магнитной ленте. Лента устанавливалась , данные вводились, и программа выполнялась, а вывод сохранялся обратно на магнитной ленте. Лента будет удалена и установлена на принтер, который будет печатать выходные данные. Таким образом было не очень удовлетворительно использовать компьютер!

Сегодня, конечно же, существует большое количество устройств, доступных, чтобы сделать компьютер доступным в любой из ситуаций. Ключевое слово за I / O сегодня является простота использования. Мы имеем в виду философию дизайна этого взаимодействия, как взаимодействие человека с компьютером (HCI). В HCI, мы рассматриваем использование компьютера в качестве человеко-ориентированного , а не машиноориентированного . В таблице 2.1 перечислены некоторые из устройств ввода / вывода, часто используемых сегодня и основные причины их использования. Устройства хранения данных исключены из таблицы, но будут включать в себя внутренний и внешний жесткий диск, оптический диск, флэш-память, магнитную ленту, и сетевые запоминающие устройства.

Сетевое хранилище - это обычно жесткий диск, доступный через сеть.
В HCI, основной упор делается на более естественные способы общения с компьютером. Эта область исследования, которая приводит в отрасли информатики, психологии, дизайна и здоровья, в частности, содержит рекомендации для более доступного использования компьютера. Например, устройства вывода Брайля доступны для пользователей с нарушениями зрения. Большие мониторы и операционные системы, которые могут легко изменить разрешение экрана (размер объектов на экране) может также помочь тем, у кого нарушено зрение. Микрофоны часто используются людьми, которые не могут использовать клавиатуру и / или мышь.
HCI также фокусируется на устройствах, которые позволят снизить нагрузку на человеческий организм из-за чрезмерного взаимодействия с компьютером. Эргономика - это изучение проектирования систем и объектов, чтобы лучше соответствовать потребностям человека. Для компьютеров, они включают улучшенную клавиатуру, которые ставят меньшую нагрузку на запястья человека, лучше указывающие устройства, такие как улучшенная мышь или

Таблица 2.1 Устройства ввода и вывода

Штрих-код считывателя	Ввод информации об упаковке, используемой в основном в магазинах
Камера	Вход видеоизображения (неподвижных или движущихся)
Очки	Вывод видео изображения, используется в основном в виртуальной реальности
Джойстик	Начальное движение/ с указыванием информации , используется, в основном, для компьютерных игр
Клавиатура	Вводимый текст информации , основное средство ввода для большинства пользователей
Микрофон	Ввод речевой информации, используется в тех случаях, когда пользователь не может использовать руки или имеет значительн ое количество информации для ввода; иногда неточные
Монитор	Первичное устройство вывода информации
Мышь	Ввод данных пользователем с системой графического интерфейса пользователя
MIDI устройство	Устройство ввода музыкальных данных

Графический планшет	Ввод письменной информации , когда клавиатура является нежелательным устройством ; иногда неточные текст вывода на принтер и / или изображения на бумаге
Принтер	Выводимый текст или изображение на бумаге
Сканер	Ввод текста и/или изображения из бумаги
Динамики	Выход музыки и звука
Сенсорная панель / точка	Альтернативно -указательное входное устройство, когда мышь не требуется (из-за проблем с переносимостью или проблем со здоровьем)
Сенсорный экран	Альтернативно -указательное входное устройство, когда мышь и сенсорная панель / точка не требую тся, первичный ввод устр ойств для портативных устройств.
Трекбол	альтернативно-указательное входное устройство, когда мышь не требуется (из-за проблем с переносимостью или проблемы со здоровьем), иногда используется для компьютерных игр.

точки касания, а также улучшенную гарнитуру. Использование как клавиатуры, так и мыши привело многих людей к развитию синдрома запястного канала, один из многих форм повторяющихся стрессовых травм. Повторяющаяся стрессовые травмы возникают от выполнения подобных действий снова и снова, таким образом, что она вызывает износ на теле. Например, с помощью клавиатуры можно неправильно напрягать мышцы в запястье, что приводит к кистевому туннельному синдрому. Плохая осанка, когда сидя перед компьютером в течение нескольких часов за одно время, может привести к другим формам стресса. В закон о реабилитации 1973 года, который разрешает гранты государств, рекламирующих услуги для граждан с тяжелыми недостатками, были внесены изменения (раздел 508) в целях повышения доступности во всех формах ИТ. Нормы, установленные в разделе 508 включают в себя рекомендации по ИТ-продуктам , включая веб-сайты, программное обеспечение, операционные системы и аппаратное обеспечение. Например, все настольные и портативные компьютеры теперь должны прийти

со слотами расширения, которые поддерживают различные HCI устройства (например, микрофон, трекбол). Операционным системам необходимо разрешить изменение разрешения экрана для размещения с нарушениями зрения. Веб-сайты, поддерживаемые правительством должны быть легко проходимой, например, разрешая использование клавиши табуляции (или других горячих клавиш), а не мыши, чтобы перемещаться из одной области в другую.

Помимо секции 508 Акта реабилитации, существует множество других руководств доступности. К ним относятся национальные учебные материалы, стандарты доступа, пропагандируемые Американским фондом для слепых, UDL 2.0 Руководства от Национального Центра по универсальному дизайну для обучения, и Руководства доступности Веб контента в соответствии с рекомендациями World Wide Web Consortium. Каждый из них фокусируется на различных технологиях, но все они продвигают идеологию, что доступность является важной. Через HCI, информационные технологии стали более удобным для использования в большей популяции.

Несмотря на то, как доступность и снижение травматизма являются важными целями HCI, исследования сегодня, прежде всего нацелены на изучение, как поставить человека в центр взаимодействия компьютера. С точки зрения данных, как мы можем улучшить способ человека вводить в компьютер? Например, микрофон в сочетании с технологией распознавания речи позволил бы человеку вводить более точно и быстро, чем клавиатура? Для обработанной информации (вывод), является ли монитор лучшим устройством вывода? Будут ли 3-D очки обеспечивать более точный способ передачи информации?

Проблемы в HCI включают не только улучшенные приборы, но и усовершенствования существующих устройств. Два примера, чтобы обеспечить более высокое разрешение дисплея, чтобы сделать вывод более разборчивыми и для обеспечения избыточности на выводе, чтобы гарантировать, что выходные сигналы понятны. Другие концепции включают в себя соответствующие ментальные модели человеческого ума, предоставляя соответствующую информацию в непосредственной близости от другой информации, связанной с использованием визуального вывода, а не текста, чтобы лучше соответствовать возможности памяти человека, и так далее.

Один последний вопрос с HCI, разрешает человеку отойти от стационарного компьютера и взять вычисления с ним или с ней. Эта концепция, которую иногда называют, повсеместные компьютерные технологии, в на-

стоящее время доступны через использование портативных устройств. Тем не менее, небольшие экраны, кажется, не поддерживают понятия, перечисленные выше. Таким образом, в настоящее время разрабатываются новые устройства ввода / вывода, которые позволяют нам брать наши вычисления с собой. Эти устройства сочетают беспроводной и сотовый телефон на базе технологии с некоторыми из более продвинутых входных и выходных устройств, таких как очки для дисплеев, микрофонов для ввода, и наушники для вывода звука. Иногда называемые переносными , эти устройства, вероятно, займут передовые места в течение ближайших 5 - 10 лет.

Шина обеспечивает беспроводной доступ в Интернет. Ваши переносные устройства подключаются к домашнему компьютеру через беспроводной доступ , так что вся обработка и хранение происходит на вашем домашнем компьютере. Ваши очки представляют вам визуальную информацию, такую как текст на основе новостей или видео. Наушники не только служат, чтобы предоставить вам аудио выход, но и пытаются блокировать звуки, исходящие от внешнего мира. Ваш микрофон позволяет говорить на компьютер, чтобы контролировать то, что вы делаете. И, наконец, очки не являются полностью непрозрачными, они являются полупрозрачными, так что вы можете также увидеть внешний мир по мере необходимости (например, так что вы можете видеть, когда вы прибудете на автобусную остановку). Но очки делают больше, чем просто представляют изображение с вашего компьютера. Ваш компьютер был запрограммирован, чтобы блокировать нежелательные изображения. И так, как вы выходите из автобуса, вы не видите рекламные объявления, отображаемые на стороне автобуса, вместо того, эти изображения заменяются на белые пятна или изображения, которые вы найдете привлекательными, надуманные ли это звуки? Может быть, это сегодня, но это не будет, как HCI продолжать улучшаться.

Помимо переносных , существует целый ряд других форм новых технологий, которые стоит упомянуть. Сенсорные экраны, очевидно, большая часть нашей жизни сегодня, поскольку они являются частью всех смартфонов и планшетных устройств. Сенсорные экраны являются устройства ввода и вывода, поскольку они являются дисплеями , которые имеют датчики отмены в местах , где человек делает контакт с экраном. Сенсорные экраны существовали с конца 1960-х годов, но были слишком непомерно дорогими для использования в вычислительных устройствах. Тем не менее, на сегодняшний день, сенсорные экраны используются не только для карманных устройств интерфейсов, но и для компьютерных игр, ки-

осков и медицинских устройств, а также в скором времени могут также быть найдены в качестве стандартного интерфейса для настольных и портативных компьютеров. Современные сенсорные экраны не требуют интерфейсного пера на основе (струйной) и способны реагировать на несколько прикосновений одновременно.

Виртуальная реальность (VR) представляет собой технологию, которая все еще находится в стадии становления, хотя она родилась примерно в конце 1980-х. В ВР , компьютер создает иллюзорный мир и через различные типы устройств ввода и вывода, человек делает вклад в этом мире. Человек взаимодействует с виртуальным миром на основе движения и ориентации головы, через гарнитуру с датчиком движения и перчатки данных. Пользователь видит и слышит виртуальный мир через очки и наушники (Рисунок 2.8). Кроме того, полный костюм может предоставить пользователю различные виды ощущений.

С VR, человек может быть погружен в место, которое обычно он не может достичь, к примеру, ходьба на Марсе или поход в действующий вулкан, или плавание глубоко в океане. Практическое использование ВР включает в себя разведку, разработку и анализ (представьте себе проведение проверки самолета, который существует только на стадии разработки), образование и развлечения. Тем не менее, несмотря на потен-

Рисунок 2.8
Виртуальной реальности гарнитура и перчатки данных в использовании. (С разрешения NASA, http://gimp-savvy.com/cgi-bin/img.cgi?ailsxmzVhD8OjEo694)

циал, который ВР предлагает, не обычным явлением, потому что есть много препятствий, которые необходимо преодолеть. В первую очередь, ограничения в устройствах ввода / вывода изображают максимально реалистической иллюзию , которая оставляет желать лучшего . Кроме того, ВР требует чрезмерного количества вычислений, чтобы правильно моделировать и представить иллюзорный мир. Таким образом, ВР все еще довольно дорогое удовольствие: перчатки и гарнитура могут стоить от $ 500 до $ 5000 каждая.

И еще одна форма новых технологий ввода / вывода , которую стоит отметить - это сеть датчиков. Датчики представляют собой устройства, которые могут ощутить некоторые аспекты окружающей среды, такие как температура, давление воды, или звуковые вибрации. Большинство устройств, которые содержат датчики, как правило, имеют ограниченные датчики, расположенные в определенных местах на устройстве. Сетевой датчик представляет собой распределенную совокупность датчиков, так что устройство имеет возможность получить более реалистичный "вид" окружающей среды . Сегодня датчики достаточно дешевы, так что сенсорные сети становятся все более распространенным явлением. Мы объединяем датчики с беспроводной связью для создания беспроводной сенсорной сети, состоящей без малого из нескольких тысяч индивидуальных датчиков, каждый из которых может взаимодействовать друг с другом, и базовым компьютером для обработки чувственных данных . Сенсорные сети находятся в различных потребительских устройствах , транспортных средств, приборов контроля за состоянием здоровья, а также военной аппаратными средствами. Целая сеть датчиков может стоить всего несколько сотен долларов. Приложения для сенсорных сетей включают в себя наблюдение, мониторинг атмосферных условий (погода, парниковые газы, загрязнение воздуха), контроля за состоянием здоровья, сейсмической активности, и разведке месторождений.
Для того, чтобы обернуть это обсуждение устройств ввода / вывода, мы также должны рассмотреть два других технологий, которые улучшили возможности пользователей для подключения устройств к своим компьютерам. Это плагин и есть USB. Идея плагина состоит в том, что вы можете подключить новое устройство к компьютеру в любое время и что устройства, некогда распознаваться операционной системой, готова к использованию. Перед тем как плагин и играть, вам придется перезагрузить компьютер , как только вы подключили новое устройство таким образом, чтобы устройство могло быть признано во время процесса за-

грузки. Plug-и-игра впервые была впервые для операционной системы Windows 95, но с тех пор стал стандартом в большинстве операционных систем.

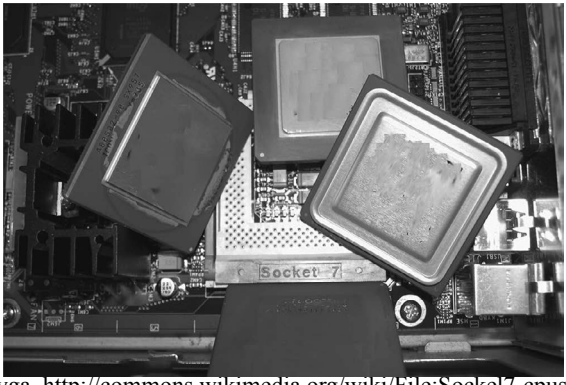
Наиболее распространенным (и не только) средства соединения нового устройства ввода / вывода к компьютеру через порт USB. USB расшифровывается как Universal Serial Bus, стандартным интерфейсом для компьютеров начиная с 1990-х годов. Стандарт USB определяет соединение, которое указывает, тип используемого кабеля способ связи между устройством и компьютером, а также блок питания, который будет существовать между компьютером и устройством. На сегодняшний день большинство устройств может быть подключен через USB, включая клавиатуры, указывающие устройства, принтеры, внешние дисковые накопители, смартфоны и другие портативные устройства. USB-диски или флэш-памяти, дает пользователю приличное количество внешних устройств хранения (1-8 Гб), что является портативным. Порт USB заменил последовательные и параллельные порты, используемые такими устройствами. Кроме того, порт USB может подавать питание к устройству.

Аппараты и сборка компьютеров (монтаж)

В этом разделе мы рассмотрим более общие компоненты компьютера и изучим базовую сборку (монтаж) на основе настольных компьютеров.

ЦПУ

Центральный процессор или процессор, хранится на одном чипе, из-



(Adapted from Gyga, http://commons.wikimedia.org/wiki/File:Sockel7-cpus.JPG.)

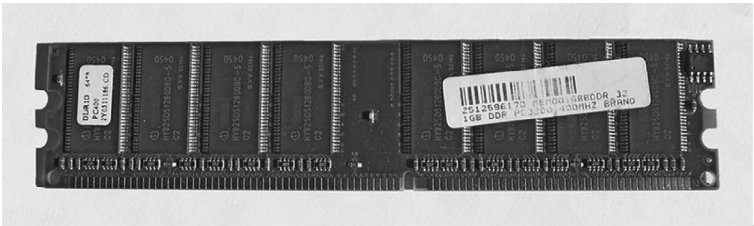
вестном как микропроцессор. Процессор имеет ряд небольших штифтов на одной стороне. Лучше всего не прикасаться к этим штырям, поскольку их сгибание, скорее всего, разрушит процессор. Вы должны вставить процессор в специальный разъем на материнской плате. Вы можете заметить, что один из углов процессора имеет треугольник. Это поможет вам расположить процессор правильно, если вставить его в гнездо.

Память

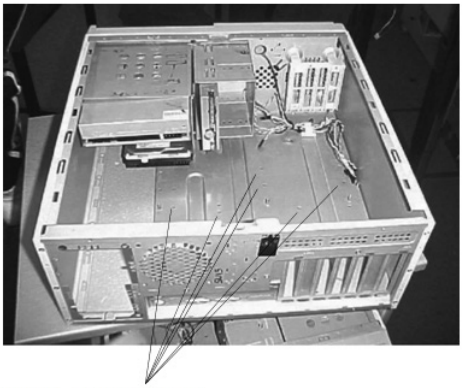
Большинство памяти компьютера хранится в оперативной памяти. Мы обычно называем это память или оперативной памятью, хотя есть и другие типы памяти в компьютере (ПЗУ, кэш). Микросхемы памяти размещены на печатной плате. Печатная плата расположена в специальном слоте на материнской плате. Опять же, вам придется разместить печатную плату правильно в слот. На рисунке, есть контакты, которые проходят вдоль нижней части этой схемы. Один набор контактов больше, чем другой; это поможет вам решить, правильное расположение в слоте памяти.

Системное устройство

Это тот случай, в котором вы будете размещать несколько ключевых компонентов компьютера. Самым важным компонентом является материнская плата. Материнская плата будет находиться на вершине "тупиков", чтобы поднять его над поверхностью. Это на самом деле будет помещен вдоль одной стороны системного блока, а не на нижнем торце. Глядя на системный блок, можно увидеть множество отверстий для винтов; некоторые из них будут иметь распорки. Вставляются в системный блок и являются блоком питания, вентилятор, а также некоторые устройства хранения данных, такие как оптический дисковод и жесткий диск. Существует возможность для других устройств в пазах вдоль передней части системного блока.



(Предоставлено Ласло Салаи, http://commons.wikimedia.org/wiki/ Файл:DDR_RAM-3.jpg.)

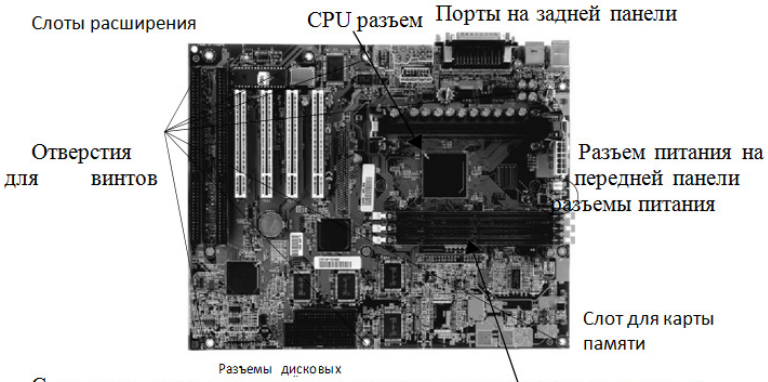


Винтовые отверстия для тупиков
(Предоставлено Норман Роджерс, http://commons.wikimedia.org/wiki/ Файл: раздел- компьютер футляр.JPG.)

Винтовые отверстия для тупиков
(Предоставлено Норман Роджерс, http://commons.wikimedia.org/wiki/ Файл: раздел- компьютер футляр.JPG.)

Материнская плата

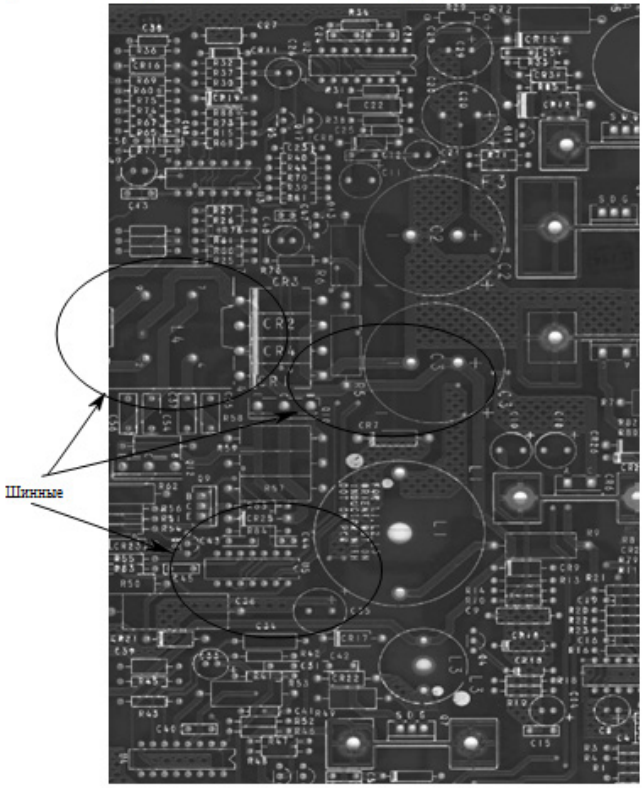
Материнская плата обеспечивает связь между процессором, памятью и другими компонентами. Эти устройства будут подключаться к материнской плате через слоты расширения и порты на задней панели. Материнская плата представляет собой часть из стекловолокна или пластика, на котором крепятся розетки и автобусные линии. Гнезда включают один (или более) разъем для процессора, гнезда для памяти DRAM, слоты



Слоты расширения процессорного разъема на заднюю панель портов

расширения для карт, которые являются интерфейсы с периферийными устройствами, разъемы для подключения устройств хранения данных, подключения питания, а также порты, которые будут придерживаться через заднюю панель системы Блок. микросхемы памяти ROM уже прикреплены к материнской плате

На системной плате имеются отверстия для винтов, так что вы можете прикрепить его к системному блоку. Несмотря на то, что системный блок, как правило, стоит вертикально, материнская плата на самом деле будет располагаться вертикально, а также, так что важно, чтобы материнская плата будет надежно подсоединена. Кроме того, тупики придают уверенность в том, что материнская плата физически не касается внутренней поверхности системного блока. Так как электрический ток протекает че-



(Предоставлено Шаттерсток /jakelv7500.)

рез материнскую плату, если коснуться металлической внутренней поверхности системного блока, это может замкнуть поток, что в результате может привести к прекращению подачи тока в соответствующие чипы. Нижняя сторона материнской платы содержит линии проводов, которые составляют часть системной шины. Ток течет по этим линиям. Поэтому важно, чтобы эти линии не имели никакого контакта с металлом. Это одна из причин, почему мы будем монтировать материнскую плату на вершине "тупиков". Это позволит установить системную плату выше от боковой стороны системного блока. Обратите внимание на то, что нижняя часть материнской платы является грубой и неровной с небольшими кусочками аппаратного содержимого, также на ней может торчать припой. Когда вы работаете на материнской плате, то лучше всего разместить ее на верхней части мягкой, выдающейся поверхности, которая желательно , чтобы коробка материнской платы и ее содержимое располагались в разных местах.

Накопитель на жестком диске

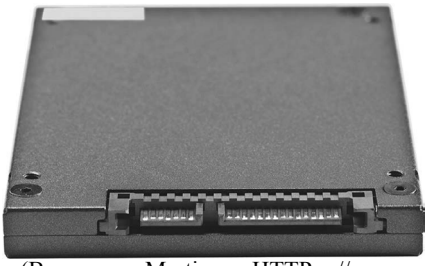
Жесткий диск является стандартной единицей долговременного хранения. В первые дни компьютеров, была использована магнитная лента (катушки к катушке затем кассеты с магнитной лентой и картриджи).

(Предоставлено Hubert Berberich, HTTP: // www.wikimedia.org / вики / Файл: Seagate-ST4702N-03.jpg).

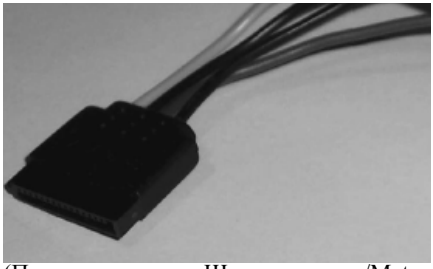
Сегодня жесткий диск хранит большое количество информации (размеры типичны для хранения около 1/2 ТБ до полного ТБ). Наши компьютеры также содержат больший по объему оптический диск, чтобы позволяет нашим компьютерам служить развлекательными консолями (для воспроизведения музыкальных компакт-дисков и фильмов DVD), чем для целей хранения. Жесткий диск содержит несколько пластина, на который информация хранится в виде магнитных зарядов. Головки чтения и записи движутся в унисон над различными поверхностями диска. Жесткий диск также содержит логику и механизмы для вращения дисков, и перемещения головки рычага для чтения / записи.

Существуют два типа дисков, которые мы используем сегодня: IDE (Integrated Drive Electronics) и SATA (Serial Advanced Technology Attachment). Ниже показана задняя часть диска SATA, где вы можете увидеть два соединения, один для питания и один для данных. Справа от привода расположен силовой кабель для привода SATA. Обратите внимание на то, что разъем SATA имеет "L-образную форму", это поможет вам разобраться с разъемами. Пример задней части IDE диска показан ниже. Справа от него находится разъем данных IDE. Подключение питания для IDE привода состоит из четырех отверстий, что соответствует четырем зубцам слева (как показано здесь, привод поставлен с ног на голову, поэтому четыре зубца будет справа) от привода.

В дополнение к материнской плате и приводным устройствам, вы также вставите блок питания в системный блок. Блок питания подключается к большому количеству кабелей. Эти кабели подключаются к материнской плате для питания материнской платы, процессора, панель дисплея (на передней панели системного блока), вентилятор, и диски. На этом рисунке показаны кабели в комплекте. Вы должны быть осторожными, так как при подключении различных кабелей питания к их местоположе-



(Взято из Martixer, HTTP: // www. wikimedia.org / вики / Файл: Molex-к-SATA- power_009.jpg).

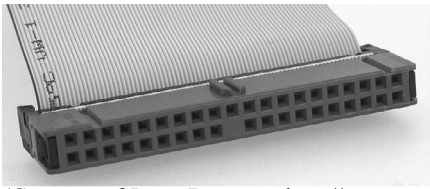


(Предоставлено Шаттерсток /Matee Nuserm.)

(Courtesy of Winhistory, http://commons.wikimedia.org/wiki/File:At-netzteil.jpg.)

нию на материнской плате или дисков, эти кабели могут быть неудобными для правильного подключения.Вы можете спросить, а как же насчет разъемов для клавиатуры, мыши и дисплея? Эти соединения уже являются частью материнской платы. Вы можете удалить панель из задней части системного блока, так что различные разъемы (порты) станут доступны извне. Теперь, когда вы увидели различные компоненты, давайте посмотрим на то, как они сочетаются вместе. Первое, что вы должны сделать, это определить все компоненты. Они будут включаться в материнскую плату, процессор, памяти (чипы), оптический винт, заземляющий браслет и инструменты. Хотя вы не обязаны носить заземляющий ремень, все же, следует помнить, что его необходимо установить до того, как вы начнете работать с электроникой (материнской платой и другим железом). Для дальнейшей защиты аппаратных средств не стоит работать на ковровом покрытии и одевать одежду, которая будет сохранять

(Adapted from Zzubnik, http://commons.wikimedia.org/wiki/File:Hard-drive.jpg.)

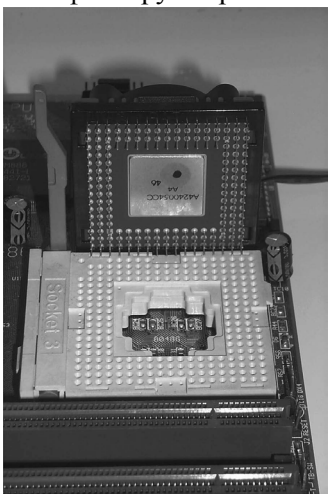


(Courtesy of Jonas Bergsten, http://commons.wikimedia.org/wiki/File:Ata_20070127_001.jpg.)

статический заряд. Статический заряд может привести к повреждению любого из чипов, особенно в центральном процессоре. Кроме того, необходимо сохранять рабочее место в чистоте, без беспорядка и мусора.
После выявления и организации ваших компонентов, первым шагом является установка источника питания в отсек питания системного блока. Вам нужно будет снять боковую панель от системного блока. Поместите источник питания в надлежащем месте и закрепите его винтами. Вам также потребуется снять заднюю панель портов.
Далее, вы вставите DVD и жесткий диск. Вы должны будете удалить переднюю "панель" от системного блока в месте, где вы хотите вставить DVD-диск (вам не нужно делать это для жесткого диска, так как поверхность не будет не видно, но DVD диск должен быть открыт). После того, как ваши диски будут установлены на месте, пока не подключайте к ним никакие кабели. Кабели будут мешать другой работе, так что лучше не сохранять соединения до последних шагов.

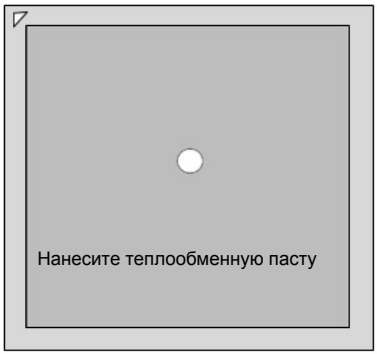
Вставьте жесткий диск в пустой отсек для дисков
в системе INIT скольжения приводных рельсов
Теперь вы можете начать собирать компоненты на материнской плате. Для этого вам нужно установить заземляющий браслет. Вы должны приложить конец заземляющего ремня к чему-то металлическому, предпочтительно к металлу "клетки" из системного блока. Используя материнскую плату, определите в системном блоке тупиковые места. Не прикрепляйте пока что материнскую плату, но установите винты в стойках в их правильном положении (опять же, с помощью материнской платы, чтобы определить, где-вам нужно будет искать резьбовые отверстия на материнской плате и совпадают они с отверстиями для винтов).

Первым компонентом для вставки на материнскую плату является процессор. Убедитесь, что поверхность процессора выровнена (обратите внимание на стрелку), поднимите рычаг блокировки, расположенный на откидной торцевой крышке процессора, пропустите процессор в разъем, опустите крышку, а затем зафиксируйте рычаг на месте.

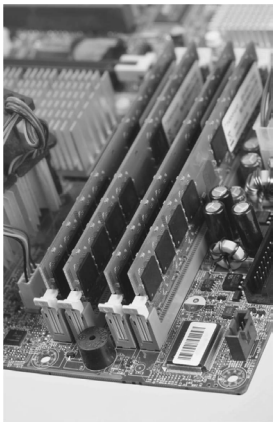


(Courtesy of Vdblquote, http://commons .wikimedia.org/wiki/File:Intel_80486DX4_ Pins_and_socket_3.jpeg.)

Далее установите блок охлаждения центрального процессора. Во-первых, вам необходимо нанести небольшое количество теплопередающей пасты на внешнюю поверхность процессора. Количество пасты, которую вы поместите на процессор мала, размером с небольшую горошину. Далее, сориентируйте блок охлаждения, так чтобы его четыре разъема (прищепки) были расположены соответственно четырем отверстиям вокруг процессорного разъема. Убедитесь, что устройство ориентировано так, что шнур питания может легко поместиться в гнездо питания на материнской плате. После того, как он правильно установлен, нажмите вниз на каждом из четырех штырей по одному за раз. После того, как все четыре штыря были вытеснены через материнскую плату, поднимите материнскую плату для подтверждения. Затем подключите каждый из штырей на поверхности, как показано на рисунке, чтобы зафиксировать штифты на месте



(Courtesy of Shutterstock/Bretislav Horak.)



Затем возьмите плату памяти и вставьте е в один из слотов памяти. Убедитесь, что плата выровнена должным образом, найдите щели в штифтах и сравните их с зазором в гнезде. Убедитесь, что выступы вдоль стороны пазов являются "открытыми". Вставьте плату памяти на отверстие, а затем, используя только большие пальцы, нажимайте вниз, пока не увидите, как вкладки закрываются автоматически. Повторите этот шаг для каждой схемы памяти, которую вы хотите добавить к вашему компьютеру. (Courtesy of Shutterstock/Jultud.)

С помощью этих устройств в месте, вы можете теперь установить винт материнской платы на место. Чтобы завершить установку, вам теперь нужно сделать все между цветовыми обозначениями блока питания, дис-

ков, вентилятор и материнскую плату. Во-первых, найти кабель, который имеет четыре разъема 2-контактный (примечание: в зависимости от блока питания, это может быть один разъем восемь-контактный). Вложения будут показаны на документе установки материнской платы. Например, рисунок ниже показывает соединения для питания дисковода LED, кнопка сброса (перезагрузка), индикатор питания и включения / выключения. Сбой подключения разъемов к соответствующим штифтами приведет эти компоненты не функционируют. На диаграмме ниже, контакты 1/3 используются для жесткого диска LED. Pins 2/4 мощности вкл / выкл LED. Pins 5/7 используются для кнопки перезагрузки. Pins 6/8 используются для света Кнопка включения / выключения в. И, наконец, контакт 9 ничего не делает.

Затем соедините ваши диски. Вы, вероятно, два диска для подключения, оптический привод и жесткий диск. Скорее всего, жесткий диск является IDE, как IDE диски являются более распространенными в компьютерах PC стиле. Оптический привод может быть или SATA или IDE. Определить тип, чтобы определить, который подключается к использованию. Во-первых, вставить диски в доступные бухты в системном блоке. Вы, вероятно, хотите вставить оптический привод на вершине. Это происходит потому, что оптический привод будет взаимодействовать с внешним миром, так что чаще встречается, чтобы он появится в верхней части системного блока. Вам придется снять крышку над этой бухты так, что оптический диск доступен снаружи. Различные системные блоки будут иметь разные типы бухт. Обычно, хотя, все, что вам нужно сделать, это вставьте дисковод в отсек, а затем поверните ключ на стороне залива.
После того, как оба (или все) диски в своих бухтах, необходимо подключить их к материнской плате и блоку питания. Найти подходящий кабель питания для присоединения к работе источника питания и соот-

ветствующий кабель для передачи данных для подключения к материнской плате. Последнее соединение — это подключить кабель питания вентилятора в кабель питания и подключить к материнской плате. На данный момент, вы можете подключить блок питания к компьютеру и устройству защиты от перенапряжений. Убедитесь, что сетевой фильтр подключен к электрической розетке и включен. Теперь вы можете загрузить ваш компьютер!

Если компьютер успешно загружается, он будет загружаться в BIOS ROM, но потому что нет никакой операционной системы на жестком диске, она не достигнет окон. Если вы успешно собрали ваш компьютер, вставьте установочный компакт-диск Windows в оптический дисковод перезагрузите компьютер. После загрузки компьютера в BIOS ROM, вы можете установить Windows. Поздравляем!

ГЛАВА 3.
Двоичная система исчисления

В главе 2 мы увидели, что цифровой компьютер работает путем перемещения электрического тока через его составные части. Тогда мы должны найти способ преобразовать информацию в двоичном представлении. Тогда мы сможем понять, как можно вычислить с помощью двоичного кода. В этой главе мы исследуем двоичную систему исчисления. В главе рассматриваются методы преобразования из двоичной системы исчисления в десятичную и из десятичной в двоичную для беззнаковых целых (положительных) чисел, отрицательных целых чисел, фракционных и чисел с плавающей точкой. В этой главе также вводятся восьмеричные и шестнадцатеричные системы исчисления. Обсуждаются представления символов. Затем в этой главе переходим к бинарным (логическим) операциям и, как проводить с ними вычисления. И, наконец, глава вводит три примера использования двоичных кодов: сетевую адресацию, файлы изображений, а также контрольное число для обнаружения и исправления ошибок.

Целями обучения данной главы являются:

1. Предоставить методы числового преобразования между основаниями: десятичной, двоичной, восьмеричной и шестнадцатеричной, для положительных и отрицательных чисел и чисел с плавающей точкой.
2. Описать, как символы хранятся в памяти компьютера.
3. Продемонстрировать применение двоичных (логических) операций AND, OR, NOT и XOR с двоичными числами.
4. Проиллюстрировать использование двоичного кода на компьютере с акцентом на IP-адресах, хранении файлов изображений и контрольных разрядах чётности.

Компьютер обрабатывает и хранит информацию в виде двоичного кода. Таким образом, мы должны понять, что такое двоичный код, как представлять информацию в двоичной системе, и как применять двоичных (логических) операторов. В этой главе рассматриваются

двоичная система исчисления и несколько связанных с ней систем исчисления. Несмотря на то, что для ученых очень важно понимать двоичный код, в области информационных технологий (ИТ) вы обнаружите, что двоичная система появляется лишь иногда, но все же стоит ее понимать.

Система исчисления (также известная как цифровая система) является письменной системой для выражения чисел. В математике есть бесконечное число систем исчисления, хотя мы, как правило, в значительной степени сосредоточены только на двух, десятичной и двоичной системе. В таблице 3.1 представлены наиболее часто используемые и исторически используемые системы исчисления.

Формально, мы определим систему исчисления в качестве основы k (k – это положительное целое число), где

1. Любое число в этой основе будет содержать цифры от 0 до k – 1.
2. Интерпретация цифры - его значение * степень.
3. Степень основывается на столбце , или позиции, где степень = основа^{столбец}, и где столбец отсчитывается справа налево и крайний правый столбец - столбец 0.

Например, у десятичной системы основой является 10. Это означает, что мы можем использовать цифры от 0 до 9. Степень будет равна $10^{столбец}$. Если наше число 130, то оно представляет значение 1 * 3 + 102 * 101 + 0 * 100, или 1 в столбце сотен, 3 в столбце десятков, и 0 в столбце единиц. Единичное основание, или основание 1, является неподходящим основанием по двум причинам. Во-первых, следуя определению, данному выше, основание 1 означает, что в него входят только цифры 0, но это означало бы, что все наши возможные числа, скажем, 0000, 0000000000, 0, все будет 0, так как значение в формуле есть значение * степень, что всегда равно 0. По этой причине, в единичном коде, мы будем использовать цифру 1 вместо 0. Во-вторых, обратите внимание, что все основания будут 1 ($1^0 = 1$, $1^1 = 1$, $1^2 = 1$, $1^3 = 1$). Результатом является то, что представленное число является лишь суммой количества цифр. Так, например, в единичном коде, 111 является представлением для значения 3, и 111111111 является представление значения 9.

Рассмотрим основание 5. Для этого основания, мы используем только цифры от 0 до 4, и "столбцы" любого числа представляют справа налево столбец единиц, столбец пяти, столбец двадцати пяти (5^2), столбец сто двадцати пяти (5^3), столбец шестьсот двадцати пяти (5^4), и так далее.

Число 1234 с основанием 5 является значением 1 * 125 + 2 * 25 + 3 * 5 + 4 * 1 =194 в десятичной системе счисления.

Таблица 3.1 Распространенные системы исчисления

База	Наименование	Причина использования / Популярность
1	Единичная	Применяется в теоретической информатике при работе с машинами Тьюринга, используется не очень часто
2	Двоичная	Цифровые компьютеры используют двоичную систему и поэтому она изучается в дисциплинах, связанных с компьютерами
8	Восьмеричная	Группы из трех двоичных цифр создают восьмеричную цифру, используемую в дисциплинах, связанных с компьютерами, чтобы сделать чтение двоичного кода более удобным
10	Десятеричная	Первичная система исчисления для человека
16	Шестнадцатеричная	Группы из четырех двоичных цифр создают шестнадцатеричное число, используемое в дисциплинах, связанных с компьютерами, чтобы сделать двоичный код более удобным для чтения, его легче читать, чем восьмеричную (т.к. требует меньшего количества цифр), но является более сложным из-за включения цифра A-F
20	Двадцатеричная	Использовалась в доколумбовой цивилизации Майя; точки и линии представляли цифры, 0-4 обозначались от 0 до 4 точек, 5-9 обозначались от 0 до 4 точек с подчеркиванием, 10-14 обозначались от 0 до 4 точек с двумя подчеркиваниями, 15-19 обозначались от 0 до 4 точек с тремя подчеркиваниями; не использовалось за пределами цивилизации Майя.
60	Клинопись	Использовалась в древней Вавилонской цивилизации

Принимая во внимание любое число $a_n a_{n-1} \ldots a_1 a_0$ с некоторым основанием b, то число интерпретируется с помощью следующей суммы $\sum_{i=0}^{n} (a_j * b^i)$. То есть, мы возьмем цифру и крайнюю правую умножим по b^0 и добавим ее к следующей правой цифре и умножим ее на b1 и добавим к следующей правой цифре и умножим на b2, ..., и добавим это все ко второй крайней левой цифре (в столбце n – 1), а затем умножим это на b^{n-1} и добавим к крайней левой цифре (в столбце n) и умножим его на b^n.

Обратите внимание на то, что в крайнем правом столбце всегда будет 1 независимо от того, какое основание. Почему? Крайний правый столбец всегда будет иметь степень основание^{столбец}, где столбец является 0, или основание^0. В математике, любое целое положительное число (1 или более) повышенное до 0-й степени всегда равно 1. Так как существует бесконечное количество систем исчисления (любое положительное k является правильным основанием), мы должны быть в состоянии обозначить основание целого ряда. Мы делаем это, помещая основание в качестве индекса в конце номера. Так 1234_5 - это 1234 с основанием 5, которое, как мы видели выше, такое же, как 194_{10}. Для удобства, мы позволили себе опустить базу, если мы имеем дело с основанием 10 (в десятичной системе), потому что это основание используется по умолчанию, так что мы бы сказали $1234_5 = 194$.

Рассмотрим основание больше 10, скажем, основание 12. Мы могли бы получить число 246_{12}. Это число на самом деле 2 * 12^2 + 4 * 12 + 6 * 1 = 342. Но напомним, что при любом основании k, нам разрешено использовать цифры 0..k - 1. Для основания 12, это были бы какие-либо цифры 0-11. Теперь у нас есть проблемы, потому что 10 и 11 не являются однозначными числами. Если бы я должен был написать номер 11_{12}, это выглядело бы так, будто я имел в виду 1 в двенадцатом столбце и 1 в столбце единиц вместо 11 в столбце единиц. Это приведет нас к путанице, правильно ли, что $11_{12} = 13$ (из двух цифр) или $11_{12} = 11$ (однозначное число)? Таким образом, мы должны найти альтернативный способ представления цифр с основой больше, чем 10. К счастью для нас, мы, как правило, используем только одно основание больше, чем 10, это основание 16, также известное как *шестнадцатеричное*. Для этого основания, было решено, что символы, используемые для представления 10, 11, 12, 13, 14, и 15 будут A, B, C, D, E и F, соответственно. Таким

образом, значение $3AB2_{16}$ будет означать 3 в столбце 16^3, A (или 10) в столбце 16^2, B (или 11) в столбце 16^1 и 2 в столбце единиц. Итак, что же представляют собой $3AB2_{16}$? $3 * 16^3 + 10 * 16^2 + 11 * 16^1 + 2 * 1 = 15026$.

ДВОИЧНАЯ СИСТЕМА ИСЧИСЛЕНИЯ

Почему все это так важно? Это не было бы таким, если бы наши компьютеры использовали основание 10 (десятичное), но компьютеры хранят всю информацию, передают ее и обрабатывают в двоичной системе вместо десятичной. И хотя, как ИТ-специалисту, вам никогда не придется беспокоиться о том, чтобы смотреть или выражать данные в двоичном коде, использовать недесятичные системы исчисления приходится время от времени. Наиболее распространенное проявление недесятичных систем — это сетевые коммуникации. IP-адреса иногда обозначают в двоичной системе, и IPv6-адреса обозначены в шестнадцатеричной системе. Восьмеричной также пользуются для различных компьютерных приложений, но это происходит гораздо реже. В этом разделе мы рассмотрим сначала двоичные системы, а затем восьмеричные и шестнадцатеричные системы.Двоичная система, вероятно, самая важная система исчисления за пределами десятичной. Из-за цифрового характера компьютеров (то есть, все хранится в одном из двух способов - есть ток или нет тока), все должно быть представлено в виде включено и выключено. Поэтому мы используем двоичную систему, где 1 означает включение, а 0 означает выключение. Это означает, что все в компьютерных магазинах должно сначала быть преобразовано в единицы и нули. Мы будем называть одно значение 1 или 0 для бита (для двоичной цифры). Перемещение данных требует один провод для каждого бита, так что передача данных (скажем 8 бит) может быть сделана в одно время, 1 бит на провод, а не 1 бит за один раз. Цифровые схемы в ЦП работают над битами как регистров, так и хранения памяти битов. Как компьютерный ученый или инженер компьютера, вы иногда должны смотреть на данные или программный код в двоичном виде. Если бы вы должны были смотреть на тысячу битов, на них было бы трудно сосредоточиться. Поэтому вместо этого, мы разбиваем наши данные на группы. Восемь битов помещены вместе в блок, который называется байтом. В более ранних компьютерах было распространено, что данная величина будет храниться в одном байте, а, возможно, и двух. Байт, будучи 8 бит, может хранить любую комбинацию из 8 нулей и единиц, например, 11111111, 10101010, 10001000, 00000000, 01111110, 01011011. Есть 256 различных комбина-

циях нулей и единиц, которые могут быть размещены в 8 битов (1 байт). Мы получаем это число простым вычислением - 28 (умножить два восемь раз).

Числовое преобразование из двоичной в десятичную систему может быть сделано с помощью формулы с предыдущей страницы. Тем не менее, есть и более простой способ сделать это. Возьмем в качестве примера байт хранения 010101012. В соответствии с формулой, мы бы преобразовали его как * 27 0 + 1 * 26 + 0 * 25 + 1 * 24 + 0 * 23 + 1 * 22 + 0 * 21 + 1 * 0 = 20 * 128 + 1 * 64 + 0 * 32 + 1 * 16 + 0 * 8 +* 4 1 + 0 * 2 + 1 * 1 = 85. Обратите внимание, что каждая цифра в нашем двоичном числе равна либо 0, либо 1, в отличие, примера основания 5 в предыдущем разделе, что позволило использовать цифры 0, 1, 2, 3 и 4. При преобразовании из двоичного кода в десятичный, вам не нужно беспокоиться о умножения цифр степени разряда. Вместо этого, цифра будет 0, в этом случае 0 * степень = 0 и может быть удалена из суммирования, или цифра 1, в этом случае 1 * степень = степень. Поэтому наше суммирование ведется лишь в этих 2 из столбцов, которые содержат 1. Отправляясь к предыдущему примеру, мы действительно могли бы записать это как 010101012 64 + 0 + 0 + 16 + 0 + 4 + 0 + 1 = 85, как показано на рисунке 3.1. Если мы сможем узнать полномочия каждого столбца (степени 2), то все, что нам нужно сделать, это сложить эти значения столбцов чьи цифры 1. Таблица 3.2 показывает, что многие из степеней основания 2. Вы получаете это, начиная с 1, а затем удваивая значение для каждой более высокой степени (или каждый столбец, как вы двигаетесь влево). Обратите внимание на то, что полномочия 2, возведенные в 10 единиц (10, 20, 30), сокращены для удобства. 2^{10} = 1024 или 1 К или приблизительно 1000 (если мы не против немного округлить). 2^{20} = 1 М или примерно 1 миллион (как по-

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	0	1	0	1	0	1	
	=		=		=		=	
	64		16		4		1	= 85

Рисунок 3.1
Пример перевода из двоичной системы в десятеричную

Таблица 3.2 Значение двойки в различных степенях

Степень 2	Значение
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1,024
2^{11}	2,048
2^{12}	4,096
2^{13}	8,192
2^{14}	16,384
2^{15}	32,768
2^{16}	65,536
2^{20}	1,048,576
2^{30}	1,073,741,824
2^{40}	1,099,511,627,776

казано в таблице 3.2, точное значение - 1048576). 2^{30} = 1 G или приблизительно 1 млрд. 2^{40} = 1 T или приблизительно 1 трлн. 2^{50} (не показано в таблице) = 1 P для пета-, что составляет примерно 1 квадриллион.

Давайте попробуем сопоставить полномочия 2 в таблице 3.2 для использования. С учетом любого двоичного числа, нужно просто сложить степень 2 из столбцов в таблице 3.2 на основании расположения 1, где самая правая цифра означает значение 2^0. Например, 1101 будет содержать 2^3, 2^2, но там не будет 2^1, и 2^0, или $2^2 + 2^3 + 2^0 = 8 + 4 + 1 = 13$. Мы будем придерживаться чисел длиной в байт (8 бит) для простоты.

11000110: 27 + 26 + 22 + 21 = 128 + 64 + 4 + 2 = 198
10101010: 27 + 25 + 23 + 21 = 128 + 32 + 8 + 2 = 170
01000111: 26 + 22 + 21 + 20 = 64 + 4 + 2 + 1 = 71
00001110: 23 + 22 + 21 = 8 + 4 + 2 = 14
00110011: 25 + 24 + 21 + 20 = 32 + 16 + 2 + 1 = 51
00000000: 0
11111111: 27 + 26 + 25 + 24 + 23 + 22 + 21 + 20 =
128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

Заметьте, что в этом последнем случае, мы имеем большое 8-разрядное число, которое равно 28 - 1.
А что насчет преобразований из десятичной системы в двоичную? Есть две стратегии, которые мы можем применить. Ниже традиционная стратегия, но и другая возможна, стратегия проще будет показана ниже. Возьмите десятичное значение и разделите его на 2. Возьмите коэффициент и разделите его на 2. Продолжайте делать это до тех пор, пока остается 0. Запишите остатки от деления (остаток при делении на 2 будет либо 1 или 0). Соберите остатки в обратном порядке (или снизу-вверх), это является двоичным значением. Например, давайте попробуем 89:

89/2 = 44 с остатком 1
44/2 = 22 с остатком 0
22/2 = 11 с остатком 0
11/2 = 5 с остатком 1
5/2 = 2 с остатком 1
2/2 = 1 с остатком 0
1/2 = 0 с остатком 1

Мы закончили (мы достигли 0). Двоичный эквивалент 89 01011001 (остатки в обратном порядке, с дополнительным битом впереди, чтобы сделать его 8-битным числом). Мы можем подтвердить это путем конвертации этого числа обратно в десятичную систему: 01011001 = 64 + 16 + 8 + 1 = 89.
Давайте попробуем еще один пример, 251:

251/2 = 125 с остатком 1
125/2 = 62 с остатком 1
62/2 = 31 с остатком 0
31/2 = 15 с остатком 1
15/2 = 7 с остатком 1
7/2 = 3 с остатком 1
3/2 = 1 с остатком 1
1/2 = 0 с остатком 1

Таким образом, 251 – это 11111011. Мы превращаем его назад, чтобы проверить, 128 = 11111011 + 64 + 32 + 16 + 8 + 2 + 1 = 251. Подход к делению прост, хотя и не обязательно интуитивен, потому что вы должны

изменить порядок остатков для образования двоичного числа. Другим способом для преобразования из десятичной системы в двоичную является использование вычитаний. Учитывая десятичное число, необходимо определить наибольшую степень 2, которая может быть вычтена из него, оставляя положительное число или 0. Сделайте это, пока ваш десятичное число не равно 0. Для каждой степени 2, которая вычитается из числа, напишите 1 в этом соответствующем столбце.
Рассмотрим в качестве примера то же значение, которое мы только что преобразовали, 251. Возвращаясь к таблице 3.2, наибольшая степень 2, что значит, мы можем вычесть из 251 128. Вычитая 128 из 251, мы имеем 123. Самая большая степень 2, которую мы можем вычесть из 123 составляет 64, в результате чего получается 59. Наибольшая степень 2, которую мы можем вычесть из 59 32, оставляя 27. Наибольшую степень 2, которую мы можем вычесть из 27 16 - 11. Наибольшая степень 2, которую мы можем вычесть из 11 8, оставляя 3. Самая большая степень 2, которую мы можем вычитать из 3 является 2, оставляя 1. Самая большая степень 2, которую мы можем вычесть из 1 равно 1, оставляя 0. Мы закончили. Наш процесс преобразования произошел вот так:

251 – 128 = 123
123 – 64 = 59
59 – 32 = 27
27 – 16 = 11
11 – 8 = 3
3 – 2 = 1
1 – 1 = 0

Таким образом, 251 = 128 + 64 + 32 + 16 + 8 + 2 + 1. Таким образом, наш двоичный эквивалент имеет 1 в сталкивающихся столбцах, представляющих 128, 64, 32, 16, 8, 2 и 1 и 0 в другом месте (только другой столбец представляет 4). Это дает нам 11111011. Мы можем проверить нашу работу, как показано на рисунке 3.2, которая преобразует 11111011 обратно в десятичную систему, 251.
Давайте разберем еще один пример. Преобразование 140 в двоичной системе. Самая большая степень 2, которую мы можем вычесть из 140 128 оставляя 12. Самая большая степень 2, которую мы можем вычитать из 12 8 оставляя 4. Самая большая степень 2, которую мы можем вычитать из 4 4 оставляя 0.

140 - 128 = 12
12 - 8 = 4
4 - 4 = 0

140 = 128 + 8 + 4. Таким образом, мы имеем 1 в столбцах, которые представляют 128, 8 и 4, 0 в другом месте. Это значение 10001100, так что 140 = 10001100.

До сих пор, мы ограничивали наши двоичные числа до 8 бит. Напомним, что с 8 битами мы имеем 256 различных комбинаций 1 и 0. Поскольку 00000000 равен 0, то 8 бит (1 байт) диапазон значений от 0 до 255. Что делать, если мы хотим хранить большее количество? Тогда нам нужно больше битов. Например, в 2 байта (16 бит), мы можем хранить любое число от 0 до 65535. Напомним, что 216 = 65536. В общем случае, если мы имеем n битов, мы можем хранить любое число от 0 до 2^n - 1. Хотя мы часто используем байт как удобный объем памяти (вместо 1 или несколько битов), ограничивая себя 1 байтом, мы загоняем себя в рамки. Современные компьютеры обычно обеспечивают 32 бита (4 байта) или 64 бита (8 байт) для типичного размера для хранения. Мы имеем в виду типичный размер памяти как размер слова компьютера.
Насколько велики 32 бита? Это дает нам такой диапазон значений:

00000000000000000000000000000000 = 0
...
11111111111111111111111111111111 = 4294967296 (около 4,3 миллиарда)

Для большинства приложений, 32 бит дает нам достаточно места для хранения. В 64 бит, наибольшее число, которое мы можем хранить - более 18 квадриллионов (за 18 следуют 15 нулей!) Это огромные цифры. Являются ли 32- или 64-битное слово расточительством памяти? Это

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	0	1	1	
=	=	=	=	=		=	=	
128	64	32	16	8		2	1	= 251

Рисунок 3.2
Преобразуем назад, чтобы проверить ответ

зависит от приложения. Тем не менее, с объемом памяти не менее 4 миллиардов байт и дискового хранения не менее половины триллиона байт, используя 32 или 64 бита для хранения номера не должно быть причиной для беспокойства.
Подумайте о том, чтобы смотреть на ряд чисел, хранящихся в 32-битных двоичных значениях. Поскольку все цифры равны 1 и 0, а потому есть много из них, в результате чего на список становится очень трудно смотреть, почти как смотреть на оптическую иллюзию. Для удобства, мы можем перевести двоичные числа в различных представлениях, обычно в восьмеричном (основание 8) или шестнадцатеричном формате (основание 16). Давайте попробуем следующий список. Представьте себе страницы и страницы этого материала!

11101100100010110010101100000111
00101100010100101010110110010001
10001011001001001011011100100111
10001000101001111010001110101011
00001001010100000111100010101100

Можно заметить, что 8 и 16 кратны 2. Это делает перевод с основанием от 2 до 8, или основание 16, простым (по сравнению с переводом основания от 2 до 10 или основанием 10 к основанию 2). Для основания 8, необходимо взять 3 двоичных разряда и сгруппировать их вместе, а затем преобразовать эти 3 бита из двоичных в восьмеричные. Это работает, потому что 23 = 8. Заметим, что так как восьмеричное использует только восемь цифр (от 0 до 7), конвертируя 3 бита, из двоичного в восьмеричное является тем же самым преобразованием, как и 3 бита из двоичной системы счисления в десятичную. В таблице 3.3 показано, как преобразовать 3 бита между двумя основаниями.

Таблица 3.3 Таблица перевода из двоичного кода в восьмеричный

Двоичный	Восьмеричный
000	0
001	1
010	2
011	3
100	4
101	5

110	6
111	7

Используя первое 32-битное число с предыдущей страницы, мы будем конвертировать его в восьмеричное.

11101100100010110010101100000111

Во-первых, разделить 32-разрядное число на 3-битовые сегменты, начиная с правого конца
101 100 11 100 010 110 010 101 100 000 111

Во-вторых, так как это не крайний левый сегмент из 3 битов, мы добавим ведущие 0, чтобы сделать его в 3 бита

011 100 100 101 010 110 010 101 100 000 111

В-третьих, используя приведенную выше таблицу, мы преобразуем каждую группу из 3-х битов в его эквивалент восьмеричного значения

3 5 4 4 2 6 2 5 4 0 7

И, наконец, мы объединяем отдельные восьмеричные цифры в одно значение: 35442625407.
Таким образом, 11101100100010110010101100000111$_2$ = 354426254078.

Остальные четыре цифры преобразуются следующим образом:

00 101 100 010 100 010 110 101 110 010 001 = 054245266218
10 001 011 001 001 011 011 001 100 100 111 = 213111334478
10 001 000 101 001 010 001 111 110 101 011 = 210517216538
00 001 001 010 100 111 100 000 010 101 100 = 011240742548

Теперь весь список чисел:

35442625407
05424526621
21311133447
21051721653

01124074254

Есть еще много цифр для перевода в нашем восьмеричном исчислении чисел. Мы можем уменьшить размер списка еще более трудным путем преобразования двоичного числа в шестнадцатеричной системе. В шестнадцатеричном формате, у нас будет группа 4-разрядных двоичных чисел и их нужно будет преобразовать в эквивалентный шестнадцатеричный десятичный знак. В таблице 3.4 приведены двоичные, шестнадцатеричные и десятичные эквиваленты для отдельных шестнадцатеричных цифр (первые цифры 0-15 в десятичной системе). Обратите внимание, как обсуждалось ранее, что для 10-15, мы используем A-F, так что мы можем хранить эти значения в качестве отдельных цифр.

Таблица 3.4 Преобразование из двоичной и шестнадцатеричную систему

Двоичный	Шестнадцатеричный	Десятичный
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Давайте используем ту же цифру, как и раньше, 111011001000101100101011000001112 и конвертируем в шестнадцатеричную. Мы используем тот же подход, что мы делали при преобразованиях из двоичного в восьмеричное за исключением того, что мы используем группы по 4 бита.

Обратитесь к таблице 3.4, чтобы увидеть шестнадцатеричные цифры.

1110 1100 1000 1011 0010 1011 0000 0111
E C 8 B 2 B 0 7

Как было показано выше, 11101100100010110010101100000111$_2$ = EC8B2B07$_{16}$.
Другие цифры в нашем списке будут преобразованы следующим образом:

0010 1100 0101 0010 1010 1101 1001 0001 = 2C52AD91$_{16}$
1000 1011 0010 0100 1011 0111 0010 0111 = 8B24B727$_{16}$
1000 1000 1010 0111 1010 0011 1010 1011 = 88A7A3AB$_{16}$
0000 1001 0101 0000 0111 1000 1010 1100 = 095078AC$_{16}$

Чтобы преобразовать восьмеричное значение обратно в двоичное, нужно просто заменить каждую восьмеричную цифру на его трехзначный двоичный эквивалент. Убедитесь, что вы используете все 3 цифры в двоичной системе. Например, 7238 = 111 010 011$_2$, а не 111 10 11$_2$. Значение 123456708 = 001 010 011 100 101 110 111 000 = 00101001110010111011100002, и если нужно было сохранить в 32 битах, мы добавим начальные нули и продлим их до 32 бит, что дает нам:

000000000010100111001011101110002

Для того, чтобы преобразовать значение из шестнадцатеричного в двоичное, просто нужно заменить каждую шестнадцатеричную цифру с 4 эквивалентным разрядом двоичного значения. Например, 2468BDF016 = 0010 0100 0110 1000 1011 1101 1111 00002 = 00100100011010001011101111100002. Преимущество использования восьмеричной системы или шестнадцатеричной мало перед двоичной и является тем, что эти цифры легче читать.

ОТРИЦАТЕЛЬНЫЕ ЧИСЛА И ДРОБИ

В двоичной системе счисления, мы видели, как любое целое положительное число (целое число) может быть преобразовано из десятичной системы счисления в двоичную. Все, что нам нужно, это достаточное количество битов для хранения двоичного числа. Как мы хранить отрицательные числа и числа с десятичной точкой? Нам нужно

использовать различные представления.
Для отрицательных чисел, компьютеры используют представление, называемое двоичное дополнение. Дополнительный код числа идентичен обычному двоичному числу, если значение положительное. Если число отрицательное, мы должны выполнить преобразование. Преобразование требует, чтобы мы приняли положительное двоичное число, переворачивая все биты (0 становятся 1, 1 становятся 0), и добавьте 1. Добавление рассматривается в двоичных операциях, но мы будем применять его к некоторым простым из всех примеров здесь.
Представьте себе, что у нас есть число 61, и мы хотим сохранить его как 8-битное до двух значений среды. Так как оно положительно, мы используем нормальное десятичное в двоичном преобразовании: 61 = 32 + 16 + 8 + 4 + 1 = 00111101, так что 61 это 00111101.
Как насчет -61? Так как 61 = 00111101, чтобы получить -61, переворачивайте все биты и добавьте 1. Перевернутые биты 00111101 11000010 дают нам (просто переделать каждый 0 в 1 и каждый 1 в 0). Теперь добавьте 1 к 11000010, и вы получите 11000011. Так что -61 = 11000011. Обратите внимание, что ведущий бит равен 1. В дополнение, отрицательное число всегда будет начинаться с положительного числа, 1 всегда будет начинаться с 0.
Давайте попробуем еще раз, 15. Это будет 8 + 4 + 2 + 1 = 00001111. Так что -15 требует, чтобы мы перевернули все биты, чтобы получить 11110000, а затем добавили 1, чтобы получить 11110001. Опять же, обратите внимание, что положительный вариант начинается с 0, отрицательный вариант начинается с 1.
Для преобразования из двоичной системы счисления в десятичную, если число является положительным, нужно просто преобразовать его, как обычно. Например, 00111101 положительно, так что число будет 32 + 16 + 8 + 4 + 1 = 61, и 00001111 положительно, так что число будет 8 + 4 + 2 + 1 = 15.
Как насчет 11000011? Это отрицательное число. Мы используем ту же самую технику преобразования, которую мы использовали для преобразования из десятичной системы в двоичную: перевернуть все биты и добавить 1. Так 11000011 становится 00111100 + 1 = 00111101, который является десятичным числом 61, поэтому 11000011 должен быть -61. И 11110001 отрицательна, поэтому мы перевернем все биты и добавить 1, что дает нам 00001110 + 1 = 00001111, который является десятичным числом 15, поэтому 11110001 должно быть -15.
Теперь давайте попробуем преобразовать -48 в двоичный формат. Мы

снова начинаем с +48 = 32 + 16 = 00110000. Поскольку мы хотим -48, мы должны перевернуть все биты и добавить 1. После того, как мы перевернули все биты, мы переходим от 00110000 к 11001111. Теперь мы должны добавить 1. Что такое 11001111 + 1? Ну, нам нужно немного двоичной арифметики для решения. Это не так уж сложно; вы просто должны выполнить несколько переносов, что даст вам 11010000. Так -48 является 11010000.
Так как двоичная арифметика рассматривается в двоичных операциях, на данный момент, мы рассмотрим альтернативный подход к преобразованию отрицательных чисел в двоичной системе. Мы снова начинаем с отрицательного числа в десятичной системе. Нам нужно преобразовать положительную версию числа в двоичном виде, как мы это делали в предыдущем подходе. Мы должны преобразовать положительное двоичное число в его отрицательном эквиваленте в виде дополнения до двух. Теперь вы можете применить эту простую стратегию:

1.Начните с правой цифры, работая справа налево, копируйте каждый 0, пока вы не достигнете первого 1 бита
2. Скопируйте первый 1 бит
3. Продолжайте работу справа налево, переворачивая все остальные биты

Например, 00110000 будут преобразованы следующим образом:

1.Начиная с правой стороны, мы копируем вниз все 0

... 0000

2.Когда мы достигнем первой 1, скопируйте и его

... 1 0000

3.Переверните все остальные биты, в данном случае 001 становятся 110

110 1 0000
Таким образом, преобразования 00110000 от положительного до отрицательного дает нам 11010000. Что если число имеет 1 в крайнем правом положении? Начнем с номером 11000011 (-61). Шаг 1 не

используется, потому что нет 0; поэтому, вместо этого, мы записываем крайний правый 1, а затем переворачиваем остальные биты, давая нам 00111101 (+61).

Дробные числа проще и более сложны для понимания. Их легче понять, потому что они являются продолжением по формуле, представленной ранее, $\sum_{i=0}^{n} (a_j * b^i)$за исключением того, что мы распространяем, *i* включите отрицательные числа для тех, которые появляются на правой стороне от десятичной точки. Вы можете вспомнить из основной алгебры, $b^{-i} = 1/b^i$. Таким образом, b^{-3} действительно $1/b^3$. В двоичном виде, *b* всегда 2, поэтому мы должны изучить некоторые отрицательные степени 2. Таблица 3.5 показывает несколько отрицательных степеней 2.

Таблица 5.3 Отрицательные (дробные) Полномочия 2

Экспоненциальная форма	Значение в качестве дроби	Значение в качестве действительного числа
$2^{-1} = 1/2^1$	1/2	0,5
$2^{-2} = 1/2^2$	1/4	0,25
$2^{-3} = 1/2^3$	1/8	0,125
$2^{-4} = 1/2^4$	1/16	0,0625
$2^{-5} = 1/2^5$	1/32	0,03125
$2^{-6} = 1/2^6$	1/64	0,015625
$2^{-7} = 1/2^7$	1/128	0,0078125

Давайте посмотрим на примере. Что является двоичным числом 110111,01? Используя нашу формулу сверху, 110111,01 = 1 * 25 + 1 * 24 + 0 * 23 + 1 * 22 + 1 * 21 + 1 * 20 + 0 * 2-1 + 1 * 2-2 =
32 + 16 + 4 + 2 + 1 + ¼ = 55,25.

Еще одним примером может служить 101,101 = 4 + 1 + 1/2 + 1/8 = 5,625. Можете ли вы разработать следующие примеры?

1111.1001

0,1111

10101.101

Преобразование из двоичной системы счисления в десятичную не очень сложно. Тем не менее, существует проблема. Когда мы хотим сохранить значение, такое как 55.25 на компьютере, оно должно быть переведено на двоичный код, таким образом мы получаем 110111.01. Тем не менее, компьютер хранит только 1 и 0, как мы можем указать десятичную точку? Ответ в том, что мы не будем пытаться сохранить десятичную точку, но мы делаем запись, где десятичная точка должна быть.

Для того, чтобы обозначить, где десятичная точка, мы используем более сложную стратегию, основанную на научном примечании. В научной нотации, значение 55,25 фактически представлено в виде $0{,}5525 * 10^2$. То есть, десятичная точка сдвигается и предшествует первой ненулевой цифре. После смещения десятичной точки, однако, мы должны умножить значение на степень 10. Таким образом, 55,25 будет $0{,}5525 * 10^2$. Теперь у нас есть два числа, чтобы хранить, 0.5525 и 2. Мы можем пропустить десятичную точку, так как подразумевается, что запятой всегда будет предшествовать первая ненулевая цифра. Мы также можем опустить 10, так как он предполагает, что основание 10.

В двоичном виде, мы делаем ту же самую основную вещь. Начнем с 110111.01, нужно сдвинуть десятичную точку, чтобы она предшествовала ненулевой цифре (первая 1) путем умножения на степень 2. Затем мы опускаем десятичную точку и 2 и сохраняем номер в виде двух частей. Используя 110111.01 в нашем примере, мы сначала сдвигаем десятичную точку, что дает нам $0{,}11011101 * 2^6$. Мы получаем 2^6, потому что мы перешли от десятичной запятой 6 позиций влево. Так как наш номер стал меньше (пошел от 110111.01 до .11011101), мы должны были умножить на большую величину. В настоящее время, мы просто храним это в виде двух чисел, 11011101 и 110 (при показателе 6). Значение 2 используется для умножения и является базовым. Так как под основанием 2 имеется в виду, что мы можем удалить его из нашего числа. На самом деле мы добавляем третий ряд с нашего склада, знаковый разряд. Мы будем использовать 0 для положительных и 1 для отрицательных. Таким образом, наше число становится равным 0 для знакового бита, 110 для показателя и 11011101 для фактического количества. Мы называем эту часть как мантиссу.

Мы организуем три части этой памяти в качестве знакового бита, бита экспоненты, бита мантиссы. Знаковый бит всегда один бит. Мы должны выбрать размер в битах для экспоненты и мантиссы. Размер мы выбираем так, что он повлияет на точность нашего представления. Давайте создадим 16-бит с плавающей точкой, такое представление выглядит

следующим образом: 1 знаковый бит, 5 бит экспоненты, 10 битов мантиссы. Показатель будет храниться в виде дополнения до двух, так что мы можем иметь как положительные, так и отрицательные показатели. Мантисса имеет положительное значение, так как знаковый бит будет указывать, является ли число положительным или отрицательным.

С 55.25 из нашего первоначального десятичного числа, мы сначала преобразуем его в двоичном виде, как 110111.01. Теперь мы переходим на десятичную точку, чтобы дать нам .11011101 * 2^6. Мы превращаем экспоненту в двоичный код с использованием 5 битов, 00110. И, наконец, мы преобразуем мантиссы на 10 бит, что дает нам 1101110100. Обратите внимание, что, когда мы добавили 0 к мантиссе, мы добавил в конце нули. Это происходит потому, что номер представляет собой дробь, и поэтому добавление нулей в конце не меняет значение. Мы добавили ведущие нули к показателю, так как показатель не дробь. Это осуществляется через добавление 0 к 12.34, мы хотели бы добавить ведущие нули слева от всей части и замыкающие нули справа от дробной части, что давало бы нам, например, 0012.340000. Теперь наше значение с плавающей точкой для 55.25 0 (знаковый бит), 00110 (бит экспоненты), 1101110100 (мантисса). Это хранится в виде 16-битового значения в качестве 0001101101110100. Попробуем преобразовать другое десятичное число в двоичный формат с помощью 16-битного представления. Мы возьмем -14.75. Во-первых, мы будем принимать во внимание знак минус на данный момент. Мы конвертируем 14.75 в бинарный код следующим образом: 14,75 = 8 + 4 + 2 + 0,5 + 0,25 или 8 + 4 + 2 + 1/2 + 1/4. Это двоичное число 1110,11. Теперь, мы должны перенести десятичную точку, чтобы она предшествовала первому 1, что дает нам .111011 * 24 Таким образом, наша мантисса 111011 и наш показатель равен 4 или 100. Продолжим мантиссу до 10 бит, что дает нам 1110110000 и наш показатель до 5 биты, что дает нам 00100. Наконец, мы начали с отрицательным числом, поэтому знак бита равен 1. Таким образом, -14,75 = 1 00100 1110110000 = 1001001110110000.

Мы сделаем еще одно преобразование из десятичной системы счисления в двоичную. Мы будем конвертировать 0,1875 в двоичную. Это выглядит сложно, потому что в этой цифре нет чалой части. 0,1875 состоит из 1/8 + 1/16 или двоичного числа 0,0011. Напомним, что мы должны перенести десятичную точку перед первым 1. В этом случае мы переносим десятичную точку справа, а не слева. Таким образом, 0.0011 становится больше, а не меньше, как мы видели в двух предыдущих примерах. Поэтому мы должны умножить на отрицательную степень 2, а не положительную степень 2. 0,0011 = 0,11 * 2-2. Наш показатель

является отрицательным числом. Мы можем хранить положительные и отрицательные цифры в виде дополнения до двух. В 5 бит, -2 = 11110. Таким образом, наш показатель является 11110. Наш знак бит равен 0. И, наконец, наша мантисса, продлена до 10 бит и составляет 1100000000. Таким образом, 0,1875 = 0 11110 1100000000 или 01111011000000000.

Давайте теперь конвертировать из нашего представления в 16-бит с плавающей точкой обратно в десятичную систему. Мы будем использовать число 1001111100010110. Мы делим это число на три части и имеем 1 00111 1100010110. Начальная 1 говорит нам, что это отрицательное число. Следующая группа цифр является показателем. 00111 двоичного кода для 7, так что наш показатель 7 означает, что мы будем сдвигать десятичную точку на 7 позиций вправо. И, наконец, последняя группа - цифры мантиссы. Десятичная точка подразумевается для предварения этих цифр. Так 1100010110 действительно равно 0,1100010110 * 200111. Когда мы умножаем на 200111, мы получаем 1100010.110. Теперь мы преобразуем это число из двоичной системы счисления в десятичную. Это число 64 + 32 + 2 + ½ + ¼ = 98,75, а так как знак бит был 1, это число -98,75.

Вот еще один пример, 0111011010000000, который является 0 11101 1010000000. Это число является положительным. Показатель является 11101, который является отрицательным числом в виде дополнения до двух. Мы превращаем его в положительной версии со сложением до двух, который является 00011, или 3. Таким образом, показатель действительно -3, что означает, что мы будем сдвигать десятичную точку на три позиции влево (а не вправо). Мантисса равна +0,10100000000, или 0,101 для удобства. Таким образом, наш номер 0,101 * -3 = 0,000101. Используя наши двоичные числа в десятичных алгоритмах преобразования, мы получаем 1 = 0,000101 * 2-4 + 1 * 2-6 = 1/24 + 1/26 = 1/16 + 1/64 = 3/64 = 0,046875.

Наше 16-битное представление на самом деле не используется в компьютерах. Есть два популярных формата, или представления, используемые для современных компьютеров. Их часто называют одинарной точностью и двойной точностью. В одинарной точности, 32 бита используются для знака, экспоненты и мантиссы. В двойной точности, используются 64 бита.

ПРЕДСТАВЛЕНИЕ СИМВОЛОВ

Помимо номеров, мы также хотим сохранить текст на компьютере.

Мы будем хранить текст, разбивая его на отдельные символы. Каждый символ будет храниться в каком-то отдельном объеме памяти (например, 1 байт на символ). Таким образом, текст, например, этой главы, потребует преобразования каждого символа в его представлении символов, а затем ему нужно будет хранить каждое представление символов в байте памяти. Символ представления должен будет включать в себя не только 26 букв алфавита, но и то будут ли буквы в верхнем или нижнем регистре (52 всего букв), 10 цифр (для того, когда мы ссылаемся на цифры, а не цифры, например, чей-то почтовый индекс или номер телефона), знаки препинания, такие как #, $, &, *, +, пустое пространство (пробел), и конец строки (возврат каретки или клавишу ввода).

Есть формы представления символов, ASCII (Американский стандартный код для обмена информацией) и Unicode. Третья форма, EBCDIC, широко использовался в 1960-е и 1970-е годы. Мы кратко обсудим эти три формы представления здесь.

Каждая из этих форм представления является техногенной, то есть, кто-

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Рисунок 3.3

Таблица ASCII. Каждый символ отображается в десятичной, шестнадцатеричной и восьмеричной системе. (Благодарности ZZT3, http://commons.wikimedia.org/wiki/File:Ascii_Table-nocolor.svg.)

то изобрел их. В каждом случае, кто-то решил, что данный символ будет сохранен с использованием конкретных двоичных разрядов. Несмотря на то, что это не все, нам важно знать не только эти битовые шаблоны, два из трех представлений показаны на рисунках 3.3 и 3.4. Рисунок 3.3 демонстрирует представление ASCII.

Рисунок 3.4 демонстрирует представление в EBCDIC. Обратите внимание, что некоторые из символов являются управляющими или непечатными символами и поэтому не отображаются в таблице.
EBCDIC используется только в больших ЭВМ IBM и ее перестали использовать отчасти потому, что это было не очень логическим представлением. Если вы посмотрите через представления ASCII, вы увидите, что для любых двух букв, представление ASCII в том же порядке, как заглавные буквы появляются в алфавите; так, например, "a" непосредственно предшествует 'B', который непосредственно предшествует 'c', и так далее. Представление организовано таким образом, что заглавные буквы появляются перед строчными буквами. Это означает, что 'A' меньше 'a', когда мы их сравниваем. В EBCDIC, хотя буквы по порядку ('a' перед 'B' до 'c'), они не обязательно сразу следуют друг за другом, например, 'j' не следует сразу за "i".
Способ ASCII работает так, что вместо символа, мы имеем двоичную последовательность. Например, 'a' сохраняется в виде десятичного значения 97, или шестнадцатеричное значение 61. Используя либо из десятичной в двоичную или шестнадцатеричной в двоичную, вы увидите, что 'a' – это 01100001. Сравните это с 'A', который является десятичной 65 (шестнадцатеричный 41), которая является 01000001. Интересно, что "a" и "A" почти идентичны, единственное различие состоит в третьем бите слева. Версия верхнего регистра каждой буквы имеет третий бит 0, а нижняя версия в каждом случае буквы имеет третий бит 1.
Представление ASCII использует 7 бит, что дает нам 128 различных комбинаций. С 52 буквами, 10 цифрами, а также различными знаками препинания, там осталось место и для других символов. К ним относятся так называемые управляющие последовательности, такие как клавиша табуляции и клавиши возврата (или "конец строки"). Несмотря на использование 7 битов для ASCII, каждый символ ASCII хранится в 1 байте, поэтому последний бит становится 0. 0 будет вставлен как приводящий бит (крайний слева). Представление EBCDIC использует 8 бит и также содержит 256 различных комбинаций.
Давайте рассмотрим пример. Мы хотим сохранить строку "Hello".

Десятичный код	Символ	Десятичный код	Символ	Десятичный код	Символ	Десятичный код	Символ
000	NUL	043	CU2	127	"	210	K
001	SOH	045	ENQ	129	a	211	L
002	STX	046	ACK	130	b	212	M
003	ETX	047	BEL	131	c	213	N
004	PF	050	SYN	132	d	214	O
005	HT	052	PN	133	e	215	P
006	LC	053	RS	134	f	216	Q
007	DEL	054	UC	135	g	217	R
008	GE	055	EOT	136	h	224	\
009	RLF	059	CUB	137	i	226	S
010	SMM	060	DC4	145	j	227	T
011	VT	061	NAK	146	k	228	U
012	FF	063	SUB	147	l	229	V
013	CR	064	BLANK	148	m	230	W
014	SO	074	¢	149	n	231	X
015	SI	075	.	150	o	232	Y
016	DLE	076	<	151	p	233	Z
017	DC1	077	(	152	q	240	0
018	DC2	078	+	153	r	241	1

Рисунок 3.4
Таблица EBCDIC. (От www.LookupTables.com, http://www.lookuptables.com/ ebcdic_scancodes.php. С разрешения.)

019	TM	079	|	161	~	242	2
020	RES	090	!	162	s	243	3
021	NL	091	$	163	t	244	4
022	BS	092	*	164	u	245	5
023	IL	093	)	165	v	246	6
024	CAN	094	;	166	w	247	7
025	EM	095	¬	167	x	248	8
026	CC	096	-	168	y	249	9
027	CU1	097	/	169	z		
028	IFS	106	¦	192	{		
029	IGS	107	,	193	A		
030	IRS	108	%	194	B		
031	IUS	109	_	195	C		
032	DS	110	>	196	D		
033	SOS	111	?	197	E		
034	FS	121	`	198	F		
036	BYP	122	:	199	G		
037	LF	123	#	200	H		
038	ETB	124	@	201	I		
039	ESC	125	'	208	}		
042	SM	126	=	209	J		

Рисунок 3.4
Таблица EBCDIC. (От www.LookupTables.com, http://www.lookuptables.com/ ebcdic_scancodes.php. С разрешения.)

Как это делается? В ASCII, каждый из символов представлен другим кодом. "H" является 01001000, "e" 01100101 "l" встречается дважды, оба раза 01101101, и «o» 01101111. Таким образом, "Hello" хранится в виде последовательности из пяти 8-битовых последовательностей, или пять байтов: 01001000 01100101 01101101 01101101 01101111. В EBCDIC, она занимает столько памяти (пять байт) для хранения "Hello", но представления различаются. В EBCDIC, мы имеем вместо: 11001000 10000101 10010111 10010111 10010110.
Ни ASCII, ни EBCDIC не имеет какие-либо представления для букв за пределами английского алфавита. С международным характером вычислений сегодня, существует необходимость представлять символы других языков; например, арабские буквы, кириллица или русский, китайские и японские иероглифы. Тем не менее, большинство компьютерных пользователей не говорят по-английски. Поэтому было принято решение о расширении ASCII от 7 бит до 16 бит, что дает нам 65,356 различных комбинаций, или возможность хранить более 65000 символов. Это стало UNICODE. В UNICODE, первые 128 символов одни и те же, как и в ASCII. Тем не менее, остальные используются для хранения символов, найденных в других языках, наряду с другими символами. Эти другие символы включают знаки зодиака, символы для шахматных фигур, знаки мира, символы инь/ян, смайлики, символы валюты, математические символы и стрелки.
Предыдущий пример для "Hello" будет выглядеть в Unicode, так как первые 128 символов Unicode те же 128 символов ASCII. Тем не менее, в UNICODE, каждый символ хранится в 16 бит, а не 7. Так, "Hello" будет выглядеть так:

0000000001001000 "H"
0000000001100101 "e"
0000000001101101 "l"
0000000001101101 "l"
0000000001101111 "o"

Таким образом, "Hello" требует в два раза больше места для хранения в Unicode, как это делает в любом ASCII или EBCDIC. В конце этой главы, мы найдем другое применение ведущего бита в ASCII (крайний левый бит, в который мы вставили 0).

ДВОИЧНЫЕ ОПЕРАЦИИ

Помимо преобразования чисел, компьютер должен обрабатывать значения. Как правило, мы думаем о компьютерах, выполняющих такие операции, как сложение, вычитание, умножение, деление и сравнение (сравнение двух значений, чтобы определить, если они равны или что они меньше, чем другие). Тем не менее, компьютер работает только на двоичные значения и выполняет двоичные операции (это то же самое, что и булевы операции).

AND – оба ли значения на единицах?

OR – есть ли значение для единиц?

NOT - преобразования значения в свою противоположность (1 становится равным 0, 0 становится 1)

XOR - исключение или, являются ли два значения различными?

Таблицы истинности для этих четырех основных логических операций показаны на рисунке 3.5.
Рассмотрим несколько кратких примеров применения двоичных, или булевых, операций.
Во-первых, мы смотрим на простых примерах, 1-битных.

```
0 OR 0 = 0      0 OR 1 = 1      1 OR 1 = 1
0 XOR 0 = 0     0 XOR 1 = 1     1 XOR 1 = 0
0 AND 0 = 0     0 AND 1 = 0     1 AND 1 = 0
```

X	Y	AND	X	Y	OR	X	NOT	X	Y	XOR
0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	1	1	0	0	1	1
1	0	0	1	0	1			1	0	1
1	1	1	1	1	1			1	1	1

Рисунок 3.5
Базовые таблицы истинности логической операции.

В сущности, мы возьмем два бита и посмотрим строку в соответствующей таблице истинности. При 0 XOR 1, мы смотрим на строку, где X = 0 и Y = 1.
Даже если мы используем AND, OR и XOR для всего двух битов (и не только 1), становится более интересно и полезно посмотреть тоже самое на больших двоичных операциях. Например, 11110000 и 10101010. В этом случае мы применяем двоичную операцию побитовым способом. То есть, мы будем применять к первому биту каждого двоичного числа, а затем ко второму биту каждого двоичного числа, а затем ..., записывается в этом случае 8-битовый ответ, потому что каждое число имеет длину 8 бит. Операция может быть легко выполнена, выравнивая два числа на две строки, а затем применяя колонку операции по столбцам.

Вычислим 11110000 И 10101010.

	1	1	1	1	0	0	0	0
AND	1	0	1	0	1	0	1	0
	1	0	1	0	0	0	0	0

Начиная с крайнего левого бита, 1 AND 1 равно 1. В следующей колонке движется вправо, 1 AND 0 равен 0. Далее, 1 AND 1 равно 1. Далее, 1 AND 0 равно 0. Итак, обратите внимание, что остальная часть битов в верхней части, как мы двигаемся вправо равны 0. 0 AND чего угодно равен 0, так что все последние 4 бита будут 0.
Давайте попробуем XOR, что немного сложнее. 10101111 00111100 XOR.

	1	0	1	0	1	1	1	1
XOR	0	0	1	1	1	1	0	0
	1	0	0	1	0	0	1	1

При этом, если два бита одинаковы, то результат будет 0, и, если два бита отличаются, результат равен 1. Более сложная последовательность операций, может быть применена, например, NOT (10110011) AND (11110000 11001100 XOR). Во-первых, мы будем применять XOR:

	1	1	1	1	0	0	0	0
XOR	1	1	0	0	1	1	0	0
	0	0	1	1	1	1	0	0

Далее, мы применяем NOT (10110011). Это просто переворачивает каждый бит дает нам 01001100. Наконец, мы применяем AND.

	0	1	0	0	1	1	0	0
AND	0	0	1	1	1	1	0	0
	0	0	0	0	1	1	0	0

Поэтому NOT (10110011) AND (11110000 XOR 11001100) = 00001100.

Вы можете спросить, почему мы возимся с такими простыми операциями. На самом базовом уровне, все эти четыре операции компьютер использует для вычисления. Таким образом, мы должны будем использовать эти четыре операции для выполнения таких задач, как сложение, вычитание, умножение, деление и сравнение. Как?

Прежде чем ответить на этот вопрос, давайте рассмотрим, как выполнить двоичное сложение с помощью нашего обычного оператора сложения +. Давайте выполним 0111 + 0110. Мы хотели бы выполнить это сложение подобным образом со сложением двух десятичных значений. Мы выстроим их, мы сложим их столбцы по столбцам в справа налево, записывая сумму, а в случае необходимости, перенося блок к следующему столбцу слева. Тем не менее, для двоичной системы, только цифры 0 и 1 нам позволено использовать, и наш перенос будет не в единицах, 10 единиц, но 2.

Перейдем теперь к решению 0111 + 0110. Решение показано на рисунке 3.6. Начнем с крайнего правого столбца. На этот столбец нет переноса (точнее, перенос в 0). Таким образом, мы хотим добавить 0 (перенос) + 1 + 0 (крайние правые цифры наших двух чисел). Что такое 0 + 1 + 0? Это 1. Таким образом, мы пишем сумму 1, не переносим. В следующем столбце, мы имеем 0 (перенос) + 1 + 1 0 + 1 + 1 равно 2. Тем не менее, нам нельзя писать 2, так как единственные цифры допустимые 0 и 1. Что такое 2 в двоичной системе? 10_2. Это говорит нам о том, что мы должны записать 0 (крайняя правая цифра 10_2), и перенос 1. Напомним, что перенос является степенью 2, поэтому мы проводим блок 2, а не 10. Тем не менее, мы пишем перенос как 1 разряд по следующей колонке. Теперь в третьем справа столбце, мы имеем 1 (перенос) + 1 + 1. Это 3. Опять же, мы не можем записать 3. Но 3 в двоичной системе 11_2. Таким образом, мы пишем 1 и переносим 1. В крайнем левом столбце мы имеем 1 (перенос) + 0 + 0 = 1. Таким образом, мы пишем 1 у нас нет никакого переноса. Перенос выхода из крайнего левого столбца может быть записан в виде флажков состояния (см. Главу 2) под битом

переноса. Это может быть полезно, потому что, если есть перенос, может сложиться ситуация, известная как переполнение. То есть, эти два числа были слишком большими, чтобы быть вписанными в место назначения, поэтому мы переполнили назначения (регистр или память).
Обратите внимание, на рисунке 3.6, что для каждого столбца, мы вычисляем две вещи. Вычисляем сумму трех цифр (включая перенос) и осуществляем перенос, который необходимо принять в следующей колонке в качестве переноса, мы всегда будем добавить три числа: две цифры от числа и переноса, мы будем всегда вычислить два различных результата, сумма этого столбца и перенос. Проведём для одного столбца перенос в следующей колонке.
Сумма алгоритма может быть выражена следующим образом:

1. Выстроить два двоичных числа.
2. Перенос в крайний правый столбец всегда 0.

1. Работаем справа налево, повторяем следующий, столбец за столбцом.

a. Вычислить сумму двух двоичных цифр плюс перенос, следующим образом:
Все три 0: 0
Один из трех является 1: 1
Два из трех являются 1: 0 (1 + 1 + 0 = 2, или 10, сумма 0, перенос 1)
Все три 1: 1 (1 + 1 + 1 = 3, или 11, сумма 1, перенос 1)

b. Вычисляется перенос из двух двоичных цифр плюс перенос,

Рисунок 3.6
Двоичное сложение, показывающее сумму и перенос в обе стороны.

следующим образом:
Все три 0: 0
Один из трех является 1: 0
Два из трех 1: 1
Все три 1: 1

c. Перенос из этого столбца становится переносом в следующий столбец.

Давайте попробуем. Мы сложим 1001 + 0011. Во-первых, мы выстраиваем номера:

```
  1001
+ 0001
```

Для крайнего правого столбца, перенос 0. Это дает нам 0, 1 и 1. В соответствии со стадией 3a выше, два 1 бита дают сумму 0. Согласно шагу 3b, два 1 бита дают нам перенос 1. Таким образом, мы имеем следующий частичный результат после одного столбца.

```
Цифра переноса: --10
                 1001
                +0011
                 ---0
```

Теперь переходим ко второму столбцу (справа), у нас есть перенос 1, а две цифры: 0 и 1, так что у нас есть два 1 бита. Это дает нам сумму 0 и перенос 1. Теперь наш частичный результат выглядит следующим образом.

```
Цифра переноса: -110
                 1001
                +0011
                 --00
```

Для третьего столбца, мы переносим в 1 и две цифры 0. Это дает нам сумму 1 и перенос 0. Теперь наш частичный результат выглядит следующим образом:

```
Цифра переноса: 0110
```

```
  1001
 +0011
  -100
```

С крайнего левого столбца мы можем перенести на 0 цифры 1 и 0. Это дает нам сумму 1 и перенос 0. Наша сумма в результате становится 1100. Если мы должны были преобразовать пример в десятичный вид, мы бы поняли, что мы только что выполнили пример 9 + 3 = 12.
Попробуйте выполнить несколько примеров самостоятельно и посмотреть, как вы это делаете.

0001 + 0101 = 0110
1001 + 0011 = 1100
0111 + 0011 = 1010

Теперь, как мы *реализуем* этот алгоритм сложения на компьютере, если у нас есть только AND, OR, NOT, и операции XOR? Мы будем разбивать наш алгоритм сложения на два вычисления, вычисления суммы и вычисления переноса. Мы начинаем с переноса, потому что его немного легче понять. Вспомним то, что изучали ранее, что переносом становится 1, если два или все три цифры 1. Назовем три значения X, Y и C (для переноса). Есть четыре возможных комбинаций при этом есть две или три 1. Это X = 1 и Y = 1, X = 1 и C = 1, Y = 1 и C = 1, или X = 1, Y = 1 и C = 1. На самом деле, мы не берем в расчет четвертую возможность, потому что три другие сравнения все равно будут правдивыми. Таким образом, мы можем определить, является ли перенос 1 с помощью следующей булевой логики: (X AND Y) OR (X AND C) OR (Y AND C).
Для того, чтобы вычислить сумму, мы должны знать, если есть либо одна 1 из трех битов (X, Y, C), или три 1 из трех битов. Мы можем определить это, умно сцепляя вместе две операции XOR:

(X XOR Y) XOR C

Давайте переделаем наш пример из рисунка 3.6, используя наши логические операции. Возьмем два номера 0111 + 0110. Начиная с крайнего правого столбца, мы берем X = 1, Y = 0, C = 0. Сумма – (X XOR Y) XOR C = (1 XOR 0) XOR 0 = 1 0 XOR = 1. Таким образом, у нас есть сумма 1. Переносом является (X AND Y) OR (X AND C) OR (Y AND C) = (1 AND 0) OR (1 AND 0) OR (0, 0) = 0 OR 0 OR 0 = 0. Таким образом, из

крайней правой колонки, мы имеем сумму 1 и перенос 0.
Во втором справа столбце, мы имеем X = 1, Y = 1, C = 0. Наша сумма (1 XOR 1) XOR 0 = 0 XOR 0 = 0. Наш перенос представляет собой (1 AND 1) OR (1 AND 0) OR (1 AND 0) = 1 OR 0 OR 0 = 1. Таким образом, мы имеем сумму 0 и перенос 1.
В третьем справа столбце, мы имеем X = 1, Y = 1, C = 1. Наша сумма (1 XOR 1) XOR 1 = 0 XOR 1 = 1. Наша перенос представляет собой (1 AND 1) OR (1 AND 1) OR (1 AND 1) = 1 OR 1, OR 1 = 1. Таким образом, мы имеем сумму 1 и перенос 1.
В нашем крайнем левом столбце, мы имеем X = 0, Y = 0, C = 1. Наша сумма (0 XOR 0) XOR 1 = 0 XOR 1 = 1, и наш перенос является (0 AND 0) OR (0 AND 1) OR (0 AND 1) = 0 OR 0 OR 0 = 0. Таким образом, наш крайний левый столбец имеет сумму 1 и нести 0.
Учитывая два двоичных числа, мы можем выполнять сложение с помощью AND, OR и XOR. Как насчет вычитания? Допустим, у нас есть два числа, X и Y, и мы хотим вычислить X - Y. Вы можете вспомнить из алгебры, что X - Y = X + (-Y). Предполагая, что X и Y - два числа, дополняющие друг друга, мы должны изменить Y, чтобы он стал -Y. Как мы преобразуем положительное число в отрицательное, или отрицательное число в положительное, в виде дополнения до двух? Мы переворачиваем все биты и добавляем 1. Как мы перевернем все биты ряда? Мы используем операцию NOT. Таким образом, в самом деле, X - Y = X + ((NOT Y) + 1). Таким образом, вычитание может использовать одни и те же операции, что и выше для суммирования и переноса, но мы добавим к нему операцию NOT сначала к Y, а затем добавим 1 к не Y. Мы добавляем 1 еще раз, используя операции суммирования и переноса. См Рисунок 3.7.
Умножение немного сложнее, но умножение является всего лишь рядом сложений (x * y требует сложения x между собой y раз). Деление представляет собой серию вычитаний (x / y требует вычитания y из x, пока мы не достигнем 0 или отрицательного значения). Таким образом, мы можем применить алгоритмы сложения и вычитания, описанные выше, многократно выполняя умножение и деление. Они используют комбинацию AND, OR, XOR и NOT.
Как насчет сравнения? На самом деле, это делается очень просто. Для сравнения двух значений X и Y, мы делаем X - Y и смотрим, будет ли результат положительным (в этом случае X> Y), отрицательным (в этом случае X <Y), или нулевым (в этом случае X = Y). Для проверки отрицательное значение или нет, смотрите на крайний левый бит

разницы (X - Y); если он равен 1, то разница отрицательна (напомним, в виде дополнения до двух, крайний левый бит является знаком), что означает, что X <Y. Для проверки нуля, необходимо OR все биты вместе и посмотреть, равен ли результат 0, что означает, что X - Y = 0 или X = Y. Например, если мы имеем 00000001, OR из этих битов равен 1, так что значение не равно нулю. С положительными значениями работать на самом деле наиболее трудно, чтобы определить это, мы должны убедиться, что результат не равен нулю и что крайний левый бит равен 0. Однако мы также можем определить, является ли число положительным, если оно не является ни отрицательным, ни нулю! Если разница положительна, то X - Y> 0 или X> Y.

Как сравнить что-то не принадлежащее к числам? Как правило, помимо цифр, мы также сравниваем другие вещи, мы можем сравнить символы и строки символов. Чтобы увидеть, что больше "Франк" и "Фред", мы сравним их посимвольно, пока мы либо не достигнем конец строки (в этом случае строки равны), либо у нас появится несоответствие. Здесь мы находим несоответствие в третьем символе 'a' против 'e'. Как мы вычислим 'a' - 'e'? Напомним, что символы хранятся в памяти, используя представление под названием ASCII. В ASCII, 'a' есть 96, а 'e' равно 100. Поэтому, чтобы сравнить "Фрэнк" с "Фред", мы делаем 'F' - 'F', который равен 0, точно также, как и 'r' - 'r' равен 0, так что мы берем 'a' - 'e', который является -4. Так как 'a' <'e', "Фрэнк" <"Фред".

ПОСЛАНИЕ ОТ ЗВЁЗД?

Физик Фрэнк Дрейк основал поиск внеземного разума (SETI). Эта работа, началась около 1961 года. он должен был использовать ра-

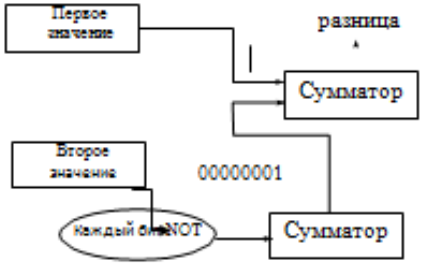


Рисунок 3.7
Вычитание двух чисел осуществляется двумя сложениями и NOT.

диотелескопы (и другие формы наблюдения), чтобы прослушивать сообщения от звезд. Какую форму может принять сообщение от инопланетян? Хотя мы не имеем ни малейшего представления, было высказано предположение, что любое сообщение будет закодировано в некотором двоичном представлении. Почему? Потому что информация легко передается в двоичной системе, можно просто отправить серию включения / выключения импульсов. Но как могло бы такое сообщение выглядеть? Инопланетяне должны выбрать представление, которое можно было бы легко интерпретировать. Например, мы не будем посылать сообщение в формате ASCII, потому что ASCII - строго человеческое изобретение, и не будет иметь смысла для инопланетян. Чтобы продемонстрировать, как может быть трудно расшифровывать сообщение, Дрейк предоставил следующее своим коллегам. Вы сможете выяснить, в чем его смысл? И они не смогли!

11110000101001000011001000000010000010100
10000011001011001111000001100001101000000
00100000100001000010001010100001000000000
00000000001000100000000000101100000000000
00000001000111011010110101000000000000000
00001001000011101010101000000000101010101
00000000011101010101110101100000001000000
00000000000100000000000001000100111111000
00111010000010110000011100000001000000000
10000000010000000111110000001011000101110
10000000110010111110101111100010011111001
00000000000111110000001011000111111100000
10000011000001100001000011000000011000101
001000111100101111

В другом типе логических операций, нам потребуются наши компьютеры для выполнения процесса, известного как маска. Маска сравнивает два двоичных значения вместе, используя один из AND, OR или XOR. Одно использование и маска покрыта в примерах с помощью двоичных чисел.

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ДВОИЧНЫХ ЧИСЕЛ

Здесь мы рассмотрим три очень разных способа, которыми двоичные

числа используются в компьютерах. В первом случае, мы смотрим двоичные числа, используемые в сетевой адресации и сетевые маски. Затем мы рассмотрим некоторые наиболее распространенные видео форматы представления файлов. И, наконец, мы рассмотрим использование двоичных чисел для обнаружения и исправления ошибок.

Сетевые адреса и сетевые маски

Компьютерные сети обеспечивают хороший пример того, как могут быть применены двоичные значения, которые вы уже узнали. Каждый компьютер в сети должен иметь уникальный сетевой адрес. IPv4 (протокол Интернета версии 4) адресация использует точечный десятичный формат для этих адресов.

IPv4-адрес состоит из четырех чисел, каждое называется октет. Каждый номер состоит из знака 8-разрядного двоичного числа. Напомним, что в 8 бит, вы можете хранить любое десятичное число от 0 до 255. Таким образом, IP-адрес, то есть четыре числа от 0 до 255, каждое число, разделенные точкой (период). Например, IP-адрес вашего компьютера может быть 10.251.136.253. Любой сетевой адрес работы просто представлен в виде одного 32-разрядного двоичного числа. Мы разобьем его на 4 октета, чтобы сделать его проще для чтения.
IP-адреса сгруппированы таким образом, что эти компьютеры в одном сетевом ресурсе, а некоторые из тех же октетов. Это похоже на то, как уличные адреса сгруппированы в почтовые коды. Компьютеры в пределах одной сети, могут взаимодействовать друг с другом без необходимости маршрутизатора. Маршрутизаторы используются для облегчения связи между сетями. Это похоже на то, как почтовые отделения используются для облегчения доставки почты между различными почтовыми индексами. Вы можете сказать, какие улицы адреса принадлежат к данному почтовому отделению на основе почтового индекса для этого адреса. Компьютеры имеют подобную концепцию. Скрытый код в пределах вашего IP-адреса является своего рода почтовым индексом, который является адресом сети. Мы называем это сетевым адресом.
Для того, чтобы определить, что сетевой адрес IP-адрес принадлежит, компьютеру используют сетевую маску. Давайте посмотрим на пример, где ваш IP-адрес 10.251.136.253, и ваша маска подсети является 255.255.255.0. Двоичное представление (с точками для облегчения чтения) этих чисел:IP адрес:

00001010.11111011.10001000.11111101
сетевая маска : 11111111.11111111.11111111.00000000

Для вычисления адреса сети, двоичное и операция выполняется между этими двумя 32-битовые числа, что приводит к сетевому адресу:

Сетевой адрес: 00001010.11111011.10001000.00000000 (10.251.136.0)

Напомним, что двоичной является - 1, только если оба двоичных бита являются 1. Это маска подсети имеет результатом возвращения первые три октета во всей их полноте, так как они в настоящее время операции заканчиваются на 11111111, с окончательным октетом, который маскируется, или обнулен, так как октет данной операции заканчивается на 00000000.

В приведенном выше примере, мы используем маски 255.255.255.0. Эта маска подсети выбрана потому, что все компьютеры в этой сети разделяют первые три октета. То есть, адрес сети первые три октета из четырех октетов. адрес каждого устройства в сети является окончательным октетом, таким образом давая каждому устройству уникальный адрес. Теперь, с маской 255.255.255.0, есть 256 (28) возможных адресов в этой сети. Таким образом, эта организация могла бы быть в состоянии предоставить 256 уникальных IP-адресов. В точечном десятичном формате, сетевой адрес вернулся из этой операции 10.251.136.0. Это обеспечивает сетевые адреса в диапазоне от 10.251.136.0 через 10.251.136.255, включая адрес нашего начального устройства от 10.251.136.253.

Два компьютера, которые совместно используют один и тот же сетевой адрес могут взаимодействовать друг с другом без необходимости маршрутизатора. Два компьютера с различными сетевыми адресами будут знать, что они должны связаться со своим дружественным маршрутизатором (часто упоминается как маршрут по умолчанию), чтобы общаться между этими двумя устройствами.

Это понятие легко понять, когда ваша маска подсети выравнивает равномерно с границами октет (т.е. точки), как это было в нашем примере выше. Тем не менее, давайте рассмотрим еще один пример, где это не так. Мы будем использовать один и тот же IP-адрес, 10.251.136.253, но с маской 255.255.240.0. Чтобы вычислить наш сетевой адрес, мы выполняем двоичную и из этих двух чисел. Рисунок 3.8 демонстрирует

применение сетевой маски 255.255.240.0, где наш сетевой адрес оказывается 10.251.128.0, а не 10.251.136.0.
Маска сети определяются количеством битов в адресе, которые составляют сетевую часть работы адреса. Некоторые сетевые адреса содержат только первый октет. Например, одна сеть может быть 95.0.0.0 (или 01011111.00000000.00000000.00000000). Эта сеть будет использовать маску подсети из 255.0.0.0.
Очевидно, что понимание двоичных чисел и их операции очень важно для тех, кто изучает сети. Сетевые специалисты должны свободно владеть понимания логических операций AND, OR, NOT и XOR. Мы детальнее изучим их подробное описание в главе 12.

Файлы изображений
Ранее в этой главе, мы увидели, что числа (положительное, отрицательное, целое, дробное) могут быть сохранены в двоичном виде либо непосредственно, либо через представления, такие как дополнение или с плавающей точкой. Мы также увидели, что символы представлены в двоичной системе с использованием кода под названием ASCII. Как изображения, хранящиеся в компьютере? Наиболее простой формат известен как растровое изображение. Сначала мы рассмотрим черно-белое изображение.
Любое изображение представляет собой совокупность точек света. Каждая отдельная точка называется пикселем (элемент изображения). Для черно-белого изображения, пиксель может взять на себя один из двух цветов, черный или белый. Мы можем представить эти два цвета, используя 1s и 0s (обычно, белый 1 и черный равен 0). Чтобы сохранить черно-белое изображение является лишь вопросом размещения в файл каждый пиксель, строка за строкой. Изображение на рисунке 3.9 представляет собой простой точечный рисунок. Фигура содержит как изображение и хранение в двоичном (справа от изображения).
Для черно-белого изображения X × Y пикселей (X строк, Y столбцы), хранение требуется X * Y бит. В 1000 × 1000 пикселей изображения требуется 1000000 бит. Это было бы 125000 байт, или около 125 КБ.
Разновидностью черно-белого изображения является серое изображение. Здесь, различные оттенки серого, добавляются в строго черно-белое изображение. Чтобы представить изображение, похожее на черно-белое изображение, каждый пиксель хранится в виде одного значения, но теперь значение находится в диапазоне от наименьшей интенсивности (черный) к большей интенсивности (белый). Любое значение между

является оттенком серого.

Чаще всего, уровень интенсивности будет целым числом от 0 до 255, так что каждый пиксель может быть сохранен в 1 байт. В 1000 × 1000 пикселей изображения потребуется 1000000 байт (примерно 1 МБ). Это в 8 раз превышает размер черно-белое изображение, которое имеет смысл, потому что мы увеличили требование хранения от 1 бита на пиксель до 1 байта на пиксель. Цвета представлены в компьютере на величину красного, зеленого и синего цветов, которые составляют свет этого пикселя. Мы называем эту стратегию как RGB. Для получения цветного изображения, мы будем хранить каждый пиксель как трио чисел, количество красного, количество зеленого цвета, и количество синего цвета. Как и в случае полутоновых изображений, мы будем хранить каждый из этих интенсивностей в виде целого числа в 1 байт, что дает нам диапазон 0-255. Цвет представлен (0, 0, 0), то есть, не красный, не зеленый, не синий, не черный. Цвет представлен (255, 255, 255) белый. Цвет представлен (255, 0, 255) фиолетовый (полный красный, не зеленый, полный синий). В двоичном виде, фиолетовый будет показано, как три числа (11111111, 00000000, 11111111). Для удобства, мы часто обозначают значения RGB вместо шестнадцатеричного. Так что фиолетовый FF00FF. Эта запись используется при разработке веб-страниц. Желтый, например, все зеленые и все синие, без красного, или (0, 255, 255), или 00FFFF.
Значение 884400 будет (128, 64, 00), то есть, было бы половина красного и одна четверть зеленого, коричневато-оранжевый цвет.
Для RGB, 1000 × 1000 пикселей изображения потребуется 3000000 байт или около 3 МБ. Обратите внимание, что с помощью RGB в качестве представления, каждый пиксель в изображении может быть одним из более чем 16 миллионов цветов (256 * 256 * 256).
Как видите, переходя от черно-белого до цветного сильно увеличивает размер файла. Существуют ли какие-либо способы, чтобы уменьшить

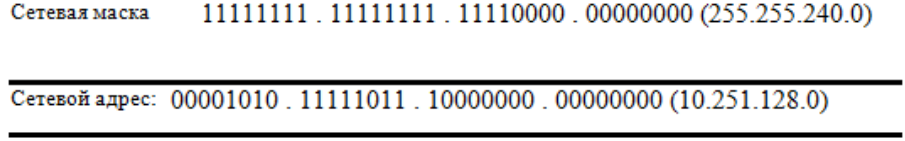


Рисунок 3.8
Применение маски подсети.

размеры файлов изображений? Да, с помощью сжатия файлов. Два наиболее широко распространенным формам сжатия являются GIF и JPG. GIF расшифровывается как Graphics Interchange Format. Идея заключается в том, что люди, как правило, не могут различать миллионы цветов, имеющимся в любом виде точного пути. Зачем использовать так много места для хранения, когда один оттенок красного может выглядеть как другой? В GIF, цветовая палитра при условии, что состоит из 256 самых популярных цветов. Это позволяет каждый пиксель ссылаться на цвет в палитре, используя только 1 байт вместо 3-х байтов (требуется для RGB); Таким образом, размер цветного изображения уменьшается на коэффициент 3. 1000 × 1000 пикселей цветное изображение теперь требует только 1 МБ дискового пространства. К сожалению, если исходное изображение использует цвета, которые не являются частью стандартной палитры, то изображение будет выглядеть по-разному, а в некоторых случаях вы получите очень неправильную замену цвета, например, синий вместо светло-желтого.

Формат JPG, возможно, чаще, чем GIF. JPG расшифровывается как Joint Photographic Experts Group (оригинал аббревиатура для JPG файла .jpeg). В GIF, если цвета доступны на палитре, результирующий файл GIF выглядит идентично файлу BMP и, следовательно, известный как сжатие без потерь. Но JPG использует сжатие с потерями. Разница заключается в том, что после того, как сжатие изображения, некоторое качество теряется. Вы жертвуете качеством изображения для хранения размера. Формат сжатия JPG работает путем объединения некоторых из пикселей вместе блокировочной групи пикселей. В действительности, изображение теряет некоторое разрешение и становится размытой. Кроме того, если изображение цветное, оно переводится из RGB в

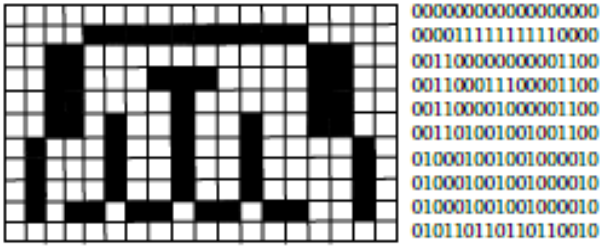


Рисунок 3.9
Черно-белый точечный рисунок и соответствующий двоичный файл.

альтернативный формат цвета, известный как YUV, где каждый пиксель представлен с использованием яркости (Y) и люминесценции (U и V). Этот формат требует либо два или четыре раза меньше места, чем RGB. Кроме того, сжатие с потерями в формате JPG может уменьшить размер изображения в еще большей степени.
Поздние PNG (Portable Network Graphics) расширяет формат GIF. Как и GIF, это без потерь форма сжатия, которая использует битовую карту. Он может поддерживать 24-битный RGB-палитра, градации серого и растровых изображений без палитр. И как JPG, он сжимает изображения, чтобы уменьшить пространство, но с использованием алгоритма без потерь так, что сжатое изображение имело одинаковое качество. В отличие от JPG, PNG сжатие может или не может добиться успеха в самом деле уменьшения размера файла.

Обнаружение и исправление ошибок
Одной важной задачей при работе с компьютером это сохранение информации при обработке ошибок, возникающих при передаче данных из одного места в другое. Это маловероятно, но возможно появление ошибок, когда данные перемещаются из одного компонента к другому в компьютере, но это гораздо более вероятно, и даже ожидать, что ошибки могут возникнуть во время процесса передачи. Таким образом, некоторые формы обнаружения и исправления ошибок приветствуются.
Самой простой формой обнаружения ошибок посредством использования четности. В общем, контроль четности означает равенство. Но при обсуждении компьютера, четности относится к ровности (или нечетности) ряда. В частности, мы будем подсчитывать количество 1 бит в последовательности данных и посмотреть, если счетчик является четным или нечетным. Затем мы можем использовать эту функцию, чтобы обнаружить простые ошибки при передаче.
Введем бит четности для каждого байта (8 бит) памяти. Бит четности будет равен 1, если число 1 бит в байте нечетно, и бит четности будет равен 0, если число 1 бит в байте четное. То есть, 8 битов плюс бит четности всегда должен иметь четное число 1 бит. Эта стратегия известна как число четности (существует вариация - называется нечетность, где 8 бит плюс бит четности всегда должен иметь нечетное число 1 бит).
Мы будем использовать даже паритет следующим образом. Учитывая байт, до этого ИГД перемещается, мы будем генерировать бит четности. Теперь у нас есть 9 бит. Мы передаем / переместить 9 бит. Получатель получает 9 бит. Если 9 бит имеет четность, то ошибки не произошло.

Если 9 бит имеет нечетный, произошла ошибка, ИГД отбрасывается, а получатель просит, чтобы данная величина будет возмущаться.
Как мы можем определить, четный или нечетный паритет? Это очень легко и просто с помощью операции XOR (см обратно рисунок 3.5). Операция XOR действует только на 2 бита за один раз, но мы можем цепи вместе операции XOR. Таким образом, данные 8 бит, мы XOR первые два бита, а затем XOR результата с третьим битом, а затем XOR результата с четвертого бита, и так далее. Давайте посмотрим на некоторые примеры вычислений и используя бит четности. Мы хотим, чтобы переместить опорную точку 10010011 в другое место на компьютере или по сети. Первый шаг должен вычислить бит четности:
стр. 85

(((((((1 XOR 0) XOR 0) XOR 1) XOR 0) XOR 0) XOR 1) XOR 1) =

((((((1 XOR 0) XOR 1) XOR 0) XOR 0) XOR 1) XOR 1) =

((((((1 XOR 1) XOR 0) XOR 0) XOR 1) XOR 1) =

((((0 XOR 0) XOR 0) XOR 1) XOR 1) =

(((0 XOR 0) XOR 1) XOR 1) =

((0 XOR 1) XOR 1) =

(1 XOR 1) = 0

Поэтому 10010011 имеет бит четности 0. Это имеет смысл, так как общее число 1 бит должно быть четным. 10010011 имеет четыре 1 бита, и таким образом, чтобы сохранить число 1 бит даже, мы добавляем 0. Теперь, 9 бит состоят из четырех 1s и пяти 0s *
Мы передаем эти 9 бит и получатель получает 10010011 & 0. Получатель выполняет аналогичную операцию, XORing все 9 битов вместе, и ветры с 0. Поэтому, правильно, то есть, не возникло никаких ошибок.
Теперь рассмотрим, если при получении, 9 битов были 10010011 & 1. В этом случае исключающее число вычисляет:

(((((((1 XOR 0) XOR 0) XOR 1) XOR 0) XOR 0) XOR 1) XOR 1) XOR 1) = 1

Так возникла ошибка. Где же возникла ошибка? Используя один бит четности, мы не знаем, а потому, что есть ошибка, получатель отбрасывает данные и просит, чтобы он был повторно отправлен. Обратите внимание, что в этом случае возникла ошибка четности бит сам. Другим примером может быть 10010010 & 0, где бит четности не содержит ошибки, но этот байт имеет ошибку (последний бит должен быть равен 1). Получатель вычисляет XOR этих 9 бит и получает

(((((((1 XOR 0) XOR 0) XOR 1) XOR 0) XOR 0) XOR 1) XOR 0) XOR 0) = 1

Обратите внимание на то, что бит четности не только не помогает нам определить ошибку, но и не оставит нас, если возникли две ошибки. Представьте, что получатель получает 10010000 & 0. В этом случае, последние 2 бита неверны. Если мы XOR эти 9 бит, мы получаем 0, это означает, что ошибки не произошло. Таким образом, бит четности может обнаружить ошибки 1, но не 2, и не может исправить ошибку. Если мы хотим, исправить ошибки, нам необходимо будет больше, чем 1 бит четности.

Кроме того, бит четности, не обязательно должны быть связаны с байтом. Напомним, ранее в этой главе, что значения ASCII используют 7 битов байта. Восьмой бит может быть использован для хранения информации о четности. Например, буква «a» представляется как x1100001. Первоначально x не используется, поэтому мы обычно поместить 0 там. Тем не менее, чтобы не тратить бит, мы можем положить немного четности там. Так что, на самом деле, 'a' будет сохранен как 11100001, если мы хотим, добавить бит четности. Это избавляет нас от необходимости добавить еще один бит.

Другое использование паритета поставляется с RAID технологии. RAID представляет собой новую форма дискового устройства для хранения данных, где есть несколько дисков в одном месте. Идея заключается в том, что дополнительные диски могут обеспечить как избыточное хранение данных и возможность доступа нескольких дисков одновременно, чтобы ускорить доступ к диску. RAID обсуждается более подробно в главе 15 (обеспечения сохранности информации и безопасности). Здесь мы кратко рассмотрим, как соотношение может быть использовано для обеспечения коррекции ошибок.

Представьте себе, что у нас есть пять дисковых накопителей. Мы будем хранить 4 байта данных на четырех разных дисках. Пятый диск будет

хранить информацию о четности из 4 байтов. Для каждого 4-байтового группировке, которая распределена по всем дискам, мы также будем вычислять информацию о четности и хранить его на пятом диске. Рассмотрим один 4-байтовый группу данных, следующим образом:

11001111	01010101	11011011	00000011

Мы будем вычислять побитовое XOR по каждому из 4 байтов. Это означает, что мы будем делать первые биты (самого левого бита) 4 байта и вычислить XOR этих 4 бита. Это будет бит четности для первого бита из четырех байтов. Мы проведем данную операцию для каждого бита из 4 байта.

	11001111	//Первый байт, хранится на первом диске
XOR	01010101	// Второй байт, хранится на втором диске
XOR	11011011	// Третий байт, хранится на третьем диске
XOR	00000011	//Четвертый байт, хранится на четвертом диске

Для получения информации о четности, то мы вычисляем каждый XOR идет вниз. Первый бит четности (((1 XOR 0) XOR 1) XOR 0) = 0. Второй бит четности вычисляется как (((1 XOR1) XOR1) XOR0) =1. Третий бит четности вычисляется как (((0 XOR 0) XOR 0) XOR 0) = 0. Четвертый бит четности вычисляется как (((0 XOR 1) XOR 1) XOR 0) = 0. Пятый бит четности вычисляется как (((1 XOR 0) XOR 1) XOR 0) = 0. Шестой бит четности вычисляется как (((1 XOR 0 XOR XOR 1) = 0. Седьмой бит четности вычисляется как (((1 XOR 0) XOR 1 XOR 1) = 1. Бит четности восемь вычисляется как (((1 XOR 1) XOR 1 XOR 1) = 0. Таким образом, наша четности байт 01000010. Этот байт хранится на пятом диске.

Теперь, давайте предположим, в течение долгого времени, что второй диск терпит неудачу, возможно, есть плохой сектор, или, возможно, вся поверхность диска повреждена. В любом случае, мы можем восстановить все данные с помощью трех других дисков с данными и диск четности. Для наших 4 байта выше, мы имели бы следующие доступные данные:

11001111 // первый байт, хранится на первом диске

XXXXXXXX // второй байт данных поврежден
11011011 // третий байт, хранится на третьем диске
00000011 // четвертый байт, хранится на четвертом диске
01000010 //четности байтов, хранится на пятом диск

XOR четыре соответствующие биты трех байтов плюс четности байта для получения недостающих данных. Например, используя первые биты каждого байта, то есть (((1 XOR 1) XOR 0 XOR0) = 0. Таким образом, первый бит нашей недостающим сведением равно 0. Второй бит будет (((XOR 1) XOR 0) XOR 1) = 1. Смотрите, если вы можете работать через оставшуюся часть этого примера, чтобы получить всю недостающую точку привязки.
Использование одного бита четности на байт иногда называют в качестве проверки горизонтального резервирования, так как бит четности вычисляется по блоку данных. Другой формой горизонтальной проверки избыточности является контрольная сумма (кратко упоминается в главе 12). Вычисление четности используется для RAID иногда называют вертикальной избыточного кода, так как вычисление четности выполняется через некоторое количество байтов, бит за битом.

Дальнейшее чтение
Изучение двоичных представлений можно найти почти на каждом компьютере организационной книги (см раздел Компьютеры и сборка компьютеров (установка) в главе 2). Текст Kindle, Binary Made Simple (R. Barton), доступный от Amazon Digital Services предлагает полезный советы, как выполнять преобразования между двоичной, десятичной и шестнадцатеричном формате. Более тщательные книги по нумерации систем обеспечивают фон на теории и вычисления, такие как следующие:

1. Конвей, Дж. Книга чисел, Нью-Йорк: Springer, 1995.
2. Гюлберг, Дж. и Hilton, П. Математика: С рождения чисел, Нью-Йорк: Нортон, 1997.
3. Ифра, Г. Всеобщая история чисел: С древности до изобретения компьютера. Нью-Йорк: Wiley, 1999.
4. Нивен И., Цукерман, Х. и Монтгомери, Г. Введение в теорию чисел, Нью-Йорк: Wiley, 1991.
Логические операции (двоичный) и логика также рассматриваются в компьютерной организации текстов.
Вы также можете найти целые тексты по этой теме. Несколько текстов на

выбор перечислены здесь.
5. Браун, Ф. Логическое Рассуждение: Логика булевых уравнений. Нью-Йорк: Довер, 2012.
6. Грегг, Дж. и нолей: Понимание Булева алгебра, цифровых схем, и логика множеств. Piscataway, NJ: Wiley-IEEE Press, 1998.
7. Вайситт, Дж. Булина алгебра и ее приложения. Нью-Йорк: Довер, 2010.
В 12 главе расписано о компьютерных сетях подробно, читайте эту главу для получения информации о таких вещах, как IP-адреса и сетевые маски.
Всесторонняя ссылка на основные формы компьютерных графических файлов доступна:
8. Милано, Дж., Сжатые форматы изображений: JPEG, PNG, GIF, XBM, BMP. Reading, MA: Addison Wesley, 1999.

ГЛАВА 4.
Введение в операционные системы

В главе 2 мы рассмотрели аппаратные средства компьютера. Для того, чтобы облегчить связь между пользователем и аппаратным средством, компьютерные системы используют операционную систему. В этой главе предоставлены многочисленные концепции операционной системы, многие из которых имеются ссылки в последующих главах этой книги. В частности, роли управления процессами, управление ресурсами, пользовательский интерфейс, защиты и безопасности введены здесь, так что последующие осмотры могут сосредоточиться на том, как использовать их в операционных системах Windows, и Linux. Операционная система понятий прерываний, то переключение контекста, процесс загрузки, а также администратор также введены. Глава завершается. Шаги, необходимые для установки системы Рабочее для Windows 7 и Linux Red Hat.

Цели обучения данной главы являются
1. Опишите роль операционной системы.
2. Сравните графический пользовательский интерфейс и интерфейс командной строки.
3. Различия между формами управления процессами.
4. Обсудите прерывание и переключение контекста.
5. Опишите виртуальную память.
6.Обсудите вопросы, связанные с управлением ресурсами: синхронизация и тупиковой ситуации.
7. Опишите процесс загрузки и инициализации системы.
8. Обеспечить шаг за шагом инструкции на Windows 7 и установки Linux Red Hat.

Что такое операционная система?

В самом простом описании, операционная система (ОС) представляет собой программу. Ее основная задача состоит в том, чтобы позволить

пользователю компьютера легко получить доступ к оборудованию и программному обеспечению компьютерной системы. Помимо этого, мы могли бы сказать, что ОС необходима для поддержания среды компьютера. ОС для контроля и удобства. Операционная система поддерживает управление тем, что она позволяет пользователю контролировать действия программного обеспечения, а также с помощью программного обеспечения для управления (или доступа) аппаратного обеспечения. Операционная система обеспечивает удобство в том, что обеспечивает доступ в простой в использовании способ. Ранние операционные системы не были легки в использовании, но сегодня ОС используют графический пользовательский интерфейс (GUI), и команды могут быть выданы с помощью перетаскивания, указывая, нажав и двойной щелчок мышью. В некоторых операционных системах, таких как Linux и Unix, существует два способа выполнения команд: через графический интерфейс и введя в команду в командной строке. В действительности, как Windows, и Mac ОС также есть командная строка подсказки, но большинство пользователей никогда не возиться с ними. Linux и Unix пользователи, все же, часто предпочитают команду линия над GUI.

Некоторые полезные Условия

В предыдущих главах мы уже ознакомили вас с компонентами любой компьютерной системы. Давайте вернемся к этим определениям, с новыми подробностями по мере необходимости. Введем также некоторые термины ОС.

Аппаратные средства

Физические компоненты компьютера. Мы можем разбить эти компоненты на две категории: внутренние компоненты и периферийные устройства. Внутренние компоненты предварительно доминантно, расположенной на материнской плате, кусок пластика или стекловолокна, которая соединяет устройства вместе. На материнской плате несколько розеток для чипов и плат расширения. Один чип процессора, другие чипы составляют память. Процессор CPU; является компонентом в компьютере, который выполняет программу. Процессор выполняет то, что известно, как Выборки-выполнения цикла, выборка каждой команды программы, по одному за раз из памяти, и выполнение его в CPU. Есть несколько форм памяти: кэш-памяти (SRAM), динамическое ОЗУ и ПЗУ. Кэш бы-

строй памяти, DRAM медленная память, но кэш, как правило, гораздо дороже, чем DRAM, поэтому наши компьютеры имеют небольшой кэш и много памяти DRAM. ПЗУ только для чтения памяти и используется только для хранения несколько важных программ, которые никогда не меняются (например, программы загрузки). Подключение устройств друг к другу как автобус, который позволяет информации перейти от одного компонента к другому. Слоты расширения позволяют подключить карты расширения, которые либо являются периферийными устройствами сами (например, беспроводного модема) или подключаться к периферийным устройствам через порты на задней панели компьютера. Некоторые из наиболее важных периферийных устройств жесткий диск, клавиатура, мышь и монитор. Вполне вероятно, что вы также будете иметь накопитель на оптических дисках, беспроводной модем (или, возможно, сетевую карту), звуковую карту и внешние динамики, и возможно принтер.

Программное обеспечение

Программы, которые работают на компьютере, упомянутые в качестве программного обеспечения, чтобы отличить их от физических компонентов компьютера (аппаратное обеспечение). Программное обеспечение существует только в виде электрического тока, протекающий через компьютер (или в виде магнитных зарядов хранится на диске). Программное обеспечение, как правило, классифицируются как прикладное программное обеспечение (программы, которые мы бежим, чтобы выполнить какую-то задачу) или системного программного обеспечения (программ, что компьютер работает, чтобы сохранить свою среду). Мы в первую очередь будем изучать системы программного обеспечения в этом тексте. В обоих Linux и Windows, операционная система состоит из множества небольших программ. Например, в Linux, существуют программы, называемые Ls, RM, CD, и так далее, которые выполняются при выполнении команды в командной строке. Эти программы также называются некоторые из инструментов с графическим интерфейсом, такие как файловый менеджер. Если вы посмотрите на Linux каталоги / BIN, / USR / BIN, / SBIN и / USR / SBIN, вы найдете много программ ОС. Многие пользователи редко взаимодействуют непосредственно с операционной системой с помощью вызова этих программ. Если вы посмотрите на каталоги, такие как / USR / LOCAL /BIN, вы найдете некоторые из приложений программного обеспечения. Прикладное программное обе-

спечение включает в себя текстовые процессоры, электронные таблицы, интернет-браузеров и компьютерные игры. В операционной системе Windows вы найдете прикладное программное обеспечение в директории Program Files и ОС в каталоге Windows.

Все программное обеспечение написано на языке компьютерного программирования, таких как Java или C ++. Программа, написанная человеком, называется исходный код программы или исходный код. Java и C ++ выглядит как английский язык в сочетании с математическими символами. Компьютер не понимает исходный код. Таким образом, программист должен запустить программу перевода, которая называется компилятор . Компилятор принимает исходный код и переводит его в машинный язык. Языковая версия машины для выполнения программы. Существует еще один класс языковых переводчиков, известных в качестве переводчика. Переводчик немного отличается тем, что она сочетает в себе трансляционный код с исполнением кода, так что запуск, интерпретируемый программы занимает больше времени. Поскольку это менее эффективно, мы склонны использовать интерпретируемый подход на более мелких программах , таких как скрипты оболочки, скрипты веб-браузер и веб-сервера сценариев. Как ИТ-студент, вы будете в первую очередь писать код, который будет интерпретирован. Языки сценариев включают язык сценариев оболочки Bash, PHP, Perl, JavaScript, Python и Ruby.

Компьютерная система

Коллекция компьютерного оборудования, программного обеспечения, пользователь(я) и сети. Наша ОС в действительности набор программных компонентов. Мы могли бы разделить эти компоненты на четыре категории, описанные ниже.

Ядро

Ядро ОС; мы выделяем эту часть из других частей, которые делятся на компоненты: оболочками, драйверы устройств, утилиты. Ядро будет включать в себя программы, которые выполняют основные задачи ОС: управление процессами, управление ресурсами, управление памятью, управление файлами, защиту и безопасность (они рассматриваются в следующем разделе).

Драйверы устройств

Программы, которые являются специальными интерфейсами между ОС под управлением компьютера и аппаратного обеспечения. Вы устанавливаете драйвер устройства для каждого нового оборудования, которое вы добавляете к вашей компьютерной системы. Драйверы, как правило идут в комплекте с аппаратными средствами или доступны для загрузки через Интернет. Многие обычные драйвера уже являются частью операционной системы, только требует установку. Реже драйвера могут потребовать загрузки от компакт-диска или через Интернет.

Оболочка

Интерфейс для пользователя, часто персонализированный для этого конкретного пользователя. Оболочка обеспечивает доступ к ядру. Например, GUI оболочка будет переводить движения мыши на вызовы ядра процедуры (например, открыть файл, запустить программу, переместить файл). Linux и Unix имеют оболочках командной строки, а также, что может включать в себя команды редактирования строки (например, используя Control + e, чтобы перейти к концу строки, или Control + B для резервного копирования на одну позицию). Популярные оболочки GUI в Linux / Unix включают CDE, GNOME и KDE, а также популярные текстовые оболочки в Linux / Unix включают Баш (Bourne-Again Shell), Ksh (оболочки Korn) и C оболочки (CSH). В тексте на основе оболочки включают интерпретатор командной строки, историю, переменные окружения и псевдонимы. Мы рассмотрим оболочку Bash подробнее в главе 9.

Утилиты

Программное обеспечение, которое помогает управлять и настраивать аппаратные средства, операционной системы и прикладного программного обеспечения. Утилиты сильно варьируются в функции и могут включать в себя утилиты файл (например, Windows, программа Проводник окна), противовирусное программное обеспечение, сжатие файлов / и дефрагментацию диска.

Виртуальные машины

Программный эмулятор — это программа, которая эмулирует другую операционную систему, то есть, это позволяет вашему компьютеру действовать, как другой компьютер. Это, в свою очередь позволяет пользователю запускать программное обеспечение, которое не является родным или скомпилированы для его / ее компьютера, но создано для другой платформы. Пользователь Макинтош, например, может использовать эмулятор Windows, для запуска программного обеспечения для Windows на Macintosh.
Сегодня мы склонны использовать виртуальные машины (VM), а не автономные эмуляторы. виртуальная машина программное обеспечение, которое симулирует быть аппаратным. Устанавливая различные операционные системы в Vm, пользователь затем имеет возможность запускать программное обеспечение для этого используя другую платформу УСС в своем компьютере, которые иначе не могли бы иметь возможность запускать это программное обеспечение.
Тем не менее, использование виртуальных машин дает значительно большую гибкость, чем просто обеспечивая платформу для программной эмуляции. Рассмотрим эти виды использования виртуальной машины.

1. Есть среда, которая является безопасной, при загруженности программного обеспечения не может влиять или повлиять на физический компьютер
2. Есть среда для изучения различных платформ в случае, если вы заинтересованы в приобретении различных ОС или различные типы компьютерного оборудования
3. Есть среда, в которой вы можете выдавать команды администратора, и все же не беспокоиться, что вы можете повредить физический компьютер, если вы допустили ошибку, удалить Vm и создать новый!
4. Есть среда, в которой несколько пользователей могли получить доступ к нему для поддержки совместной работы
5. Есть условия, при которых можно было бы получить удаленный доступ

Существуют различные типы VMS. Виртуальная машина Java (JVM) существует в веб-браузерах для выполнения Java-кода. Программное обеспечение Ут, такие как VirtualBox (от солнца) и VSphere (от VmWare),

предоставляют пользователям возможность расширить их компьютерного оборудования для поддержки нескольких ОС в безопасной среде.

Пользовательский интерфейс

Графический интерфейс позволяет пользователю управлять компьютером с помощью мыши и наведения, и щелчка на объектах на экране (иконки, меню, кнопок и т.д.). Каждая операционная система предлагает различные "устройства". Например, Mac OS имеет стандартный набор меню, всегда перечисленных в верхней части рабочего стола и каждое приложение добавляет к пунктам меню. В Windows 7 нет меню выбора рабочего стола, но вместо этого каждое название программного обеспечения имеет свой собственный набор меню. Linux ГПИ включают Ghrome и KDE. Каждый из этих операционных систем обеспечивает значки на рабочем столе. Ярлыки и каждое окно можно свернуть, развернуть и есть кнопка закрытия. Кроме того, каждый из этих интерфейсов основан на идее "точки и нажмите кнопку". Мышь используется для позиционирования курсором, а кнопки мыши используются для определения операций через единый щелчок, двойной щелчок и перетаскивание. Сотовые телефоны и планшеты, основанные на сенсорных экранах, имеют gesture- интерфейс, где движения включают в себя такие операции, как салфетки, кран, масштабирование, так и в обратном крайнем случае. Windows 8 в настоящее время с сенсорным экраном, а не точки и щелчки по кнопке.
GUI является гораздо более простой способ для управления компьютером, чем с помощью выдачи команд с помощью командной строки. Командная строка в большинстве ОС работает в оболочке. Оболочка является лишь частью операционной системы, которая позволяет пользователям вводить информацию, с помощью которой пользователь может управлять ядром операционной системы. Оболочка содержит текстовый интерпретатор, он интерпретирует текст, более подходящий для выполнения команды в ядре операционной системы. В качестве примера, пользователь Linux может ввести команду типа:

```
find ~ -name 'core*' -exec rm {} \;
```

Эта команда выполняет команду найти Linux, чтобы найти, в домашнем направлении теории этого пользователя, что-нибудь с именем, которое начинается с ядра, а затем выполняет команду rm на любых таких най-

денных файлах. Другими словами, эта команда находит и удаляет все файлы, начинающиеся с букв "ядра", найденных в домашнем каталоге пользователя. В частности, инструкция работает следующим образом:

1. Задача «поиска», в которой имеется несколько параметров, чтобы определить, как «поиск» должен работать.
2. ~ / указывает, где должны искать (в домашнем каталоге пользователя).
3. - name 'ядро *' являются парой, указывая, имя файла и содержать буквы "ядро", (The является подстановочным символом).
4. -exec указывает на то, что любой найденный файл должен иметь инструкцию, которая выполняет действие на нем.
5. rm операция удаления, {} указывает найденный файл.
6. \; заканчивает инструкцию.

1. Что это множественное число от ОС? Есть несколько вариантов, я выбрал ОС из удобства, но некоторые люди этого не любят, потому что ОСС выступает за Open Source Software. Другие люди выбирают операционную систему OS. Для обсуждения смотрите http://weblogs.mozillazine.org/gerv/archives/007925.html.

Многие переводчики командной строки ОС (например, DOS, которая доступна в ОС Windows) гораздо более упрощены . Linux является фаворитом некоторых компьютерных пользователей из-за способности выполнить очень сложные инструкции. Вы будете изучать интерпретатор Bash в главе 9, где вы узнаете больше о том, как работает интерпретатор командной строки.

Управление процессом

Основной задачей любого компьютера является запуск программы. Программа выполняется на компьютере, называется процессом. Причина различать программы и процессы является то, что программа представляет собой статический субъект, в то время как процесс является активным объектом. Процесс имеет статус. Его статус может включать в себя "работает" (процесс в настоящее время выполняется процессором), "ожидание" (процесс ожидает ввода или вывода, или в режиме ожидания, чтобы был загружен в память), или "готов" (процесс загружается в память, но в настоящее время не выполняются процессором). Этот процесс также имеет специфические данные, хранящиеся в памяти,

кэш-памяти и регистров. Эти данные изменяются с течением времени и от исполнения к исполнению. Таким образом, этот процесс является динамическим.

Компьютер может выполнять один процесс в одно время, или несколько процессов в той или иной форме перекрытого (параллельной) стиля.

Операционная система отвечает за начало процесса, наблюдая, как он выполняет, обработку прерывающих ситуаций (объяснено ниже) и ввода / вывода (I / O) операций, обработка множественного взаимодействия процессов и завершения процессов.

Прерывание — это ситуация, когда процессор прерывается из его выборки-выполнения цикла. Как уже говорилось в главе 2, центральный процессор непрерывно считывает, декодирует и выполняет инструкции, данные из памяти. Оставляя за собой возможность, процессор будет делать это до бесконечности. Тем не менее, бывают случаи, когда внимание процессора должна быть смещена от текущего процесса к другому процессу (включая операционную систему) или обратиться к другому оборудованию. Таким образом, в конце каждого выборки-выполнения цикла, процессор анализирует флаг прерывания (часть состояния регистра флагов), чтобы увидеть, если кто-то поднял прерывание. Сигнал прерывания может исходить от аппаратных средств или с помощью программы.

Если возникает прерывание, процессор обрабатывает прерывание следующим образом. Во-первых, сохраняется информация процесса, который был запущен. Для этого необходимо принимать значения различных регистров и сохранять их в памяти. Во-вторых, процессор определяет причину прерывания. Для этого требуется процессор, чтобы определить, какое устройство было задействовано в прерывании, или, если прерывание было вызвано программным обеспечением. В-третьих, процессор переходит к выполнению ОС. В частности, центральный процессор начинает выполнение обработки прерывания. Операционная система будет иметь обработку прерывания (набор кода) для каждого типа прерывания. Смотрите в таблица 4.1 представлен список типов прерываний. Затем CPU возобновляет выполнение выборки-цикла, но теперь выполняет обработка прерывания ОС. По завершении выполнения цикла, который позаботился о прерывании операции, процессор восстанавливает значения регистров, сохраненные в памяти. Это «освежает» прерванный процесс. Затем процессор возобновляет выполнение выборки-цикла, но теперь он продолжает работу с процессом, который был ранее написан, как если бы он никогда не останавливался.

Таблица 4.1 Типы прерываний

Причина	Устройство(а) для прерывания
Диск	Файл не найден
	диск не в дисковод
	диск не отформатирован
Клавиатура	Пользователь вводит нажатие клавиши
	пользователь нажимает Ctrl + Alt + Del
Мышь	Мышь перемещается
	Нажатие кнопки мыши или отпустив ее
Сеть / Модем	Сообщение прибывает
Принтер	Застревание бумаги для печати
	Закончилась бумага
	Распечатка завершена
Программы	Ошибка во время прогона программы (например, деление на 0, вход плохой пользователь)
	Требуется связь с другой программой
Таймер	Таймер истекает

Прерывание может быть принято только несколько циклов машин или секунд до нескольких минут, если прерывание вовлекает пользователя. Например, если прерывание было вызвано "диск не в приводе", то пользователь должен физически вставить диск. Прерывание будет решено только после того, как пользователь вступил в действие.
ОС будет осуществлять управление процессом по-разному. Самый простой подход заключается в использовании одного задания, где начинается процесс, выполняется, и завершается без запуска каких-либо других процессов.

Это не очень удовлетворительное использование компьютерных ресурсов, однако, пользователи, как правило, не хотят быть ограничены использованием одной программы, в данное время. Различные формы параллельной обработки существуют. Параллелизм означает, что процессы выполняются в некоторых перекрытых стилях. К ним относятся мультипрограммирования, многозадачности (или совместное использование, как он первоначально назывался), многопоточность и многопроцессорность. Каждого из них объединяет то, что операционная система позволяет нескольким процессам быть в некотором состоянии выполнения. За исключением многопроцессорной обработки (который использует несколько процессоров), есть только один процессор, так что только один процессуальному может быть выполнение в любой момент

времени. Остальные ждут. Как долго они будут ждать, почему они ждут, и где они ожидают отличается в некоторой степени в зависимости от типа управления процессами.
Мы будем изучать различные формы управления процессами в следующем разделе. Однако, прежде чем продолжить, мы должны сначала ввести многозадачность, как это упоминается далее в этом разделе. В многозадачности, есть два или более процессов, запущенных. Процессор выполняет только один процесс в любой момент, но выключается между всеми запущенными процессами быстро. Так, например, ЦПУ может выполнять несколько тысяч циклов машины каждый по процессу 0, а затем процесс 1, а затем процесс 2, а затем возвращать на обработку 0. Таймер используется для времени, сколько времени выполняется каждый процесс. Когда таймер истечет, он прерывает процессор и ОС, то вызывается, чтобы заставить переключаться между процессами. Таймер сбрасывается и процессор переходит к следующему процессу.

Планирование

Если мы хотим, чтобы компьютер не выполнял ничего, кроме одной обработки, то есть, запуск одной программы, пока она не будет завершена, а затем перейти к следующей задаче пользователя, операционная система будет выполнять планирование. Есть несколько типов графиков. Например, в многозадачном режиме, когда таймер истекает, центральный процессор переключается с текущего процесса к другому. Который из? Как правило, операционная система использует циклический планировщик. Если есть несколько процессов, работающем, они будут помещены в очередь. Процессы затем приведены внимание CPU в том порядке, что они находятся в очереди. Когда процессор выполнил некоторое количество инструкций по последнему процессу в очереди, возобновляется с первым процессом в очереди. То есть, очередь очереди "наматывается вокруг", таким образом, термин Круговой. Операционная система также отвечает за определение того, какие процессы должны быть загружены в память в любое время. Только процессы, присутствующие в памяти, будут помещены в очередь. Обычно сегодня, все процессы будут загружены по требованию, но с низким приоритетом процессы могут быть удалены из очереди и готовой памяти и находиться в очереди ожидания. И, наконец, пользователь может также указать, что процесс должен быть запущен. Это может быть сделано путем выбора, например, процесс с вкладки в нижней части рабочего стола. Это заставляет

процесс перехода от фона на передний план. Мы будем рассматривать планирование более подробно в следующем разделе, а также в главе 11.

Управление памятью

Поскольку основная память имеет меньший размер, чем размер программного обеспечения, как правило, мы хотим работать, операционная система отвечает за перемещение частей программ и данных в и из памяти по мере необходимости. Каждая программа (наряду с его данными) делится на фиксированные размеры блоков, называемых страницами. Перед тем, как программа может быть выполнена, копии ОС "образ" программы в файл подкачки и загружает некоторые его страницы в память. "Образ" является исполняемый код программы вместе с пространством памяти, которая составляет данные о том, что программа будет иметь доступ. Пространство подкачки представляет собой зарезервированную область жесткого диска. Теперь, как проходит

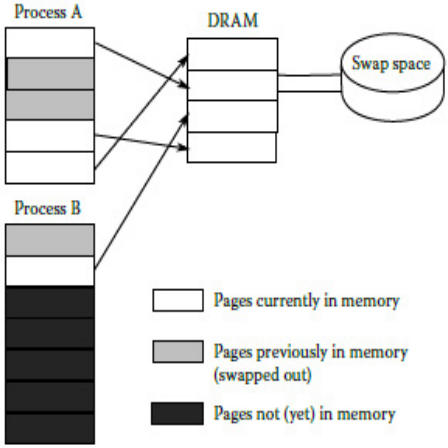


Рисунок 4.
Виртуальная память

Перевод:
DRAM – Динамическое ОЗУ
Swapspace – пространство свопинга
Pagescurrentlyinmemory – страницы в памяти в данный момент
Pages previously in memory (swapped out)- Страницы заранее в памяти (выгруженные)
Pages not (yet) in memory – страницы не (еще) в памяти.

процесс, только необходимые страницы загружаются в память. Это позволяет использование памяти каждой программы вниз так, что память может сохранить больше программ. Если каждая программа должны были быть загружены в память в полном объеме, память будет заполнить быстро и ограничить количество программ, которые могли бы поместиться.

Использование пространства подкачки для резервного копирования памяти дает пользователю иллюзию, что есть больше оперативной памяти, чем есть на самом деле. Это называется виртуальной памяти. Рисунок 4.1 демонстрирует, виртуальную память с двумя процессами (A и B), каждый из которых в настоящее время страницы загружаются в основную память, а остальные процессы хранятся в пространстве подкачки не сопоставленные страницы ссылаются на страницы, которые еще не были загружены в память. Выгрузка страниц, которые в свое время были в памяти, но были удалены, чтобы вмещать новые страницы. Когда процессор генерирует адреса памяти (например, для извлечения инструкции или опорную точку), этот адрес должен быть переведен с его логическим (или виртуальным) адресом в адрес постоянного. Для того, чтобы выполнить преобразование, операционная система поддерживает таблицу страниц для каждого процесса. Эта таблица означает для каждой страницы процесса, если он находится в данный момент в памяти и если да, то где. Рисунок 4.2 показывает, что таблицы страниц будет выглядеть для двух процессов на рисунке 4.1. Процесс A имеет три страницы, загруженные в память в тех местах, 0, 1, и 3 (эти места называются кадрами) и процесс B имеет одну страницу, загруженной в память на месте 2. Допустимый столбец бит, указывающий, если страница находится в настоящее время в памяти (действительный бит установлен в 1) или нет.

Если ссылка страница не в настоящее время в памяти, ошибка страницы генерируется, что вызывает прерывание. Операционная система вовлекается в данный момент для выполнения замены. Во-первых, операционная система находит доступный кадр для использования в памяти. Если нет доступных кадров, то операционная система должна выбрать страницу отбрасывать. Если страница была изменена, то он сначала должен быть сохранен обратно, чтобы поменять пространство. После того, как кадр был доступен, ОС загружает новую страницу из пространства подкачки в этом кадре в памяти. ОС изменяет таблицу страниц, чтобы указать местоположение новой страницы (и, если страница была удалена из памяти). И, наконец, операционная система

позволяет процессору возобновления текущего процесса.
Поскольку пространство подкачки хранится на жестком диске, любой подкачки (обмен) включает в себя доступ к жесткому диску. Поскольку время ответа жесткого диска гораздо медленнее, чем время отклика памяти, любая перестановка приведет к замедлению выполнения программы. Таким образом, подкачка следует избегать, насколько это возможно. Если страница должна была быть написана еще раз, чтобы поменять место, во-первых, это увеличивает время подкачки еще больше.

Process A page table

Page	Frame	Valid
0	1	T
1	–	F
2	–	F
3	3	T
4	0	T

Process B page table

Page	Frame	Valid
0	–	F
1	2	T
2	–	F
3	–	F
4	–	F
5	–	F
6	–	F

Рисунок 4.2
Пример таблицы страниц для процессов на рисунке 4.1.

Перевод:
Process Apagetable – Таблица страницы процесса А
Page – страница
Frame – рама
Valid – действительно
Process Bpagetable – Таблица страницы процесса Б

Другим фактором, требующим управления памятью является то, что процессы генерируют адреса для данных доступов (загрузка и хранение). Что это такое, чтобы предотвратить процесс от генерации адреса секции памяти, которая не содержит данные этого процесса? Ситуация, когда процесс генерирует адрес другого процесса "пространства памяти называется нарушение памяти. Это может происходить за счет использования указателей, например, в языках программирования, таких как C и C ++. Нарушение памяти должно привести к прекращению процесса, хотя в C ++, программист может также справиться с этой ситуацией с помощью обработчика исключений, который представляет собой обработчик прерывания, написанный программистом и включены в текущую программу.

Управление ресурсами

Помимо памяти и процессора, существует много других ресурсов, доступных в том числе файловой системы, доступ к сети, а также использование других устройств. Операционная система поддерживает таблицу всех активных процессов и ресурсов, что каждый процесс в настоящее время использует или хочет использовать. Большинство ресурсов могут быть доступны только в взаимоисключающим образом. То есть, как только процесс начинает использовать ресурс, никакой другой процесс не может использовать ресурс, пока первый процесс не освободит его. После освобождения, операционная система может решить, какой процесс, следующий получает доступ к ресурсу.
Есть много причин для взаимного исключения, но рассмотреть эту ситуацию. Файл сохраняет свой баланс текущего счета. Предположим, что в настоящее время $ 1000. Теперь представьте, что вы и доверенное вами лицо пришли в банк, стоите в очереди, и каждому из вас помогают разные кассиры одновременно. Вы оба просят выдать $ 1000. В то же время, оба кассиры получить доступ к данным и находят доступные $ 1000. Грустный, что вы закрываете счет, они оба ведут ваше дело, снова в одно и то же время. Оба компьютера сообщают что "доступ к тому же общий файл диска, один, хранящий информацию об учетной записи. Одновременно обе программы чтения значения ($ 1000), определить, что он больше или равен сумме, которую вы запрашиваете, и сбросить значение до $ 0. Вы и другой человек запрашиваете $ 1000, а остаток был сокращен до $ 0. Несмотря на то, есть только $ 1000, чтобы начать, теперь у вас есть $ 2000! Это хорошая новость для вас, но очень плохая новость

для банка. Для предотвращения подобной ситуации не произошло, мы проводим Несовместные доступ к любому общему ресурсу.
Доступ к любой общей опорной точки должны быть синхронизированы, является ли точка привязки совместно через компьютерная сеть или распределяется между двумя процессами, работающими одновременно на одном компьютере. Если один процесс начинает доступ к опорной точки, никакой другой процесс не должен быть разрешен доступ к нему, пока первый процесс не завершит доступа и освобождает ресурс. Операционная система должна обрабатывать межпроцессную синхронизацию для обеспечения взаимоисключающие доступа к ресурсам. Возвращаясь к нашему примеру кассира банка, синхронизация с совместно используемой учетной записи проверки нулевой точки будет работать, как это. Операционная система будет получать два запроса для доступа к общему файлу. Операционная система будет выбрать один из двух процессов для предоставления доступа. Другой будет вынужден ждать. Таким образом, в то время как один кассир может выполнять операцию базы данных для выдачи $ 1000, а другой придется подождать. После того, как первая кассир закончит, ресурс освобождается, а второй кассир теперь может получить доступ к файлу. К сожалению, второй кассир найдет баланс, который составляет $ 0, а не $ 1000, и не раздают никаких денег. Таким образом, вы и ваше доверенное лицо сможете уйти только с $ 1000.

Помимо отслеживания открытых ресурсов, операционная система также обрабатывает тупиковую. Простая версия тупиковой ситуации показана на рисунке 4.3. Операционная система многозадачности между процессами P0 и P1, и есть (по крайней мере) два имеющихся ресурсов, R0 и R1.

P0 работает некоторое время, а затем запрашивает доступ к R0.

Операционная система определяет, что никакой другой процесс не использует R0, поэтому предоставляет запрос.

P0 продолжается, пока таймер не истечет, и операционная система переходит на P1.

P1 работает на некоторое время, а затем запрашивает доступ к R1.

Операционная система определяет, что ни один другой процесс не использует R1, так что разрешает подать запрос.
P1 продолжается, пока таймер не истечет и ОС переключается на P0.

P0 продолжает использовать R0, но теперь также просит R1.

Операционная система определяет, что R1 представляет собой, в данный момент использует P1 и перемещает P0 в очередь ожидания, и переключается на P1.

P1 продолжает использовать R1, но теперь также просит R0.

Операционная система определяет, что R0 в настоящее время используется P0 и перемещает P1 в очередь ожидания.
Операционная система готова к выполнению процесса, но и запущенных процессов, P0 и P1, находятся в ожидании очереди. Кроме того, ни P0, ни P1 может начать не работать снова, пока процесс в ожидании (R1 и R0, соответственно) становится доступным. Но так как каждый процесс находится в ожидании, и у всех другие потребности, и каждый ждет, никогда не будет времени, когда ресурс будет доступным, какому либо из процессов, чтобы начать снова, и таким образом задача тупиковая.

Для того, чтобы иметь дело с тупиковыми ситуациями, необходимы некоторые проверки операционных систем, чтобы увидеть, есть ли

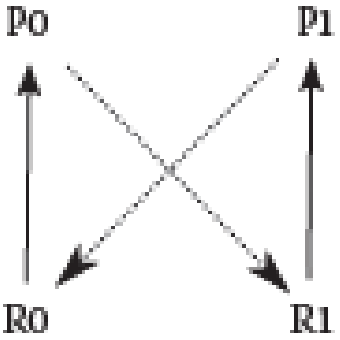


Рисунок 4.3.
Тупиковая ситуация (Сплошные линии указывают на предоставленные ресурсы. Пунктирные линии указывают на запрашиваемые ресурсы)

взаимоблокировка, которая может возникнуть перед предоставлением какой-либо запроса. Это, как правило, перестраховки и не используется большинством операционных систем. Другие операционные системы потратят время каждый раз, чтобы увидеть, если возникла тупиковая ситуация. Если да, то один или нескольких зашедших в тупик процессов произвольно удалят, освобождая свои ресурсы и позволяя другому процессу (ам), чтобы продолжить. Процессы будут перезапущены в любое другое время в будущем.

Существуют и другие операционные системы, которые не справляются с тупиковыми ситуациями вообще. Это дает право пользователю решить, если возникла тупиковая ситуация и принять надлежащие меры!

Управление файловой системы

Основная задача раннего ОС была предложить пользователям возможность доступа и управления файловой системой (имя MS-DOS означает Microsoft Disk Operating System, а команды почти полностью касаются файловой системы на жестком диске). Типичные команды в управлении файлами должны открыть файл, переместить файл, переименовать файл, удалить файл, распечатать файл, а также создавать, перемещать, переименовывать и удалять каталоги (папки). Сегодня эти возможности можно найти в программах управления файлами , таких как Windows Explorer. Хотя эти команды могут быть выполнены путем перетаскивания, щелкнув и т.д., они также доступны из командной строки. В Unix / Linux, команды файловой системы включают в себя компакт-диск, LS, мв, ПРС, P.M., MkDir, RmDir и Lp. В DOS, команды файловой системы включают в себя компакт-диск, перемещать, копировать, Del, MkDir, RmDir и печать . Есть широкий спектр дополнительных команд Unix / Linux, которые позволяют системному администратору управлять файловой системой, в целом, путем установки новых устройств или перемещения , где эти устройства будут доступны в файловом пространстве, а также проверить целостность файлов (например, после того, как файл остается открытым, и система перезагружается).
За кулисами, операционная система управляет доступом к файлам. Когда пользователь отправляет команду, чтобы переместить файл или открыть файл, будь то непосредственно из командной строки или из некоторого программного обеспечения, пользователь в действительности подает запрос. Операционная система в настоящее время принимает . Во-

первых, операционная система должна гарантировать, что пользователь имеет права доступа для запрашиваемого файла. Во-вторых, операционная система должна найти файл. Хотя пользователи будут указывать местоположение файлов, эти места будет необходимо найти в файловой системе. Такая спецификация будет включать в себя раздел диска и каталог файла (например, C: \ Users \ Foxr \ Мои документы \ CIT 130). Однако, как будет обсуждаться в главе 5, файлы делятся на более мелкие части . Операционная система должна определить часть файла, желательно, и его физическое местоположение. Операционная система должна отображать логическое местоположение в физическое местоположение, которое состоит из поверхности диска и места на поверхности. Наконец, операционная система должна инициировать связь с диском, чтобы выполнить требуемую операцию.

Защита и безопасность

Большинство компьютеров в эти дни являются частью более крупной сети компьютеров, где могут быть совместно используемые файлы, доступ к сети, или какой-либо другой совместно используемый ресурс. Кроме того, несколько пользователей могут совместно использовать один и тот же компьютер. Файлы, хранящиеся как локально, так и через сеть должны быть защищены таким образом, чтобы пользователь не смог случайно (или злонамеренно) перезаписывать, изменять, или неправильно использовать файлы другого пользователя. Защита гарантирует, что пользователь не злоупотребляет системой, не использует чужие файлы, не злоупотребляя системными ресурсами и т.д. Безопасность обеспечивает защиту всей сети.

Общий механизм защиты и безопасности является обеспечение учетных записей пользователей. Каждый пользователь будет иметь имя пользователя и механизм аутентификации (чаще всего, пароль). После того, как операционная система установила, кем является пользователь, пользователь затем может получить доступ к ресурсам и файлам, к которым пользователь имеет право на доступ. Они будут включать в себя общие файлы (файлы, которые указали другие пользователи являются доступны) и свои собственные файлы. Unix и Linux используют довольно упрощенный подход путем размещения каждого пользователя в группу. Файлы можно прочитать, записать и выполнять , а также доступностью файла можно управлять, чтобы обеспечить различными правами владельца файла, группу файла, и остальной мир . Например,

вы можете создать файл, который может считываться/редактироваться/ исполняться вами , считываться /исполняться кем-либо в вашей группе, и исполняться миром . Обсудим учетные записи пользователей, групп и методы контроля доступа в главе 6.

ФОРМЫ УПРАВЛЕНИЯ ПРОЦЕССОМ

Различные способы, в которых ОС будет выполнять программы
1. Одиночное задание
2. Пакетный
3. Мультипрограммирование
4. Многозадачность
5. Многопоточный
6. Мультипроцессорный

Система одиночного задания, самая старая и самая простая, просто выполняет одну программу, пока она не завершается , а затем переключается обратно в ОС и дает возможность пользователю запросить другую программу для выполнения. MS-DOS был как ОС. Все программы были выполнены в порядке, в котором они были запрошены. Если запущенный процесс требовал внимания, такие как затянутая операция ввода или вывода , процессор будет на холостом ходу, пока пользователь или устройство ввода / вывода не ответит. Это очень неэффективно. Это также может быть неприятно для пользователей, которые хотят сделать больше чем одну задачу за один раз. Помимо ранних компьютеров, самые ранние мэйнфреймы были также с одиночным заданием .
Система пакетной обработки аналогична одной системе постановки задач, за исключением, что существует очередь (линия ожидания) для процессов. Главное отличие здесь в том, что пакетная обработка системы имеет более одного пользователя. Таким образом, пользователи могут представлять свои программы для исполнения в любое время. При подаче запроса на программу, процесс добавляется в очередь. Когда процессор заканчивает с одним процессом, ОС вызывается, чтобы решить, какой из ожидающих процессов, довести до ЦП следующим . Некоторым формам планирования необходимо решить, какой процесс выбирается следующим . Алгоритмы планирования включают в себя схему приоритетов, кратчайшую работу сначала, и первое задание первым и выполняется. Для схемы приоритета процессам дается приоритет

в зависимости от статуса пользователя (например, администрация , преподавательский состав , аспирант, студент). На основе алгоритма планирования, нагрузка на машину, и где процесс помещается в очередь, вы могли заметить, что для этого потребуется несколько минут, часов или даже дней, прежде чем ваша программа будет выполнена! Для выполнения первоочередных работ, ОС пытается оценить количество процессорного времени, каждый процесс будет требовать, и выбирать процесс, который займет наименьшее количество времени. Статистически, это держит среднее время ожидания до минимума, хотя это может показаться несправедливым по отношению к пользователям, которые имеют процессы отнимающих много времени. *Пришел первый, обслужен первым* - является традиционной схемой для любой очереди, будь то в компьютере или в банке. Эта схема, известная также как FIFO (первый вошел, первый ушел), является справедливым, но не обязательно эффективным .

Другое различие между единой системой постановки задач и системы пакетной обработки данных является то, что система пакетной обработки данных не имеет интерактивности с пользователем. В одной постановки задач, если I / O требуется, процесс приостанавливается, пока компьютер ждет ввода или вывода для завершения. Но в пакетной обработке данных, все входные данные должны быть представлены в тот момент, когда пользователь отправляет процесс. Поскольку пакетная обработка наиболее часто используется в течение первых трех поколений компьютеров, как правило, вход был введен перфокартой или хранился на магнитной ленте . Кроме того, все выходные данные будут обрабатываться в "автономном режиме", будучи отправленными на магнитную ленту или принтер. Очевидно, что без интерактивности, много типов процессов будет функционировать совсем иначе, чем мы привыкли. Например, компьютерная игра потребует, чтобы все ходы пользователя были завершены до начала игры, а текстовый процессор будет требовать, чтобы все редактирования и изменения форматирования были сохранены перед запуском текстового процессора. Поэтому, пакетная обработка данных имеет свои ограничения. Вместо этого, пакетная обработка широко используется для таких задач, как вычисление заработной платы или налогов, или делать математические вычисления.

Обратите внимание на то, что в пакетной обработке, как в одной постановки задач, только один процесс выполняется, включая операционную систему. Операционная система не будет реагировать, когда новый процесс был запрошен пользователем. Что же произойдет,

когда пользователь отправляет новый запрос на выполнение процесса? Отдельная система периодического массового обслуживания была ответственна за прием новых заданий пользователя и добавление их в соответствующую очередь.

Хотя пакетная обработка была распространена на компьютерах первого по третей генерации, работающие на мэйнфреймах и мини-компьютерах, вы все еще можете найти некоторые партии обработки сегодня. Например, в Unix и Linux, существуют планирования команд, таких как хрон и на, так и в Windows, программа планировщик заданий доступна. Программы по расписанию будут работать бесперебойно в том, что они работают без вмешательства пользователя, хотя они могут работать в многозадачном режиме.

В системе пакетной обработки данных, вход становится доступным, когда процесс начинается с выполнения через какой-либо источник , такие как перфокарты или файл на магнитной ленте или диске. Одна система постановки задач, с другой стороны, могла бы получить ввод напрямую от пользователя с помощью клавиатуры или другого устройства. Включение пользователя (человека) в процесс значительно замедляет обработку. Почему ? Потому что человек настолько медленен , при поступлении информации, чем диск, ленточный накопитель или считыватель перфокарт (даже если читатель перфокарты очень медленный). Кроме того, ожидание на выходе может замедлить обработку, даже если выход обрабатывается в автономном режиме. Скорость магнитной ленты, диска или принтера намного медленнее, чем у центрального процессора. Затем единая система постановки задач имеет значительную неэффективность - вход и выход замедляет обработку. Это также относится к пакетной обработки, даже если пакетная обработка имеет не столь значительное влияние, из-за того что нет ни одного человека в цикле проработки.

Система мультипрограммирования аналогична системе периодической загрузки за исключением того, что, если текущий процесс требует ввода / вывода, то этот процесс перемещается в другую очередь (очередь с ожиданием ввода / вывода), а операционная система выбирает другой процесс для выполнения. Когда первоначальный процесс завершается с его вводом / выводом , оно возобновляется, и процесс замены перемещается обратно в очередь. Таким образом, длительные ввода / вывода не вызывает процессор оставаться в режиме ожидания, и поэтому система является гораздо более эффективной . Идея - процессор выполняет операции ввода / вывода, называется кооперативной

многозадачностью . Мы вернемся к этой идее ниже.
Есть несколько различных применений для очередей в многозадачности, поэтому нам необходимо провести различие между ними. Существует готовая очередь (очередь процессов, ожидающих ЦП), очередей ввода / вывода (по одному для каждого устройства ввода / вывода), и очередь ожидания (очередь процессов, которые были запрошены для запуска, но еще не были перемещены в готовую очередь). Причина, по которой существует очередь ожидания в том, что те процессы в готовой очереди уже загружаются в памяти . Поскольку память ограничена в размерах, могут быть процессы, запрашиваемые пользователями, которые не могут поместиться, и поэтому они сидят в очереди ожидания, пока не освободиться место в готовой очереди. Там будет место в готовой очереди, если процесс заканчивается, и выходит или если многие процессы были перемещены из готовой очереди к очереди ввода / вывода.
Система мультипрограммирования требует дополнительного механизма, известного как контекстный переключатель . Переключение контекста является лишь переход ЦП от одного процесса к другому. Мы изучали этот вопрос кратко ранее. Давайте поближе рассмотрим . Для переключения процессов, центральный процессор должен сначала сохранить текущий процесс "состояния и получения состояния следующего процесса". Рисунок 4.4 предоставляет снимок компьютера, чтобы проиллюстрировать переключение контекста. На этом этапе процесс P3 выполняется процессором. Реестр PC хранит адрес в памяти следующей команды P3, ИК сохраняет текущую инструкцию P3, СП сохраняет вершину стека во время выполнения P3 в памяти, флаги сохраняют статус последней выполненной команды, и регистры данных хранят соответствующие данные для P3. Переключатель контекста требует, чтобы эти значения сохранялись в памяти. Затем, P3 перемещается в памяти соответствующей очереди ввода / вывода, перемещая процесс P7 к передней части готовой очереди. Наконец, процессор должен восстановить статус P7, идя в памяти и извлекая сохраненные значения регистров для P7. Теперь компьютер будет хранить адреса в памяти следующей инструкции P7, ИК сохранит текущую инструкции P7, СП будет хранить в верхней части стека во время выполнения P7, флаги сохранят статус последней выполненной инструкции P7, и регистры данных будут хранить последние данные, используемую P7. ЦП возобновляет выполнение выборки, но теперь на P7, а не P3. Это продолжается до тех пор, пока либо P3 готов возобновить или P7

требует ввода / вывода или завершает свою работу. В первом случае, переключение контекста происходит, в котором P3 восстанавливается и P7 попятился в очереди, а в последнем случае процессор переключается с P7 на P0.
Поскольку переключение контекста требует сохранения и восстановления контекста двух процессов, оно не происходит мгновенно, а требует несколько больших затрат времени. Процессор вхолостую во время переключения.

Переключение контекста требует несколько, возможно от десятков до двух десятков, операций с памятью. Это медленно по сравнению со скоростью процессора. Если переключение контекста может хранить и восстанавливать из кэша, тем лучше. Тем не менее, некоторые высокопроизводительные компьютеры используют дополнительные наборы регистров для переключения контекста. Это обеспечивает быстрый отклик и, таким образом, сохраняет время простоя процессора во время наименьшего количества времени. Тем не менее, несмотря на время простоя процессора, вызванного переключением контекста, с помощью переключателя контекста в мультипрограммировании до сих пор гораздо более эффективно , чем позволить простою процессора во время операции ввода или вывода.
Многозадачность воспринимает идею мультипрограммирования на один шаг вперед. В мультипрограммировании, процесс только приостановлен

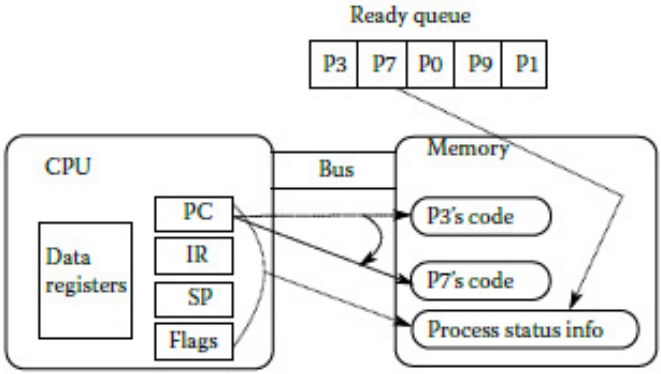


Рисунок 4.4
Переключение контекста требует изменения значений регистра.

(вынужден ждать), когда он нуждается в операции ввода / вывода, тем самым вызывая ЦП , чтобы переключиться на другой процесс. Но, в многозадачном режиме, таймер используется для подсчета количества тактов, прошедших с момента последнего переключения контекста начала этого процесса. Таймер устанавливается на показатель некоторого начального значения (скажем, 10000). С каждым новым заданием тактовый цикл, таймер уменьшается. После того, как она достигнет 0, таймер прерывает процессор, чтобы заставить его переключиться на другой процесс. Рисунок 4.5, который представляет собой вариант рисунку 4.4, показывает переключение контекста как вынужденную таймером. Процесс подвешивания получается перемещенным в конце готовой очереди и должен ждать своей очереди. Это может звучать как огромный провал процесса, но на самом деле, так как современные процессоры очень быстрые, приостановленный процесс ожидает лишь несколько миллисекунд, самое большее, прежде чем он переместится обратно в ЦП . В многозадачных системах, люди даже не заметят , что процесс был приостановлен, а затем возобновлен , потому что время человеческого отклика больше, чем миллисекунды. Поэтому, хотя компьютер, выполняет несколько программ одновременно с многозадачностью, правда заключается в том, что компьютером выполняются процессы в параллельном, или совмещающем друг друга режиме.

В многозадачности, мы видим два механизма, с помощью которого процессор будет переходить от одного процесса к другому: поскольку

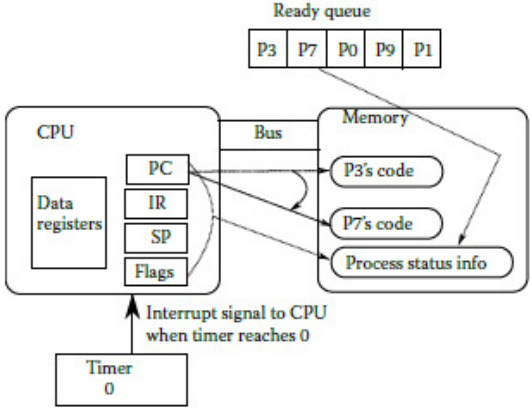


Рисунок 4.5
Переключение контекста в многозадачности.

текущий процесс запрашивает некоторую форму ввода / вывода или потому, что таймер истек на этом процессе . Многозадачность более правильно считается кооперативной многозадачностью , когда процесс добровольно отказывается от процессора. Это происходит в мультипрограммировании когда процесс запрашивает В /В .

В случае, когда таймер переключает контекст, эта форма многозадачности называется конкурентоспособной многозадачностью или вытесняющей многозадачностью . В этом случае процесс вынужден отказаться от процессора. Есть и другие причины, по которым процесс может перенести себя в другую очередь или обратно в конец готовой очереди. Одной из таких причин является вмешиваться в процесс пользователем, который перемещает процесс от переднего плана на задний . Мы используем термин *переднего плана* для обозначения процессов, которые либо немедленно доступны для ввода данных пользователем или в настоящее время часть готовой очереди. Фоновые процессы "сидят на заднем плане". Это означает, что пользователь не взаимодействует с ними, или в настоящее время ожидает, пока у них есть еще один шанс на получение доступа к процессору. В Windows, например, на переднем плане процесс является одной вкладкой, которая была выбрана, один "поверх" других окон , так как они появляются на панеле задач. В Linux то же самое относится и к ГИП процессам . Из командной строки, процесс находится на переднем плане, если он не был выполнен с использованием командной строки &. процесс находится на переднем плане, если он не был выполнен с использованием & команды, Это рассматривается более подробно в процессе выполнения в главе 11.

Еще одна причина, по которой процесс может добровольно отказаться от ЦП это длительность ожидания на завершение. Завершение происходит, когда один процесс ожидает некоторую информацию (вывод , общую нулевую точку) из другого процесса. Еще одна причина для процесса сдать ЦП является то, что процесс имеет низкий приоритет и другие, более высокие процессы приоритета ждут.

Первоначальная идея многозадачности называлась разделение по времени. В режиме разделения времени процесс раздавался процессором самостоятельно. Только позже, когда ОС приступили к реализации конкурентной многозадачности, что совместное использование термина было занижено. Разделения времени впервые был реализовано на мэйнфреймах в 1957 году, но не регулярно использовалось до третьего

поколения. К четвертому поколению, это было обычным явлением в мэйнфреймах , но не в личных компьютерах до 1980-х. В самом деле, ПК с Windows ОС до того Windows 95 и Mac OS, до OS X выполнена кооперативно , но не выдерживает конкуренции, многозадачности.
Многозадачность использует процедуру циклического планирования. Это означает, что все процессы в очереди дают свое собственное время с ЦП . ЦП переходит от одного процесса к следующему, как истечет время. Таймер затем сбрасывается. Несмотря на то, что это очень справедливый подход, пользователь может захотеть один конкретный процесс, чтобы иметь больше внимания процессора, чем другие. Для того, чтобы это сделать, пользователь может указать приоритет для любого или каждого процесса. Как правило, все процессы начинаются с тем же приоритетом. Мы рассмотрим настройки и изменения приоритетов процесса в главе 11.
Сегодня процессы могут содержать несколько ходовых частей, называемых потоками. Потоки разделяют одни и те же ресурсы.

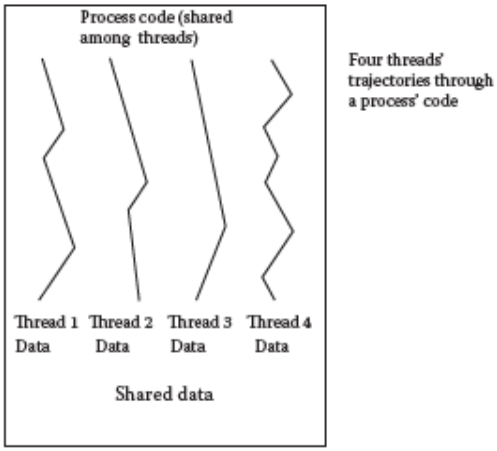


Рисунок 4.6
Нити процесса.

Four threads trajectories through a process code - Четыре траектории нитей через процесс кода
Process code (shared among threads) – Процесс кода (разделенный между нитей)
Thread – нить
Data – данные
Shared data – разделенные данный

Совместно используемые ресурсы являются типичным совместным программным кодом хранящийся в памяти, хотя они также могут обмениваться данными, значениями регистров и статусом . Чем отличается один поток от другого в том, что они также будут иметь свои собственные данные. Например, вы можете запустить веб-браузер Firefox с несколькими открытыми вкладками или окнами. Вы используете один и тот процесс (Firefox), где каждая вкладка или окно имеют свой собственный поток. Единственное отличие между каждой вкладкой или окном является его данные. Поскольку потоки имеют ту же программу, ОС работает один процесс, но несколько потоков. Многозадачная система, которая может выключить между процессами и между потоками одного и того же процесса, известного как многопоточной ОС. См. Рисунок 4.6, который иллюстрирует процесс четырех потоков. Каждый поток вносит свой путь через код, и хотя эти четыре потока разделяют некоторые данные, каждый из них имеет свои собственные отдельные данные.

Потоки служат другим целям в области программирования. С помощью потоков в процессе, они могут контролировать их наличность и другими потоками. Рассмотрим процесс, который содержит два потока. Один поток создает значения для буфера памяти. Другой потребляет значения из буфера. В многопоточности, процессор выключается между потоками. Тем не менее, поток потребителя не может продолжаться, если буфер пуст. Это происходит потому, что нет ничего, чтобы потреблять. Таким образом, потребительная нить может заставить себя ждать, пока производитель потока не укажет , что новая точка привязки стала доступной. С другой стороны, производитель потока может заставить себя ждать, если буфер не заполнен, до тех пор, пока потребительские сигналы не будут запущены, которые нужны для его потребления. Два потока могут "встретиться" на задании. Заставив себя ждать, потоки в этом примере используют кооперативную многозадачность (многопоточность) в том, что они добровольно отказываются от процессора, а не вынуждены отказаться от процессора, когда таймер истечет.

Переход на один шаг дальше, важно, чтобы два потока или процессы не пытались получить доступ к общей опорной точке в одно и то же время. Рассмотрим, что может произойти, если поток 1 пытается получить доступ к общему буферу и поток прерывается при переключении контекста. Он уже скопирован следующим элементом буфера, но не было возможности сбросить буфер, чтобы указать, что конкретный элемент в настоящее время "потребляли". Теперь, поток 2 возобновляется, и он пытается поместить опорную точку в буфере только чтобы обнаружить,

что буфер полон. Он на самом деле не полный, потому что один элемент был израсходован. Таким образом, доступ к буферу потоком 2 вызывает поток ждать, когда он не должен. Другая ситуация, когда два потока одни и те же точки привязки. Напомним, наш банковский пример из ранней главы , мотивируя использование взаимного исключительного доступа. Для потоков, опять-таки, доступ к общей данности должен быть взаимоисключающим . Теперь мы рассмотрим тот же пример с использованием потоков. Тема 1 считывает баланс проверки счета в размере $ 1000. Поток 1 готов вычесть $ 1000 из него, но прерывается таймером. Тема 2 считывает баланс проверки счета, $ 1000, вычитывает из него $ 1000 и сохраняет новое значение обратно, $ 0. Переключиваясь обратно на тему 1, она имеет устарелое значение, $ 1000.

Он вычитывает $ 1000, сохраняет $ 0. Результатом является то, что значение устанавливается равным $ 0, но две разные нити вычитали $ 1000. Это позволило бы привести к повреждению данных.

Для обеспечения надлежащего доступа к общим данным, операционная система должна поддерживать правильную синхронизацию. Синхронизация требует, чтобы процесс или поток не прерывался при доступе к опорной точке , или что точка привязки не доступна, как только один процесс или поток оказался доступным для него, пока он не был освобожден. Мультипрограммирование, многозадачность и многопоточность требует синхронизации.

Преимущество многопоточного ОС над многозадачной ОС является то, что контекстный переключатель между потоками происходит быстрее и эффективнее, чем переключение между процессами. Причиной этого является то, что переключение контекста требует меньшего количества операций для сохранения текущего потока и восстановления следующего потока, чем это делать для сохранения и восстановления процессов. Представьте себе, ссылаясь обратно к рисунку 4.5, код этого процесса P7 была перемещен из памяти и обратно к виртуальной памяти (подкачки). Переключение контекста между P3 и P7 будет несколько катастрофическим, поскольку операционная система будет иметь не только статус сохранить P3 и восстановить статус P7, но и придется загружать несколько страниц P7 с диска в память. Этого не произойдет при переключении между потоками. Предположим, что P3 и P7 нити одного и того же процесса. Переключение контекста между ними требует гораздо меньше усилий, и поэтому гораздо быстрее.

Большинство компьютеров, вплоть до недавнего времени не были

известны как однопроцессорные системы . То есть, они имели только один процессор. С одним процессором, только один процесс или поток может быть активно выполнен в любое время. В мультипрограммировании , многозадачности, и многопоточности , несколько процессов (или потоков) могут быть активными , но процессор будет выполнять только один в любой данный момент времени. Таким образом, параллельное выполнение было возможно с помощью любого из этих способов выполнения процесса, но не соответствует действительности параллельной обработки. В многопроцессорной обработке , компьютер имеет более одного процессора. Если у компьютера , например, было два процессора, он может выполнить два процесса или потока одновременно, по одному на каждый процессор. Сегодня большинство компьютеров производятся с несколькими процессорами.

Более традиционная форма многопроцессорной системы была одна, которая содержала более одного ИС процессора. Такие компьютеры были дорогими и несколько персональных компьютеров были мультипроцессорные системы. Тем не менее, с дальнейшей миниатюризацией которая была представлена за последнее десятилетие, достаточно мест было предоставлено в одном процессоре для размещения нескольких ядер. Таким образом, современные процессоры называются двойными или четырех ядерными процессорами. Каждое ядро, по сути, процессор; оно имеет свою собственную ALU (арифметика / логическое устройство), блок управления, регистры и кэш. Единственное, что ее разделяет с другими процессорами на чипе, являются булавки, которые крепят весь набор ядер к системной шине компьютера.

Параллельная обработка представляет собой возможность и вызов. Возможность заключается в выполнении нескольких процессов одновременно. Например, одно ядро может выполнять процесс веб-браузера, в то время как другой может выполнить программу развлечений, такие как видео или mp3-плеер. Проблема, однако, состоит в обеспечении того что каждое ядро используется в максимально возможной степени. Если пользователь только работает один процесс, может, что процесс каким-то образом быть распределенным на несколько ядер таким образом, что каждое ядро выполняет отдельную часть процесса? Это требует иного мышления при написании программного кода. Программист может попытаться отделить логику программы от компьютерной графики подпрограмм. Таким образом, одно ядро может выполнять логику и другое ядро компьютерной графики подпрограмм. Идея распределения обработки на несколько процессоров называется

параллельным программированием, и она остается одной из больших проблем, открытых для программистов сегодня.

В этом разделе рассматриваются и сравниваются некоторые подходы к процессу управления. Представьте себе, что вы хотите запустить три программы, называемые p1, p2, p3. Мы увидим, как они работают на одной системе, постановке задач пакетной системы, мультипрограммировании системы и многозадачной системе .

В одной постановке задач, p1 выполняется до тех пор, пока не будет завершена, а затем p2 запускается, а затем p3 запускается. Это не очень приятно для пользователя, потому что вы не в состоянии сделать несколько вещей одновременно (например, вы могли бы редактировать документ, поиск в Интернете для некоторых рекомендаций, и отвечать по электронной почте, вы не делаете их одновременно, вы не выполняете все задачи одновременно). Вы не можете копировать и вставлять из одной программы в другую либо, потому что одна программа завершается до начала следующего.

В пакетной обработке , выполнение трех процессов практически идентично одной постановки задач. Есть два изменения. Во-первых, так как пакетная обработка любого отказывается от взаимодействия с пользователем, все входные данные должны поставляться с программой и любой выход осуществляется только после выполнения. Во-вторых, отдельная система обработки может использоваться для планирования трех процессов , так как они добавляются в очередь ожидания. Планирование может быть первый пришел - первый обслужен, приоритет, самая короткая работа первой, или какой-либо другой формы .

В мультипрограммировании , компьютер будет работать до тех пор, пока p1 p1 не будет прекращен или p1, необходимый для выполнения ввода / вывода. В этот момент, операционная система будет двигаться p1 в очередь ожидания, и начинать p2. Теперь p2 будет работать до тех пор, пока потребности p2 I / O, p2 не прекратятся , или p1 готов возобновить. Если p2 требует ввода / вывода или прекращается, то операционная система начнет p3. Если p1 будет готов, p1 возобновляется и p2 вынужден ждать. В то время как в мультипрограммировании , ввод и вывод может быть сделан по требованию, в отличие от партии, мультипрограммирование не может дать пользователю внешний вид взаимодействия с компьютером, так как процессы, вынуждены ждать своей очереди с центральным процессором и устройствами ввода / вывода.

В многозадачности, мы действительно получаем перекрывающее

выполнение, так что пользователь не может сказать, что процессор переключается между процессами. Для многозадачности, процесс работает следующим образом:

• нагрузки p1, p2, p3 в память (или, как большая часть каждого процесса по мере необходимости)
• Повторение
• Пуск (или возобновить) p1 и установить таймер на системно заданную величину (скажем, 10000 циклов)
• Декремент таймер после каждого машинного цикла
• Когда таймер достигает 0, вызовите ОС
• Операционная система выполняет переключение контекста между p1 и p2
- Переключение контекста требует сохранения статуса П1 и значения регистров и восстановления состояния Р2 и значения регистров, возможно, также обновление памяти по мере необходимости
• Повторите с p2
• Повторите с p3
• До тех пор, пока все процессы не прекращаются
Если какой-либо из p1, p2, p3 были нити одного и того же процесса, то многопоточность будет почти идентичен многозадачности за исключением того, что переключение контекста будет тратить меньше времени.

Буттинг и инициализация системы

Основная память непостоянства , то есть, она требует постоянной подачи питания, чтобы сохранить его содержимое. Выключите питание, и основная память станет пустой (все его содержимое становится 0s). После отключения компьютера, память пуста. В следующий раз, когда вы включаете компьютер, память становится пустой . Это важно, потому что, для того, чтобы использовать компьютер, вам нужно иметь ОС загружающийся в память и работающий . Операционная система — это программа, которая позволит загружать и запускать другие программы. Это создает парадоксальную ситуацию: как мы можем получить загрузки операционной системы и работать , когда эта операционная система, которая заботится о загрузке и выполнении программ для нас и ОС не в памяти, когда мы сначала включаем компьютер? Нам нужен специальный, одноразовый процесс, называемый загрузка . *

Процесс загрузки работает следующим образом:

1. Процессор инициализирует себя (инициализирует свои регистры) и посылает управляющие сигналы к различным компонентам в компьютерной системе.
2. Базовая система ввода / вывода (BIOS) выполняет включение питания самотестирования (POST), где он проверяет устройства памяти и тесты для ответов. Он также может выполнять такие задачи, как установить системные часы, включить, или отключить различные аппаратные компоненты, а также взаимодействовать с контроллерами дисков.
3. Контроллеров жестких дисков (SCSI, а затем IDE) сигнализируют инициализировать себя, другие устройства тестируются, сетевая карта, USB и т.д.
4. BIOS определяет, где хранится ОС. Обычно это делается путем тестирования каждого из этих устройств в заданном порядке, разыскивая ОС, пока не будет найден: дискеты, CD-ROM, первый диск, второй диск или сеть.
5. Если ваш компьютер имеет несколько операционных систем, то программа загрузчика выполняется, чтобы определить, какую операционную систему для загрузки и запуска. Две программы, используемые для двойной загрузки в Windows, и Linux являются GRUB (Grand Unified Boot Loader) и LILO (Linux Loader).
6. Ядро ОС загружается с диска в память. Ядро представляет собой части операционной системы, которая должна быть в памяти, чтобы использовать компьютер или основные компоненты операционной системы.
7. На данный момент, управление переходит из программы загрузки в ядро операционной системы, которая затем запускает скрипты инициализации для завершения процесса загрузки. Скрипты инициализации именно то, что они звучат, как, сценарии оболочки, которые при запуске, инициализируют различные аспекты ОС.

* Термин загрузка происходит от слова самонастройки, который описывает, как можно вытащить себя своими собственными ремнями, как пожарному , когда он встает посередине ночи в ответ на пожарную тревогу. Сам термин приписывается рассказам барона Мюнхгаузена.
(A) Linux начинается путем запуска программы инициализации. Его основная работа состоит в том, чтобы запустить ОС в определенном

уровне запуска, например, только текст, текст-с-сетью, графикой или с графической нагрузкой сети. После того, как был выбран уровень, программа запускает другие сценарии, такие как сценарии rc.sysinit. Этот сценарий, основанный на уровне запуска, запускает различные услуги. Например, если система запускается на уровне доступности сети, то сетевые службы должны быть запущены.
(Б) В операционной системе Windows, одним из решений является запустить ли ОС в безопасном режиме или в полноэкранном режиме пользователя. Безопасный режим представляет собой диагностический режим, в основном используется для того, чтобы системный администратор мог удалить вредоносное программное обеспечение без программного обеспечения, что делает медном х годов сам по себе или перемещение себя . Еще одна задача инициализации - определить, нужен ли безопасный логин и если это так, то открыть журнал в окне. Одним из последних шагов инициализации - запустить оболочку конкретного пользователя, основываясь на том, кто вошел в систему.

Процесс загрузки представляет собой программу. Тем не менее, он не может постоянно находиться в оперативной памяти, так как, когда компьютер включен, ОЗУ пуст. Так как процесс загрузки не меняется, мы будем хранить его в ПЗУ, которое является энергонезависимым, так как его содержание прочно зафиксировано на месте. Тем не менее, так как ПЗУ, как правило, дорогой , и так как части процесса загрузки должны быть гибкими, мы можем хранить порции программы загрузки, такие как программа загрузчика и части BIOS, на жестком диске.

Учетная запись администратора

Для выполнения системных операций, ориентированных на операции, большинство операционных систем, имеют набор привилегированных инструкций . Например, для создания новых учетных записей пользователей, проверки на наличие предохранительных дыр, и манипулирования системными файлами, необходимо иметь доступ администратора. Большинство операционных систем имеют два типа учетных записей, учетных записей пользователей и учетных записей администраторов. * Учетную запись администратора иногда называют корнем или суперпользователем . Учетная запись администратора поставляется с паролем администратора, что лишь немногие должны знать, чтобы предотвратить случайному пользователю войти в систему

как администратор и делать что-то, что не должно быть сделано.
В операционной системе Windows, учетная запись называется администратор. Чтобы переключиться на администратора, вы должны войти в систему как администратор, введя правильный пароль. В Linux, учетная запись называется корнем. Для того, чтобы изменить к корню, вы используете команду SU (переключить пользователя). Как правило, су используется для перехода от одной учетной записи пользователя к другому, говоря, су имя пользователя. Операционная система затем требует пароль для имени пользователя. Если вы су без имени пользователя, то вы просите, чтобы изменить корень. Потому что вы можете запустить су из командной строки, вы не имеете права выйти из системы. На самом деле, вы можете открыть множество окон, некоторые из которых находятся под контролем, как вы, пользователь, а некоторые как корень.
Учетная запись администратора является владельцем авторских прав на все системное программное обеспечение и некоторые из программных приложений. В Linux -l Ls (длинный список) каталогов, таких как / SBIN, / бен, / USR / бен и / USR / SBIN продемонстрирует, что многие программы и файлы принадлежат корень. Во многих случаях, корень является единственным пользователем, который может получить доступ к ним. В других случаях, корень является единственным

* Есть некоторые операционные системы, которые имеют более двух уровней счетов. В операционной системе MULTICS, например, существует восемь уровней доступа, с каждым более высоким уровнем приобретает все права доступа.
Пользователь, который может писать или изменять их, но другие могут только просматривать их. Например, вы должны быть суперпользователем для запуска useradd и userdel, так что только системный администратор(ы) можно добавлять и удалять учетные записи пользователей. Корневая учетная запись в Linux также свойственно в домашнем направлении тории этого корня не находится в том же разделе, домашние каталоги пользователей. Домашние каталоги пользователей, как правило, в / дома / имя пользователя, в то время как корень в / корень.
Как ИТ-человек, вы должны всегда быть в курсе, когда вы вошли в систему как администратор и когда вышли. В Linux, вы можете различать два способа: посмотрев на строке в командной строке. Корень строки в Linux, как правило, # и быстрое пользователь, как правило, $ (если вы не изменить его). Вы можете также узнать, кто вы в Linux с помощью

команды WHO AM I. Вы можете спросить, почему это важно помнить, кто вы, но вы не хотите выдавать некоторые команды, как корень случайно. Например, если вы являетесь пользователем, и вы хотите удалить все файлы в каталоге, включая подкаталоги, вы можете переключиться на этот каталог и выпуск -rf *. Это означает "удалить все файлы рекурсивно, не спрашивая разрешения". Под "рекурсивно" Удаление файлов, он также удаляет все подкаталоги и их файлы, и подкаталоги. Если вы находитесь в неправильном каталоге, операционная система, вероятно, скажет вам, что он не в состоянии выполнить, потому что вы не являетесь владельцем файлов. Но если вы находитесь в неправильном каталоге, и вы корень, то операционная система выполняет удаление и теперь вы удалили неправильные файлы по ошибке. Вы можете думать, что вы будете иметь никаких проблем не помня, кто вы есть, но на самом деле там будет ситуации, когда вы будете входить в одно окно, как корень, а другой, как самого себя, чтобы изменить настройки операционной системы (как корень) и протестировать эти изменения вне (как обычный пользователь).

Установка ОС

При покупке большинства компьютеров на сегодняшний день, они имеют предустановленной операционной системой. Некоторые пользователи, возможно, пожелают установить другую операционную систему, или из-за таких ситуаций, как компьютерный вирус, удаленных файлов или устаревших операционных системах, пользователь может пожелать переустановить или обновить ОС. Кроме того, добавление ОС не обязательно означает, что текущая операционная система должна быть заменена или удалена.

Пользователь может установить несколько операционных систем, каждый в своем собственном разделе диска, а также использовать некоторые загрузочные программы, такие как GRUB, LILO и BOOTMGR. Когда компьютер первые загружается, четыре этапа инициализации аппаратного обеспечения (смотрите раздел, загрузке и инициализации системы) выполняются. Шаг 5 является выполнение программы загрузчика, которая обеспечивает подсказку для пользователя, чтобы выбрать ОС для загрузки. После того, как выбор сделан, процесс загрузки загружает выбранную операционную систему, которая затем инициализируется, и теперь пользователь может использовать компьютер в выбранной ОС . Чтобы изменить операционные системы, пользователь

должен закрыть текущую операционную систему и перезагрузить, чтобы достигнуть программы начального загрузчика. Загрузка нескольких операционных систем на компьютере может привести к трудностям, особенно при обновлении одной из операционных систем или попытка перераспределить жесткий диск. Другой способ, чтобы иметь доступ к нескольким ОС является использование виртуальных машин. VM, по своей сути, является самодостаточной средой , в которую вы можете установить ОС. Сама виртуальная машина хранится в виде данных на жестком диске до тех пор, пока не будет выполнена . Таким образом, виртуальная машина занимает дисковое пространство. Если ваш компьютер имеет старую процессора, вполне возможно, что выполнение VM значительно замедлить работу компьютера, но с применением современных многоядерных процессоров доступны, виртуальные машины могут эффективно и эффективно работать. Это также позволяет иметь два (или более) ОС открыт и работает в то же время, просто перемещая в и из окна виртуальной машины. С помощью программного обеспечения VM, вы можете создать несколько виртуальных машин и запускать любые или все из них в то же время.
В оставшейся части этого раздела описывается, как установить две операционные системы, Red Hat Linux (Специфично, CentOS 5.5), и для Windows 7. Рекомендуется, что при попытке либо монтаже, что вы делаете это изнутри VM. Есть коммерческие и бесплатные VM программные продукты, доступные, такие как VMWare клиент VMware, в VMWare Player, VMware Server и Vsphere и VirtualBox Sun (или VBox).

Установка Windows

Для установки Windows 7, вы начинаете путем вставки Windows 7 компакт-диск в оптический привод, а затем загрузки компьютера. Как ваш компьютер настроен на загрузку ОС на жестком диске, если ваш компьютер не имеет предварительных ОС, вам придется прерывать нормальный процесс загрузки. Это делается, при загрузке, нажав функциональную клавишу F12.[1] Лучше нажимать клавишу снова и снова, пока не увидите, что ваш компьютер ответит на него. Это приведет вас к параметрам загрузки, который представляет собой список различных

1 Фактическая функциональная клавиша может отличаться в зависимости от конкретного производителя. Инструкции появляются во время загрузки системы, чтобы сказать, какая функциональная клавиша(ы) для использования.

устройств, которые могут загружаться на Ваш выбор, как правило, жесткий диск, оптический диск, сеть, USB-устройство, и, возможно, дискета. Выберите оптический диск. Оптический диск будет доступен и начинается процесс установки.
Через несколько минут, вам будете представлены первые из нескольких установочных диалогов Windows 7. Типичная установка требует только выбора по умолчанию (или рекомендуется) настройки и нажав на кнопку «Далее» с каждым окном. В самом начале вам будет предложено выбрать язык для установки (например, на английском языке), время и формат валюты, а также ввод / тип клавиатуры (например, США). Затем, вам будет предложено нажать на кнопку Install Now (это будет единственным вариантом для продолжения). Вам будет предложено принять условия лицензирования.
Затем Вам дали опции для выборочной установки или обновления. Вы выбрали бы обновление, если у вас уже была версия Windows 7 установлена и смотрели, чтобы модернизировать систему. Это может быть в случае, например, если ваш Windows 7 были частично повреждены или лет устарели. Выборочной установки не сохраняет все предыдущие файлы, параметры и программы, в то время как обновление сохраняет их все. Пользовательская установка может также позволяет менять диски и разбиение дисков. Затем Вы спросили, где окна должны быть установлены. Если у вас есть только один жесткий диск, нет никакого выбора, чтобы сделать.
С этого момента программа установки будет работать на некоторое время без прерывания или необходимости пользовательского ввода (возможно, от 10 до 20 минут в зависимости от скорости вашего оптического привода и процессора). В это время, Windows будет перезагружен несколько раз. При повторном запросе, вам будет предложено создать учетную запись начального пользователя и назвать ваш компьютер. По умолчанию имя компьютера просто имя учетной записи вы ввели с последующим -PC. Например, если вы укажете имя Zappa, то компьютер будет по умолчанию Zappa-PC. Вы можете, конечно, изменить имя по умолчанию. В следующем окне есть вы закончите информацию учетной записи, предоставляя начальный пароль для учетной записи вместе с "намек" в случае, если вы склонны к забывая пароль. Этот первоначальный счет позволит пользователю немедленно начать использовать компьютер, не требуя, чтобы администратор создать учетную запись.

Прежде чем продолжить, Windows в настоящее время запрашивает ключ

продукта. Это код, который умело задан на упаковке CD. Ключ будет представлять собой комбинацию букв и цифр и должен быть длиной 25 символов. Это гарантирует, что ваша версия Windows, имеет права и не была взломана.

Следующий шаг для Windows, для выполнения автоматических обновлений. Хотя этот шаг не является обязательным, это очень полезно. Это позволяет вашей установке, получить самые последние патчи операционной системы Windows 7. Без этого шага, вы будете ограничены установкой версии ОС, так как она существовала, когда компакт-диск был изготовлен. Если вы решили пропустить этот шаг, Windows установка будет обновления на более позднее время, например, в первый раз вы пытаетесь выключить компьютер.

После установки обновления, вы установите часовой пояс и будет возможность корректировки даты и времени. И, наконец, вам будет предложено указать текущее местоположение компьютера, который является по сути выбора сети для вашего компьютера, чтобы попытаться подключиться. Ваши варианты домашней сети, работа сети и сети общего пользования. Это вариант, который можно сбросить в более позднее время. После выбора, ваш компьютер будет пытаться подключиться к сети. И, наконец, ваш рабочий стол подготовлен и операционная система инициализирует в пользовательском режиме. Вы готовы работать!

Хотя вы теперь готовы использовать ваш компьютер, Windows загружается с параметрами, которые были установлены программистов Windows. На данный момент, если вы хотите внести изменения в свой рабочий стол, вы должны сделать это с помощью панели управления. Вы можете, например, изменить стиль окна, фон рабочего стола, настройки цвета, используемые, разрешение экрана, а также от размера значков рабочего стола. Можно также указать, какие программы должны быть прикреплены к панели задач, которая проходит вдоль нижней части рабочего стола, а также те, которые должны появиться в верхней части меню программ. Вы также должны убедиться, что ваш сетевой брандмауэр работает.

Установка Windows 7 отнимает не очень много времени и легка в установке. От начала до конца, вся установка должна занять менее часа, возможно, всего 30 минут.

Установка Linux

Аналогично Windows 7, установка Red Hat Linux может быть сделана

с помощью компакт-диска. Здесь мы предполагаем, что у вас есть установочный компакт-диск CentOS 5.5. Установка сложнее, чем с Windows, и требует понимания таких понятий, как разбиение диска (диск разбиений описаны в главе 5).
При загрузке с установочного компакт-диска, вам будет представлен экран, который имеет несколько вариантов установки, таких как тестирование средств массовой информации (на самом деле не требуется, если вы не создали компакт-диск чтобы установить на свой собственный) и настройки языка по умолчанию.
Теперь вы достигнете шага разбиения диска. В новой установке , вполне вероятно, что жесткий диск не разбит на разделы, и вы должны будете указать разделы. Если вы устанавливаете Linux на машине с существующей ОС, вы должны быть осторожны, чтобы это был свободный раздел диска. Обратите внимание, что это не обязательно требует двух или более жестких дисков. "Свободный диск" является логическим назначение и может быть разделен, который не является частью другой ОС. Если разбиений уже существуют, которые вы хотите удалить, необходимо выбрать опцию «Удалить все разделы Linux на выбранных дисках и создать макет по умолчанию" и выберите "Обзор и изменить разбиение макета" флажок.
Здесь необходимо указать разделы вашего диска Linux. Вы хотите, чтобы различные разделы для каждого из корневого раздела, подкачки, а также пользовательских каталогов. Вы можете выбрать более тонкую группу разделов путем, например, имея раздел для / вар и для / USR; в противном случае, эти каталоги будут помещены под корень. Для каждого раздела, вы должны выбрать точку монтирования (каталог) или файловую систему (для подкачки), размер раздела, и должен ли раздел быть установлен в размере, или заполнить оставшееся пространство, доступное. В качестве примера, можно разделить диск Linux следующим образом:

• Точка монтирования: выберите / (корень), размер 4000 (т.е., составляет 4 ГБ), фиксированный размер
• Точка монтирования: выберите / Var, размер 1000, фиксированного размера
• Тип файловой системы: замена (не выбрать точки монтирования), размер 1000, фиксированный размер
• Точка монтирования: выберите / дома, чтобы заполнить максимально допустимого размера

На следующем экране вам будет предложено параметры по установке. GRUB по умолчанию и должен быть выбран и установлен на основной жесткий диск, который, вероятно / DEV / sda1. Для сетевых устройств, необходимо указать, каким образом IP-адрес должен быть создан. Наиболее распространенной техникой является, чтобы он назначается сервером через DHCP. Последний вопрос, чтобы выбрать часовой пояс. Вам будет предложено указать корневой пароль системы. Этот пароль будет использоваться системным администратором каждый раз, когда требуется случайная работа системного администрирования, таких как путем создания учетной записи или установки программного обеспечения. Вы хотите использовать пароль, который вы не забудете. На следующем экране вы можете указать, какое программное обеспечение автоматически должно быть установлено с CentOS. Desktop-Гном должен быть уже выбранным . Вы можете выбрать другое программное обеспечение на данный момент, или настроить позже. Установка готова. Этот процесс обычно занимает от 5 до 10 минут. Когда это сделано, вам будет предложено перезагрузить систему. После перезагрузки, вы завершить процесс установки. Это включает в себя создание начальных настроек брандмауэра (по умолчанию, вероятно, будет достаточно) и хотите ли вы, обеспечить соблюдение SELinux (безопасности улучшенного качества). Опять же, по умолчанию (исполнение) лучше. Вы можете установить дату и время. И, наконец, вам будет предложено создать первоначальную учетную запись пользователя, так же, как с Windows. Эта учетная запись необходима для того, чтобы, как пользователь, вы можете войти в графический интерфейс. Не рекомендуется, чтобы вы когда-нибудь вошли в GUI как корень. В отличие от Windows, вам не нужно перезагружать компьютер в данный момент, чтобы начать работу с системой; вместо этого, вы можете продолжить работу, войдя в систему под учетной записью пользователя, которую только что создали.

ГЛАВА 5. ФАЙЛЫ, КАТАЛОГИ И ФАЙЛОВАЯ СИСТЕМА

В этой главе рассматриваются логические и физические реализации файловых систем. Логическим представлением файловой системы является организационные единицы файлов, каталогов и разделов. Физическая реализация является преобразованием из логического представления в физический участок диска хранения компонентов файла. Файловые системы как DOS/Windows, так и Linux представлены каталогом верхнего уровня и реализацией блока индексации. Глава завершается разделом, который исследует, как перемещаться по файловой системе с помощью инструкций командной строки.
Целями обучения данной главы являются:

• описание элементов файловой системы, как с логической, так и с физической точки зрения.
• Описание способов физического хранения файлов и доступа к ним.
• Представление использования каталогов верхнего уровня как в Windows, так и в Linux.
• Иллюстрирование навигации по файловой системе с помощью примеров Linux и DOS команд.

В этой главе мы рассмотрим компоненты файловой системы и посмотрим детально как на файловую систему Windows, так и на файловую систему Unix/Linux.

Файлы и каталоги

Файловая система существует на двух уровнях абстракции. Существует физическая природа файловой системы – это устройства хранения данных, индекс файл и распределяющий процесса именования и определения его физического местоположения в пределах жесткого диска (и других носителях). Существует также логический характер файловой системы – это то, как физическая система разбивается на независимые именованные объекты. Как пользователи, так и системные администраторы, мы в

основном интересуемся логической файловой системы. Детали того, как файлы и каталоги распределены, становится интересным только, когда мы должны устранить неполадки файловой системы (например, поврежденные файлы, необходимость дефрагментации файловой системы). Далее в этой главе мы кратко рассмотрим физическую природу файловых систем, но сейчас мы сосредоточимся на логической стороне. В файловой системе, у вас есть три типа объектов:

• Разделы
• Каталоги
• Файлы

Раздел является логическим подразделением в пределах файловой системы. В операционной системе Windows, раздел обозначается с помощью буквы, например, C:\ или с D:\ могут оба относиться к частям жесткого диска, но к двум отдельным разделам. В Linux, раздел относится к отдельному диску логического диска, хотя, на самом деле, несколько разделов могут находиться на одном диске. Мы устанавливаем определенные каталоги в разделе, сохраняя, таким образом, некоторые из каталогов разделенными. Например, пространство подкачки будет размещено на одном из разделов, домашние каталоги пользователей на втором, и, вполне вероятно, операционная система (ОС) Linux на третьем. Обычно вы создаете раздел при первой установке операционной системы. После установки, размер или количество разделов изменить неудобно, потому что это может уменьшить пространство существующего раздела, что потребует перемещения или удаления некоторых файлов в этом разделе. Разделы не столь интересны, и вполне вероятно, что как только вы установили свою ОС, вам не придется снова беспокоиться об отдельных разделах.
Каталоги, также известные как папки, позволяют пользователю координировать размещение файлов. В отличие от разделов, которых мало, вы можете создать столько каталогов, сколько вам хотелось бы. Каталоги могут быть размещены внутри каталогов, давая пользователю возможность создавать иерархию файлового пространства. Такой каталог обычно упоминается как подкаталог. Иерархическое файловое пространство часто представляется в форме "дерева". На вершине, или корне, дерева находятся разделы. Ниже находятся каталоги верхнего уровня. Еще ниже файлы и подкаталоги. Дерево ветвится так, что листья находятся в нижней части. Программы диспетчеры файлов

часто иллюстрируют дерево слева направо (левая сторона имеет корень дерева); двигаясь в правом направлении, у вас расширяются подкаталоги, и листья (которыми являются отдельные файлы) находятся на правом конце любой ветви. Рисунок 5.1 иллюстрирует типичное файловое пространства, как оно отображается в программном обеспечении диспетчера файлов: Windows на левой стороне и Linux справа. В обеих операционных системах, самым верхним уровнем является сам компьютер, который имеет под собой различные файловые пространства или диски. В современных компьютерах, обычно распространено иметь жесткий диск (C:) и оптический диск (D:). Там могут быть перечислены дополнительные диски, такие как внешний жесткий диск, флоппи-диск, USB флэш-диск, файловый сервер, доступные по сети, или разделы в пределах самого внутреннего жесткого диска. В этом последнем случае, в Linux, все они могут по-прежнему появляться под файловой системой, как показано на рисунке 5.1. Верхний уровень файловой системы Linux содержит ряд предустановленных каталогов. В операционной системе Windows, файловая система, как правило, ограничивается исходными каталогами Program Files (две директории в Windows 7 для обозначения 32-разрядных программ и 64-разрядных программ), Users (домашний каталог пользователя), а также Windows (ОС). В Linux, структура каталогов является более определенным. Мы детально изучим использование различных каталогов позже в этой главе.

Файлы являются значимыми единицами хранения в файловой системе. Они включают в себя данные и программы, хотя мы можем считать

Рисунок 5.1
Типичные файловые пространства

программы специальными типами файлов данных. Файлы данных бывают всех размеров и типов. Файлами данных могут быть текстовые файлы, отформатированные текстовые файлы, с использованием особого выхода или управляющих символов, или двоичные файлы. В Linux, текстовые файлы очень распространены. В операционной системе Windows, файлы данных чаще всего создаются такими программами, как Word (DOCX) и Excel (XLSX). Файлами данных могут быть также мультимедийные файлы, такие как музыкальные или звуковые файлы (WAV, MP3), файлы изображений (GIF, JPG, BMP) и видео файлы (MPG, AVI, MOV).

Файлы имеют имена. В более старых ОС, имена файлов были ограничены до восьми символов с последующим дополнительным расширением, которое может составлять до трех символов. Разделение имени от расширения является периодом. В более старых ОС, символах , используемых в именах файлов, были ограничены только алфавитными, цифровыми, с подчеркиванием и символом дефис. Сегодня имена файлов могут включать в себя пробелы и некоторые другие символы пунктуации, хотя и не каждый символ является допустимым. Более длинные имена часто бывают полезны, потому что имя должно быть очень описательного характера. Некоторые пользователи игнорируют расширения, но расширения имеют большое значение, потому что они говорят пользователю, каким типом файла он является (то есть, какое программное обеспечение создало его). Расширения файлов часто используются ОС, чтобы определить, каким образом должен быть открыт файл. Например, если бы вы дважды щелкнули мышью на значке файла с данными, то ОС смогла бы автоматически открыть этот файл в соответствующем программном обеспечении, если файл имеет расширение, а расширение сопоставляется с соответствующим программным обеспечением.

Помимо типов файлов и имен, файлы имеют другие свойства ссылок. Их логическое расположение в файловой системе обозначается указателем. То есть, дескриптор файла будет также содержать его местоположение в пространстве файловой системы. Например, C:\Users\foxr\Мои документы\cit130\Примечания\ch7.docx описывает логическую позицию файла в файловой системе, а не его физическое местоположение. Дополнительный процесс должен преобразовываться из логического местоположения в физическое.

Размер файла, время последнего изменения и дата, и владелец также являются полезными свойствами. В зависимости от операционной

системы, время создания, дата и группа, которой принадлежит файл, также могут быть записаны. Вы можете найти эти свойства в Windows, щелкнув правой кнопкой мыши на любое имя файла из проводника Windows и выбрав Свойства. В Linux, вы можете увидеть эти свойства, когда вы выполните LS -l команду.

Файлы не обязательно хранятся в одном непрерывном блоке дискового пространства. Вместо этого, файлы разбиты на фиксированные размеры, известные как блоки. Блоки файлов не могут распространяться на диске так, чтобы они находились на одной дорожке или секторе, или даже на той же самой поверхности. Вместо этого блоки могут быть распределены по нескольким поверхностям диска. Это будет рассмотрено более подробно далее в этой главе.

Команды многих старых ОС были ориентированы на управление файловой системой. Операционная система ДОС, например, означает Дисковую Операционную Систему. Кроме команд дисковых операций существовало еще несколько команд, которые нужны были пользователю. Это не означает, что операционная система выполняет только операции в файловой системе, но что пользовательские команды разрешали несколько других типов операций. Общие команды в обоих MS-DOS и Linux перечислены ниже. Кроме того, предусмотрены операции для Windows мыши. Мы будем изучать Linux и команды DOS более подробно в разделе Перемещение по файловой системе.

Linux:

• ls - список файлов и подкаталогов данного каталога
• mv - переместить или переименовать файл, или каталог
• cp - скопировать файл или папку в новое место
• rm - переместить (удалить) файл
• cat, более, менее - отобразить содержимое файла на экран
• mkdir создать новый (под)каталог
• rmdir-удалить каталог; каталог должен быть пустым для этой команды

DOS:

• dir - список файлов и подкаталогов данного каталога
• move - переместить файл на новое место или переименовать файл
• copy - скопировать файл в новое место
• del - переместить (удалить) файл
• type - отобразить содержимое файла на экран
• mkdir - создать новый (под)каталог
• rmdir - удалить каталог; каталог должен быть пустым для этой команды

ОС Windows (с помощью проводника Windows, см Рисунок 5.2):

• Для просмотра содержимого каталогов, щелкните по имени каталога в левой панели; его содержимое отображается в правой панели, если каталог не на самом высоком уровне, вам придется развернуть его исходные каталоги из верхней части файловой системы, пока не достигнете искомого каталога.
• Чтобы переместить файл, перетащите значок файла из правой панели в текущей директории в папку в левой панели.
• Чтобы скопировать файл, нажмите правой кнопкой мыши на иконке файла, выберите копировать, переместитесь в новый каталог, нажав на этот каталог в левой панели, а затем в правой панели, щелкните правой кнопкой мыши и выберите команду Вставить.
• Чтобы переименовать файл, нажмите правой кнопкой мыши на иконке файла, выберите команду переименовать (или дважды медленно нажмите на иконке имени файла), а затем введите новое имя.
• Чтобы удалить файл (или папку), перетащите значок в мусорную корзину (или правой кнопкой мыши щелкните на иконке и выберите Удалить или щелкните левой кнопкой мыши на иконке и нажмите кнопку Удалить)

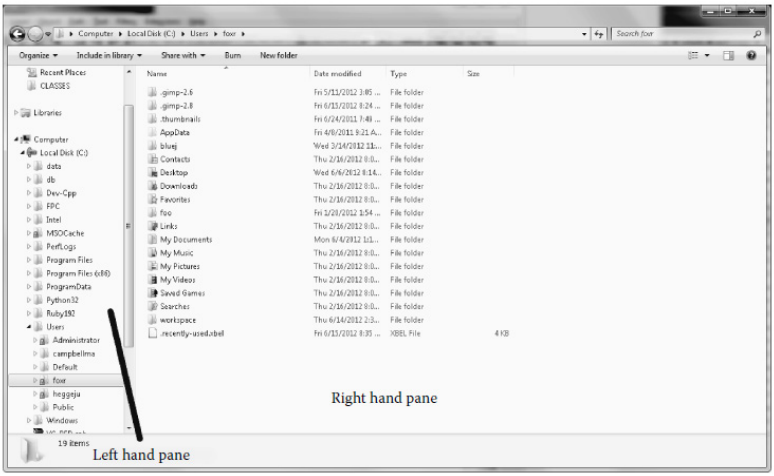


Рисунок 5.2
Проводник Windows.
Right hand pene – правая панель; left hand pane – левая панель

-вы придется отдельно очистить корзину, чтобы навсегда удалить файл.
• Чтобы создать новую директорию (папку), нажмите на кнопку New Folder. Или, щелкните правой кнопкой мыши в правой панели и выберите New, а затем Folder, а затем введите имя папки, как только она появится.

• Чтобы отобразить содержимое файла, вы можете дважды щелкнуть на значке файла, и он должен запуститься в программном обеспечении, связанном с расширением файла.

Обратите внимание, что в Windows, расширения файлов могут не отображаться в менеджере файлов, поэтому вам придется настроить его так, чтобы расширения файлов появились (если вы хотите их видеть).
Для того чтобы уменьшить влияние, которое большие файлы имеют на доступное пространство в файловой системе, пользователь может выполнить *сжатие файлов*. Есть две формы сжатия файлов. В сжатии файлов *без потерь*, файл уменьшается в размерах таким образом, что, никакие данные не будут потеряны, когда он распаковывается к своей первоначальной форме. Есть много алгоритмов для выполнения сжатия файлов без потерь, и они будут работать особенно хорошо над текстовыми файлами. При сжатии файла без потерь, вы обычно должны распаковать файл, прежде чем вы можете получить доступ к нему (есть исключения, такие как алгоритм сжатия звука FLAC). Преимуществом сжатия файла без потерь является то, что файлы, которые не используются в настоящее время, могут быть сохранены в значительно уменьшенных размерах. Основным недостатком является то, что сжатие и распаковка файлов отнимает много времени.
Сжатие файлов *с потерей* фактически выбрасывает некоторые данные, чтобы уменьшить размер файла. Большинство форм потокового аудио и видео файлов — это файлы с потерей данных. К ним относятся .avi, .mpg, .mov (видео) и .wav и .mp3 (аудио). Большинство форм хранения изображений также используют сжатие файлов с потерями, такие как .jpg. Если бы вы сравнили растровый файл (.bmp) в формате .jpg, то вы бы увидели, что .bmp имеет большую (возможно, гораздо большую) чистоту. Файл .jpg экономит пространство за счет "размывания" ближайших пикселей. Формат .gif, а не отбрасывает некоторые из цветов, так что если изображение использует только стандартную палитру цветов, то вы не увидите каких-либо потерь в ясности. Однако, если .gif использует различные цвета, то изображение, имея при этом одну и ту же чистоту, не будет иметь правильных цветов.

Когда у вас есть набор файлов, то другим полезным действием является связывание их вместе в архив. Это позволяет легко перемещать файлы в качестве группы то ли размещаете вы их в резервном хранилище, то или выгружаете/загружаете их через Интернет. Архивирование файлов распространено, когда программист хочет поделиться программами с открытым исходным кодом. Исходный код, вероятно, включает в себя множество файлов, возможно, десятки или сотни. Архивируя коллекцию файлов, весь набор исходного кода может быть загружен одним действием, а не требуя от пользователя загружать файлы по частям. Одна из самых распространенных форм архивирования является через zip(зип) формат файла, который выполняет как сжатие файлов, так и архивирование. Чтобы извлечь «зазиппованный» файл, файл нужно распаковать (unzip). Существуют многочисленные популярные программные пакеты для архивирования(zip) и разархивировать(unzip) в том числе и WinZip WinRAR в Windows, и Gzip и gunzip в Linux. Linux также имеет старые средства для архивирования под названием tar. Программа tar (архив ленты) первоначально использовалась для сбора файлов вместе в пучок и сохранения их на ленточном носителе информации. На сегодняшний день tar используется для архивирования файлов в первую очередь для передачи через Интернет.
Программа tar не сжимает, поэтому необходимо сжать архивный файл, чтобы уменьшить его размер.
Сжатие с потерями в основном используется в файлах изображения, звука и видео, в то время как сжатие без потерь используется в текстовых файлах и исходных файлах программного обеспечения. Величина сжатия, то есть уменьшение размера файла, зависит от применяемого алгоритма сжатия и от самих данных. Текстовые файлы часто можно значительно уменьшить за счет сжатия, на целых 88%. Сжатие аудио с потерями обычно может сжимать файлы данных вплоть до 10% или 20% от их первоначальной версии, в то время как сжатие аудио файлов без потерь уменьшает файлы только до 50% от их первоначального размера.

Файловые системы и диски

Файловая система - это иерархическая структура, которая включает в себя физическое и логическое пространство файла. Файловая система включает в себя индекс для разметки от логического имени файла к физическому местоположению. Файловая система также включает в себя возможность роста или сокращения файлового пространства

путем добавления и удаления пространства. Это иногда называют *монтаж*. Смонтированное файловое пространство — это то, которое в дальнейшем может быть удалено (размонтировано). Файловая система физически состоит из вторичных устройств хранения данных, наиболее преимущественно внутреннего жесткого диска, внутреннего оптического привода и возможно внешнего накопителя на жестких дисках. Кроме того, пользователи могут подключить флэш-диски и накопители на магнитной ленте. Для сетевого компьютера, файловое пространство может включать в себя пространство удаленных диски (часто называемые файловые серверы).

Операционная система поддерживает файловое пространство на двух уровнях: логическом и физическом. Наше представление файлового пространства - это логическое представление - разделы, каталоги и файлы. Для того, чтобы получить доступ к файлу, операционная система разметить путь от логического расположения до физического местоположения. Это делается путем записи физического расположения файла в информации его каталога. Место обычно указывается посредством указателя. Предположим, что мы имеем дело с файлами, хранящимися только на жестком диске. Указатель указывает на конкретный *блок* файловой системы. Он сам должен быть переведен из целого числа в блок привода, поверхность (если мы имеем дело с многослойным диском), в сектор, а также дорожку на диске (Рисунок 5.3). Термин цилиндр используется, чтобы выразить одни и те же номера

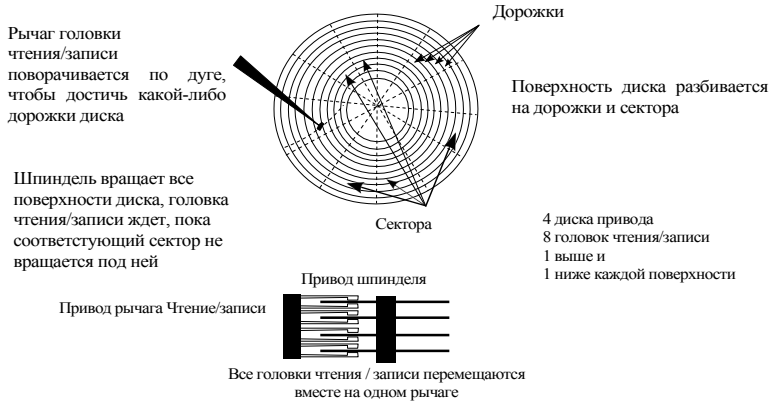


Рисунок 5.3

Макет жесткого диска. (Частично адаптировано из публичного домена изображения LionKimbro в http://commons.wikimedia.org/wiki/File:Cylinder_Head_Sector.svg.)

дорожки и сектора во всех пластинах вращающегося диска. Мы могли бы сослаться на цилиндр, когда мы хотим получить доступ к одним и той же дорожке, и сектору одновременно на каждой поверхности. В качестве примера, представьте файл f1.txt находится в блоке 381551. Наша файловая система состоит из одного жесткого диска с четырьмя пластинами или восемью поверхностями. Кроме того, каждая поверхность имеет 64 сектора и 2048 дорожек. Мы хотели бы найти этот блок на поверхности 381551/(64*2048)=поверхности 2 (начиная отсчет на поверхности 0). Поверхность 2 будет на самом деле содержать блоки от 262144 до 393216. Мы нашли бы блок 381551 в секторе 58 (опять же, начиная отсчет в секторе 0). так как (381551 - 262144)/2048=58.3. И, наконец, мы нашли бы блок 623 (381551-262144)-2048*58=623. Итак, fl.txt, расположенный в блоке 381551, находится на поверхности 2, сектор 58, трек 623.

Как показано на рисунке 5.3, жесткие диски содержат несколько пластин. Пластины вращаются в унисон, так как моторизованный шпиндель связан с каждой пластиной через отверстие. Скорость вращения дисков зависит от типа диска, но приводы жестких дисков вращаются со скоростью от 5400 до 15000 об/мин. Каждая пластина будет иметь две головки чтения/записи, связанные с ней (так что можно использовать обе поверхности пластины). Все головки чтения/записи жесткого диска перемещаются в унисон. Рычаги головок чтения/записи управляются с помощью исполнительного механизма, который перемещает их по поверхности дисков, в то время как диски вращаются с помощью шпинделя. Головки чтения/записи на самом деле не прикасаются к поверхности диска, а парят чуть выше/ниже поверхности на воздушной подушке,

Рисунок 5.4
Рычаг головки чтения/записи. (Предоставлено Hubert Berberich, http://commons.wikimedia.org/wiki/File:Seagate-ST4702N-03.jpg.)

создаваемой быстрой скоростью вращения дисков. Важно, чтобы головки чтения/записи не прикасались к поверхности диска, так как они могут повредить поверхность (поцарапать или помять его) и уничтожить данные, хранящиеся там. По этой причине важно "припарковать" рычаг головки чтения/записи перед перемещением компьютера, так чтобы вы не рисковали повредить диск. Рычаг головки чтения/записи паркуется при выключении компьютера. Многие современные портативные компьютеры используют датчики движения для обнаружения резких движений, так что рычаг головки чтения/записи может быть быстро припаркован в случае движения рывками или если весь ноутбук упал.

Данные сохранятся на жестком диске в виде положительных и отрицательных магнитных зарядов. Головка чтения/записи может считывать заряд с поверхности или написать новый заряд (операции загрузить/открыть и запомнить/сохранить, соответственно). Пример рычага головки чтения/записи для восьми пластин (шестнадцать головок, видна только верхняя головка) показан на рисунке 5.4.

Файлы не сохраняются последовательно по всей поверхности диска. Вместо этого файл разбивается на участки фиксированного размера, называемые блоками, а блоки файла распределены по поверхности диска, возможно, через несколько поверхностей. Рисунок 5.5 иллюстрирует эту идею, где файл, состоящий из 6 блоков (с номерами от 0 до 5) расположен в разных местах дисковых блоков, расположенных по всему жесткому диску. Обратите внимание, что поскольку каждый блок имеет одинаковый размер, то последний блок не может быть заполнен до объема. Пунктирная линия в дисковом блоке 683 показывает фрагмент - то есть, конец файла не заполняет весь блок диска, поэтому часть его не используется.

Существует несколько причин для рассеяния файловых блоков по всей поверхности(ям) диска. Во-первых, из-за скорости, с которой диск

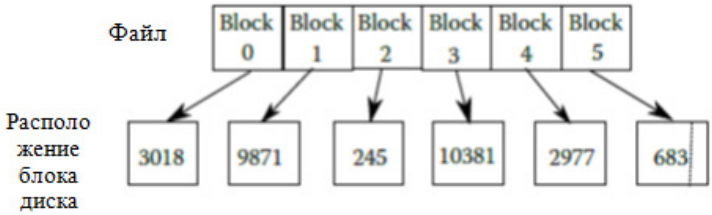


Рисунок 5.5
Блоки файла, размеченные на блоках диска

вращается, если головка чтения/записи должна иметь доступ к нескольким последовательным блокам диска, то вполне вероятно, что время, которое требуется, чтобы передать ценное содержимое блока в память будет больше, чем время, которое потребуется диску, чтобы переместить следующий блок под головкой чтения/записи. Результатом будет то, что головка чтения/записи будет вне позиции для следующего доступа и, следовательно, доступ должен был бы ждать, пока соответствующий блок вращался бы под головкой чтения/записи, то есть другого вращения диска. Одним из способов борьбы с этой проблемой является вынесение дисковых блоков, в то же общее место, но не смежное. Например, блоки диска могут быть сохранены на любом другом секторе данной дорожки, а затем на следующей смежной дорожке (например, сектор 0/дорожки 53, сектор 2/дорожки 53, сектор 4/дорожки 53, сектор 6/дорожки 53, сектор 8/дорожки 53, сектор 0/дорожка 54, сектор 2/дорожка 54).

Во-вторых, это устраняет проблему фрагментации. Как показано на рисунке 5.5, только последний дисковый блок файла может содержать фрагмент. Рассмотрим, что может произойти, если файлы на диске помещаются в последовательных ячейках. Представьте, файл хранится в 3-х последовательных блоках. Сразу же после этого хранится второй файл в 2-х последовательных блоках. Третий файл хранится в 4-х последовательных блоков после второго файла. Второй файл отредактирован и расширен. Он больше не помещается в 2-х блоках, и поэтому сохраняется после третьего файла. Теперь у нас есть файл 1 (3 блока), освободившееся место (2 блока), файл 3 (4 блока) и файл 2 (3 блока). К сожалению, это свободное пространство из 2-х блоков не может быть полезным, так как новым файлам, возможно, потребуется больше места для хранения. Таким образом, мы имеем фрагмент из 2-х блоков. Вполне возможно, что в какой-то момент, нам бы понадобилось пространство из 2-х блоков для повторного использования, но оно может оставаться не используемым в течение довольно длительного времени. Это похоже на то, что произойдет, если вы не записываете несколько объектов на видео или кассеты и решите, что вам больше не нужен один из предыдущих объектов. Вы можете или не можете быть в состоянии заполнить пространство элемента, которой вам больше не нужно. Так как мы часто редактируем и пересохраняем файлы, то использование последовательных блоков диска, вероятно, создаст десятки или сотни непригодных фрагментов по всей файловой системе. Таким образом, путем распределения блоков, мы можем использовать любой свободный блок в любое время, когда нам нужен новый блок.

В-третьих, описание доступного свободного пространства в файловой системе поддерживать легче, используя один и тот же механизм для отслеживания дисковых блоков файла. Свободное пространство состоит из таких блоков диска, которые либо еще не были использованы, или от удаленных файлов. Таким образом, мониторинг и запись, которые используются и бесплатны, не займет дополнительного дискового пространства.

Это приводит к естественному вопросу: если файл диска разбивается на блоки, которые разбросаны по файловой системе, как ОС находит данный блок (например, третий блок файла)? Напомним, что операционная система поддерживает список первого дискового блока каждого файла. Каждый блок диска содержит как часть его место хранения следующего дискового блока в файле. Хранение является указателем. Это создает то, что известно, как *связанный список*. См. Рисунок 5.6, где ОС хранит файловый указатель на первый блок, а каждый блок хранит указатель на следующий блок в файле.

Чтобы загрузить отдельный блок диска, операционная система вначале проверяет список каталогов для получения первого указателя файла. Первый блок содержит указатель на второй блок, который содержит указатель на третий блок, и так далее. Для достижения блока *n*, вы должны сначала пройти через *n*-1 предыдущих блоков, следуя этих

Рисунок 5.6
Указатели используются для определения следующего блока файла на диске.

указателей. Следование связанному списку является неэффективным, поскольку каждый блок требует отдельного доступа к диску, который, как уже отмечалось ранее, является одним из замедляющих аспектов любого компьютера. Для последовательного файла, то есть того, который должен быть доступен от начала до конца, доступ к файлу, таким образом, не является недостатком, так как файл будет загружен, в порядке блок за блоком.

Дисковые утилиты

вы можете вспомнить из главы 4, что часть ОС состоит из сервисного программного обеспечения. Это программное обеспечение включает в себя антивирусные программы, например. Другой класс полезных программ на дисковые утилиты. Дисковые утилиты могут помочь пользователю или системному администратору управлять и поддерживать файловую систему, выполняя задачи, которые в противном случае не доступны. Ниже приведен список распространенных дисковых утилит.

Дефрагментация диска: хотя использование дисковых блоков, разбросанных по всей файловой системе, и предотвращает создание фрагментов, но фрагменты все же могут возникнуть. Дефрагментация диска перемещает блоки диска через поверхности диска, чтобы переместить их вместе и удалить свободные блоки из промежутка между используемыми блоками. Это может сделать производительность диска более эффективной, поскольку он перемещает блоки файла ближе друг к другу таким образом, что, когда происходит обращение к файлу целиком, то время доступа уменьшается, потому что все блоки находятся в пределах одной и той же общей площади на поверхности. В то время как файловая система Windows, иногда может извлечь выгоду из дефрагментации, файловая система Linux предположительно не станет неэффективной с течением времени.
Восстановление файлов: если вы удалите файл, а затем очистите корзину, то этот файл ушел, не так ли? Не так. Программы восстановления файлов могут попытаться собрать воедино файл и восстановить его, до тех пор, пока дисковые блоки, составлявшие файл, не были переписаны заново новым файлом.
Резервное копирование данных: многие пользователи не резервируют копию файловой системы, и поэтому, когда возникают катастрофические ошибки, их данные (возможно, многолетние) теряются. Резервные копии

чрезвычайно важны, но ими иногда трудно управлять, вы резервируете копии всего или только того, что изменилось с момента последнего резервного копирования? Утилита резервного копирования может помочь вам управлять процессом резервного копирования.

Многие файлы могут быть доступны случайным образом, то есть, доступ к любому блоку может быть получен в любое время. Чтобы учесть это, некоторые операционные системы, включая Windows, используют таблицу размещения файлов (FAT). FAT накапливает все указатели каждого дискового блока в один файл, хранящийся в начале диска. Затем этот файл загружается в память, так что ОС нужно искать только в памяти, чтобы найти местоположение данного блока, а не искать сам диск. Поиск памяти надлежащего места блока на диске является гораздо более эффективным, чем поиск диска. На рисунке 5.7 показана часть FAT. Обратите внимание на то, что каждый блок на диске хранится в FAT, с своим наследным блоком. Таким образом, в данном случае мы видим, что файл, который содержит блок 150 имеет следующий блок на месте 381, в то время как файл, который содержит блок 151 имеет следующее местоположение в 153 и после 153, следующее расположение 156, которое является последним расположением в файле. Блок 152 является плохим секторам, а блоки 154, 155 и 732 являются частью другого файла. EOF означает "конец файла".

Свободное пространство также может быть сохранено с помощью связанного списка, где первый свободный блок указывается специальным указателем в разделе файловой системы, а затем каждый последующий свободный блок указывается предыдущим свободным блоком. Чтобы удалить файл, нужно только изменить пару указателей. Последний свободный указатель пространства будет указывать на первый блок в удаленном файле, и запись в каталоге изменяется с указателя на первый блок на пустое значение (чтобы указать, что файл больше не существует). Обратите внимание на то, что вы можете легко восстановить удаленный файл (восстановить файл), если удаленные блоки файлов не были использованы повторно. Это происходит потому, что удаление файла не удаляет файл с диска физически, а вместо этого добавляет дисковые блоки файла в список свободных блоков. Смотрите обсуждение восстановления файлов в боковой панели на предыдущей странице.

Указатели из файловой системы (каталога) на физическое расположение на диске иногда называют *жесткими ссылками*. То есть, связывание имени файла на месте является жесткой связью. Linux позволяет иметь несколько указателей жестких ссылок на файл. Некоторые операционные

системы также предлагают *мягкие ссылки*. В мягкой ссылке, указатель из файловой системы указывает не на физическое местоположение файла, а на другую запись в структуре каталога. Мягкие ссылки, также известный как символические ссылки в Linux и ярлык в Windows, указывает на оригинальный ввод. Например, если вы создаете файл f1.txt в директории /home/foxr, а затем создаете мягкую ссылку под названием f2.txt в /home/ foxr/cit130/stuff, а затем каталог /home/foxr/cit130/ stuff имеет запись f2.txt, чей указатель указывает на /home/foxr/f1.txt, а не к первому блоку самого файла. См Рисунок 5.8 для примера, где две жесткие ссылки, под названием ФАЙЛ1 и ФАЙЛ2, указывают на первой блок файла, в то время как мягкая ссылка, называется File3, указывает на жесткую связь File2.

При использовании любого вида связи, создаются псевдонимы. Псевдоним означает, что на конкретной объект (в данном случае файл) можно ссылаться с помощью нескольких имен. Псевдонимы могут быть опасны, потому что кто-то может изменить элемент, не понимая, что они меняют еще один пункт. Например, если бы я должен был изменить f1.txt, то f2.txt меняется, а поскольку f2.txt является, лишь указателем на f1.txt. Хуже обстоит дело, когда удаляется исходный файл. Операционная система не обязательно

проверяет, чтобы увидеть, есть ли мягкие ссылки, указывающие на файл, который находится в процессе удаления. Результатом является то, что файл больше не существует, но мягкая ссылка все же указывает на него. В этом случае, следование по мягкой ссылке дает ошибку. Мягкую ссылку можно удалить, не удаляя сам физический файл или вызывая проблему, но удаление исходного файла без удаления мягких ссылок может привести к ошибкам.

Доступ к диску гораздо медленнее, чем скорость процессора или доступ к основной памяти. Время доступа к диску представляет собой комбинацию из нескольких различных факторов. Адрес должен быть размечен от логической позиции до блока на диске через FAT с последующим отображением на физическом расположении с точки

Таблица размещения файлов (часть)

блок	150	151	152	153	154	155	156
следующее местоположение	381	153	Bad	156	155	732	EOF

Рис.5.7
часть таблицы размещения файлов

зрения поверхности, сектора и дорожки. Это относительно быстро, так как эта разметка выполняется с помощью доступа процессора к памяти вместо диска. Теперь, головка чтения/записи должна быть установлена над соответствующим местом на диске. Это требует перемещения головки чтения/записи по правильному пути (известному как *время поиска*) и ожидания надлежащего сектора, выделяемого под чтение/запись (известному как *задержка вращения* или *вращательный отклик*). Теперь, головка чтения/записи может начать чтение или запись магнитных зарядов на диске. Это действие требует больше времени, подразумевая, что доступ выполняется в DRAM (динамическая RAM). Поскольку читаются или записываются биты, то информация должна быть передана между диском и памятью. Это известно, как *время передачи*. Сочетание разметки (обычно ничтожно), времени поиска, задержки вращения и времени передачи составляет порядка нескольких миллисекунд (возможно от 5 до 30 мс). Миллисекунда является тысячной долей секунды. Напомним, что наши процессоры и статическая оперативная память (SRAM) работают со скоростью наносекунд (миллиардная секунды) и DRAM, возможно, от 25 до 100 раз медленнее, но все еще в диапазоне десяток наносекунд. Таким образом, время доступа к диску является фактором миллионы раз медленным, чем скорость отклика процессора и памяти!

Файловая система LINUX

Файловое пространство Linux

Файловое пространство Linux устанавливается по умолчанию с

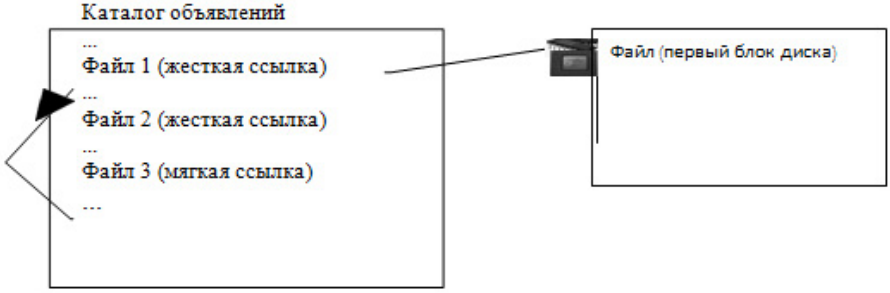


Рисунок 5.8
Жесткие ссылки и мягкие ссылки.

каталогами корневого уровня, которые заполняются основными файлами при установке Linux. Это позволяет знать, где найти определенные типы файлов. Эти стандартные каталоги:

• /bin: общие программы операционной системы, такие как ls, grep, cp, mv.
• /sbin: аналогичен /bin, но программы, хранящиеся здесь, это команды Linux, которые предназначены для системного администратора, хотя многие из них могут быть также доступны для нормальных пользователей. Примечание: /sbin часто не является частью пути пользователя, так что пользователю может потребоваться ввести полное имя в команду, например, /sbin/ifconfig.
• /etc: конфигурационные файлы, специфичные для машины (подробности приводятся ниже в этом разделе).
• /root: домашний каталог для корневого пользователя. Он, как правило, не доступен для других пользователей системы.
• /lib: общие библиотеки, используемые в процессе динамического связывания; эти файлы похожи на файлы DLL в Windows.
• /dev: файлы устройств. Это специальные файлы, которые помогают пользователю связываться с различными устройствами в системе. Некоторые из важных устройств, перечислены ниже.
• /dev/fd0 – флоппи-диск (если таковой имеется)
• /dev/hda0 - мастер IDE диск на первичном контроллере IDE
• /dev/ht0 - первый ленточный накопитель IDE
• /dev/lp0 - первое устройство параллельного принтера
• /dev/null – не устройство, но вместо этого направление для вывода программы, которую вы не хотите отобразить на мониторе - в сущности, посылая что-либо в /dev/null, заставляет его исчезнуть
• /dev/pcd0 первый параллельный порт привода CD ROM
• /dev/pt0 первый параллельный порт ленты
• /dev/random и /dev/urandom - генераторы случайных чисел (urandom имеет потенциальные проблемы, если они используются для генерации случайных чисел для криптографического алгоритма)
• /dev/sda0-первый диск SCSI на первой шине SCSI
• /dev/zero -это простой способ получить много нулей

Примечание: для устройств, оканчивающихся на 0, если у вас есть другие устройства того же типа, вы бы просто увеличьте число, например, lp1, lp2 для двух дополнительных принтеров или pcd1 для второго привода

CD-ROM.

• /tmp: временное хранение файлов для запуска программ, которым необходимо создать и использовать временные файлы.
• /boot: файлы, используемые начальной загрузкой, например, lilo (Linux Loader) или grub (Grand Unified Загрузчик).
• /tmp: точка монтирования используется для временных разделов, смонтированных системным администратором (не постоянно монтируемые разделы)
• /usr, /var, /home: точки монтирования для других файловых систем.
• /home Пространство файлов пользователей
• /var - данные выполнения, сохраненные различными программами, включая файлы журналов, файлы электронной почты, буферных файлов принтеров и заблокированных файлов
• /usr - различные системные и пользовательские приложения программного обеспечения, с подкаталогами:
- /usr/bin: здесь хранятся многие пользовательские команды Linux, хотя некоторые из них также хранятся в / bin или /USR/local/bin.
- / usr/sbin: здесь хранятся команды системного администрирования.
- /usr/share/man, /usr/share/info, /usr/share/doc: различные схемы и страницы документацию.
- /usr/include: файлы заголовков для языка программирования C.
- /usr/lib: неизменные файлы данных для программ и систем.
- /usr/local: прикладное программное обеспечение и другие файлы.
• /proc: это своеобразная запись, так как это на самом деле не физический каталог, хранящийся в файловой системе, но вместо этого сохраняется операционной системой в памяти, храня полезную информацию о процессах, таких как список драйверов устройств, сконфигурированных для системы, какие прерывания являются в данный момент ключевыми, информацию о процессоре, а также активные процессы.

Существуют многочисленные важные файлы системного администрирования в /etc . Некоторые из наиболее важных файлов перечислены здесь:
• сценарии запуска
• /etc/inittab: первоначальный сценарий запуска, который устанавливает уровень запуска и вызывает другие сценарии (в более новых версиях Linux, этот сценарий был заменен программой /etc/init)
• /etc/rc.d/rc0.d, /etc/rc.d/rc1.d, /etc/rc.d/rc2.d и т.д.: каталоги символических

ссылок, указывающих на сценарии запуска для служб, списки в каждом каталоге определяют, какие службы запускать и какие не запускать во время инициализации системы, основанной на уровне запуска (описаны в разделе Формы управления процессами в главе 4 и главе 11 для получения дополнительной информации)
• /etc/init.d: каталог, в котором хранятся многие из сценариев запуска
- Эти сценарии запуска обсуждаются в главе 11.
• Файлы учетной записи пользователя
• /etc/passwd: база данных учетной записи пользователя, с полями хранения имени пользователя, реального имени, домашнего каталога, вход в оболочку, и другая информация. Хотя это и называется файл паролей, пароли больше не хранятся там, потому что этот файл доступен для чтения кем-либо, а также размещение там паролей представляют собой угрозу безопасности; поэтому пароли теперь хранятся в теневом файле.
• /etc/shadow: хранит пароли пользователей в зашифрованном виде.
• /etc/group: аналогично /etc/passwd, но описывает группы вместо пользователей.
• /etc/sudoers: список пользователей и прав доступа, которым предоставляется ряд привилегий за пределами обычных прав доступа пользователей.

- Эти файлы обсуждаются в главе 6.

• файлы конфигурации сети
• /etc/resolv.conf: перечисление DNS-серверов локальной машины (имя домена системы серверов).
• /etc/hosts: хранит списки имен хостов, часто используемых машиной и их IP-адреса, так что поиск DNS не требуется.
• /etc/hosts.allow, /etc/hosts.deny: хранит списки адресов машин, которые либо разрешены или запрещены, журнал доступа.
• /etc/sysconfig/iptables-config: файл конфигурации брандмауэра Linux, здесь устанавливаются правила разрешения или запрета для каких типов сообщений, портов и IP-адресов.
• /etc/xinetd: файл конфигурации Интернет, разметка служб серверам, например, разметка telnet на 23/TCP, где Telnet является службой и 23/TCP – это номер порта и сервер, который обрабатывает telnet; это замена для менее безопасного конфигурационного файла inetd.

- Некоторые из этих файлов обсуждаются более подробно в главах 11 и 12.

• Файловая система файлов
• /etc/fstab: определяет файловые системы, смонтированные во время инициализации системы (также вызывается с помощью команды mount -a).
• /etc/mtab: список смонтированных файловых систем, автоматически обновляется с помощью команд mount и umount, и используется командами, такими как df.
• /etc/mime.types: определяет типы файлов; это конфигурационный файл для файла и остальных команд, таких, что эти команды знают, как трактовать данный тип файла.

• Файлы сообщений
• /etc/issue: содержит краткое описание или текст приветствия к системе, содержимое для системного администратора.
• /etc/motd: сообщение дня, автоматически выводится после успешного входа в систему, содержание для системного администратора и часто используется для доведения информации до каждого пользователя, такой как предупреждения о планируемых простоях.
• сценарии запуска пользователя
• /etc/profile, /etc/bash.rc, /etc/csh.cshrc: файлы, выполняемые при входе в систему или время запуска оболочками Bourne, BASH, или C. Это позволяет системному администратору устанавливать глобальные значения по умолчанию для всех пользователей. Пользователи также могут создавать отдельные копии их в своем домашнем каталоге, чтобы персонализировать свое окружение.

- Некоторые из этих файлов обсуждаются более подробно в главе 11.

• /etc/syslog.conf: файл конфигурации, который диктует, какие события регистрируются и где они зарегистрированы, чтобы (мы рассмотрим это подробно в главе 11).
• /etc/gdm: каталог, содержащий файлы конфигурации и инициализации для Gnome Display Manager (одна из систем Linux GUI).
• /etc/securetty: идентифицирует безопасные терминалы, т.е. терминалы, от которого корневому пользователю разрешено войти в систему, как правило, таким образом, перечислены только виртуальные

консоли, чтобы не представлялось возможным получить привилегии суперпользователя, войдя в систему через модем или сеть.
• /etc/shells: списки доверенных оболочек, используемых командой chsh, что позволяет пользователям изменять их оболочку входа в систему из командной строки.
Хотя различные реализации Linux будет меняться в конкретных формах, таких как скрипты запуска, типы GUI, а также типы и названия служб , большинство диалектов Linux имеют вышеуказанные каталоги и файлы, или аналогичные каталоги и файлы. Смотри рисунок 5.9, который показывает типичное расположение каталогов наряду с некоторыми из начальных подкаталогов.
Каталог /home конечно, будет отличаться в зависимости от численности пользователей. Отстающая общность структуры файловой системы Linux помогает системным администраторам переходить от одной версии Linux к другой. Тем не менее, с каждым новым выпуском версии Linux,

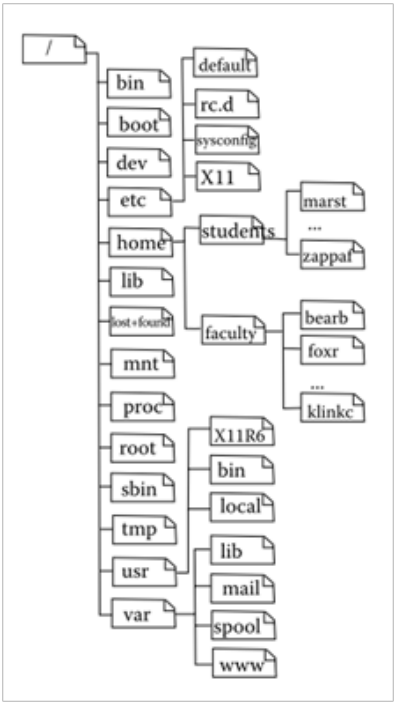


Рисунок 5.9
Типичная структура каталога Linux

вносятся изменения, и поэтому системный администратор должен идти в ногу с этими изменениями.
Файловая система Linux состоит не только из жесткого диска, но и монтируемых (съемных) устройств. Для того, чтобы получить доступ к устройству, оно должно быть физически подключено, но оно также должно быть логически установлено с помощью команды mount. Некоторый монтаж делается для вас во время загрузки системы, имея файл с именем монтирования /etc/fstab (таблица файловой системы). Чтобы отключить что-то из файловой системы, используйте umount. Для того, чтобы смонтировать все в файле fstab, используйте mountall. Примечание: вы не сможете редактировать fstab или выполнить mount, umount, или mountall, если вы не в корне. Еще один файл, /etc/mtab хранит таблицу тех разделов, которые в настоящее время установлены.

Linux Разделы

Для того, чтобы гарантировать, что различные каталоги имеют достаточно места, системные администраторы, как правило, разбивают жесткий диск на три или более областей, каждый из которых хранит различные разделы файловой системы. Это позволяет использовать конкретные области файлового пространства, которые будут предназначены для различных целей, таких как ядро, подкачка и каталог пространства пользователя. Разделы обеспечивают степень защиты данных, при которых повреждение одного раздела не повлияет на другие разделы (в зависимости от источника ущерба). Также в Linux возможно установить различные элементы управления доступом на разных разделах. Например, один раздел может быть установлен только для чтения, чтобы гарантировать, что никто не сможет записывать или удалять файлы из этого раздела. Можно также установить дисковые квоты разделов так, что некоторые разделы будут обеспечивать соблюдение квоты, в то время как другие этого делать не будут. Разделы также позволяют легко монтировать и размонтировать дополнительные файловые пространства, по желанию.
Корневой раздел (/) будет хранить Linux каталог /boot, который содержит загрузочные программы, а также достаточно места для корневых данных (таких как электронная почта и суперпользователь). Самый большой раздел, скорее всего, будет использоваться для каталога /home, который содержит все каталоги и подкаталоги пользователей. Другие каталоги, такие как /var и /usr могут быть предоставлены отдельными

разделами, или же они могут быть частью / или /home. Раздел swap используется для виртуальной памяти. Он служит в качестве дополнения к основной (DRAM) памяти, так что только в настоящее время и недавно используемые порции процессов и данных, должны постоянно находиться в памяти DRAM. Этот раздел должен иметь достаточно памяти, чтобы хранить исполняемый код всех текущих выполняемых процессов (в том числе их данных). Обычно пространства swap создается с размером в два раза превышающим размер памяти DRAM. Пространство swap также может быть распределено между двумя (или более) дисками. Если пространство swap было разделено между двумя жесткими дисками, оно может обеспечить дополнительную эффективность, так как можно было бы обрабатывать две страницы подкачки одновременно (см. файл подкачки в главе 4).

Разметка файлового пространства осуществляется во время установки OS, и, следовательно, после его завершения, разделы и их размеры фиксированы. Это печально, так как вы выполняете установку задолго до того, у вас есть пользователи, чтобы заполнить раздел пользователя, и поэтому вы должны сделать обоснованное предположение в отношении того, каким большим должен быть этот раздел. На более позднем этапе, если вы хотите изменить размеры разделов, вам, возможно, придется переустановить операционную систему, что будет означать, что все учетные записи и каталоги будут удалены. Вы могли бы, в случае необходимости, создать резервные копии всех файлов операционной системы и учетных записей пользователей/каталоги, переустановить операционную систему, изменить размер раздела, а затем восстановить файлы операционной системы и учетные записи пользователей/каталоги. Это заняло бы много времени (вероятно, заняло бы несколько часов, как минимум), и должно быть сделано, только если оригинальное разбиение на разделы неуместно. К счастью, Linux и Unix предлагают другое решение, использование динамического изменения размера разделов. Тем не менее, его использование не является чем-либо легким. Например, если сбой отключения питания или диска происходит при изменении размера, данные могут быть навсегда уничтожены. С другой стороны, это гораздо проще, чем сохранение файловой системы, переустановка операционной системы, и восстановление файловой системы.

Есть целый ряд Linux команд, которые имеют дело с файловой системой. Команда df (отобразить файловую систему) покажет вам все разделы и насколько они заполнены как в байтах, так и в процентах использования. Как уже упоминалось выше, файловые команды mount и umount

позволяют монтировать и демонтировать разделы из файловой системы. Команды quotaon и quotaoff позволяют контролировать отслеживаются ли дисковые квоты активно или нет. Системный администратор может установить дисковые квоты для пользователей и для групп. Если для данной файловой системы есть дисковые квоты, то операционная система будет гарантировать, что эти пользователи/группы, использующие эту файловую систему, ограничены установленными квотами.
Команда file будет пытаться определить, какого типа данный файл. Например, она может выдать, что файл представляет собой текстовый файл, каталог, файл изображения или документ Open Office. Инструкция find выполняет поиск файла с данным именем в файловой системе. Хотя find очень полезная инструкция Linux, она не обсуждается, потому что это довольно сложная инструкция.
Утилита fsck (проверка файловой системы) анализирует файловую систему на несовместимые файлы. Эта утилита может найти плохие сектора, файлы, которые были испорчены, потому что они все еще были открыты, когда система была закрыта в прошлом, или файлы, информация исправления ошибок которых указывает на то, что файл имеет проблему (например, вирус). Утилита fsck не только находит плохие файлы и блоки, но пытается восстановить их. Она будет искать физические и логические ошибки в файловой системе. Она может работать в режиме, чтобы найти только ошибки, или найти и попытаться исправить ошибки. Программа fsck, как правило, запускается во время загрузки системы, чтобы обеспечить безопасную и правильную файловую систему. В случае обнаружения ошибок, процесс загрузки приостанавливается, позволяя fsck найти и попытаться устранить любые найденные повреждения.

Индексный дескриптор в Linux

Индексные дескрипторы являются Unix/Linux термином для компонента файловой системы. Индексный дескриптор не является файлом, но это структура данных, которая хранит информацию о файле. При установке Linux и разбиения на разделы файловой системы, операционная система создает ряд индексных дескрипторов (примерно 1% от общего пространства файловой системы, зарезервированного для индексных дескрипторов). Обратите внимание на то, что есть заданное количество индексных дескрипторов для системы, но маловероятно, что система будет когда-нибудь выходить за доступные индексные дескрипторы. Индексный дескриптор будет хранить информацию об одном файле;

Таким образом, для каждого файла в системе есть один индексный дескриптор. Индексный дескриптор для создаваемого файла будет хранить пользователя и владельца группы этого файла, разрешения, тип файла и указатель на котором файл физически хранится, но индексный дескриптор не хранит имя файла, и не является самим файлом. Вместо этого, в списке каталогов, имя файла имеет указатель на индексный дескриптор на диске, а сам индексный дескриптор указывает на файл на диске. Если файл имеет жесткую связь, то и имя файла, и жесткая ссылка указывают на индексный дескриптор. Если файл имеет мягкую ссылку, то имя файла указывает на индексный дескриптор, в то время как мягкая ссылка указывает на имя файла.

Для того чтобы получить номер индексного дескриптора, команда в ls (список) предоставляет опцию -i. Так, ls -i имя файла возвращает номер индексного дескриптора этого файла. Вы можете получить информацию файла, через индексный дескриптор с помощью ls -l.

Файлы не хранятся в одном смежном блоке на диске, как описано ранее в этой главе, а вместо этого разбиты на блоки, которые разбросаны по файловой системе. Каждому файлу дается начальное число блоков для запуска. Индексный дескриптор, помимо хранения информации о файле, содержит ряд указателей, чтобы указать на эти первоначальные блоки. Первый указатель указывает на первый блок диска, второй указатель указывает на второй блок диска, где первый блок диска сохраняет первую часть файла, второй блок диска сохраняет вторую часть файла и т.д.

Если файл нуждается в дополнительных блоках, позже указатели индексных дескрипторов могут быть созданы таким образом, чтобы указатель индексного дескриптора указывал на блок указателей, каждый из которых указывает на дополнительные дисковые блоки. Такой блок указателя называется *косвенный блок*. Туда может быть добавлены любое количество уровней, например, указатель индексного дескриптора, может указывать на косвенный блок, который сам по себе указывает на косвенный блок, который имеет указатели на фактические дисковые блоки. Рисунок 5.10 иллюстрирует эту концепцию, показывая иерархическую структуру блоков диска, индексных дескрипторов и указателей в Linux. Индексный дескриптор освобождает ОС Linux от необходимости поддерживать систему FAT (как и в Windows).

Помимо хранения информации о файлах, индексные дескрипторы используются для хранения информации о каталогах и символических ссылках. Символическая ссылка является просто альтернативным путем доступа к файлу. Вы часто создаете символические ссылки,

если вы хотите иметь возможность указать доступа к данному файлу из нескольких исходных точек. Чтобы создать символическую ссылку, используйте ln –s имя_файла1 имя_файла2. Здесь имя_файла1 является существующим файлом и имя_файла2 является символической ссылкой. Как правило, один из этих двух имен файлов содержит путь так, что файл и символическая ссылка существуют в двух отдельных каталогах (иметь символическую ссылку в том же каталоге, что и файл, на который вы ссылаетесь, имеет мало смысла).
ё
Например, если есть важная исполняемая программа в каталоге /usr/share/progs, называемая foo, и вы хотите легко получить доступ к ней из вашей домашней директории, вы можете изменить свой домашний каталог, а затем выполнить команду ln –s /usr/shar/progs/foo foo. Это создаст символическую ссылку foo в вашем домашнем каталоге, который ссылается на файл /usr/shar/progs/foo.
Нужно быть осторожным, когда имеете дело с символическими ссылками, потому что, если кто-то удаляет или перемещает исходный файл, ссылка больше не действует. Если файл foo был удален из /usr/shar/progs, а затем введение ~/foo приведет к ошибке.

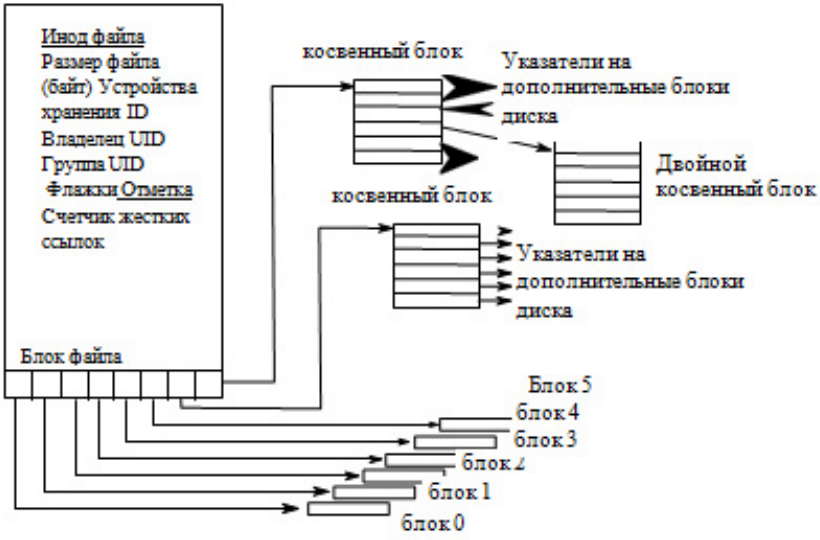


Рисунок 5.10
Иерархическая структура индексных дескрипторов в Linux.

Компьютерные Вирусы

Любое обсуждение файловой системы было бы неполным без обсуждения компьютерных вирусов и их опасности. Компьютерный вирус представляет собой программу, которая может копировать себя. Чаще всего, вред компьютерного вируса заключается в том, что его истинная цель состоит в том, чтобы привести к повреждению компьютера, на котором он хранится, или шпионить за пользователями этого компьютера.

Как вы можете заразиться? вирус прячется внутри другого исполняемого файла. Когда эта программа выполнена, код вируса выполняется. Код вируса, как правило, состоит из двух частей, первая должна копировать себя и сохранять свой собственный исполняемый код внутри других программ или файлов данных. Вторая часть является вредоносной частью.

Часто вирус ждет возникновения определенного условия (более старые вирусы могут ждать определенный день, например, как в пятницу 13-го, или время, или действие, например, пользователь пытается удалить данный файл) до того, как выполнится вредоносная часть. Вредоносная часть вируса может удалить данные или скопировать данные и отправить его третьей стороне (эти данные могут включать в себя информацию, которая хранится в cookies, такие как истории браузера и / или номера кредитных карт).

Сегодня слово "вирус" относится ко многим различным типам вредоносного программного обеспечения: программам-шпионам, рекламному ПО, троянским коням, червям, рут-китов. Тем не менее, термин вирус действительно должен применяться только к самовоспроизводящемуся программному обеспечению, которое не будет включать в себя программы-шпионы, рекламное ПО или троянских коней. Группа комбинированного программного обеспечения теперь называется вредоносной программой.

Как защитить себя? лучшим способом является избегание Интернета (включая электронную почту)! Но это непрактично. Вместо того, наличие последней актуальной программы антивирусной/защиты от вредоносных программ имеет большое значение. Также может помочь отключение cookies. Кроме того, не открывайте вложения от людей, которых вы не знаете. Также полезно иметь некоторый план аварийного восстановления. Один подход заключается в создании резервных копий вашей файловой системы - в случае плохого заражения, вы можете

переустановить операционную систему и восстановить файлы. Если вы являетесь пользователем Linux или Mac, вы можете чувствовать себя в безопасности от вредоносных программ. Большинство целей вредоносных программ являются машины Windows, так как Windows является самой распространенной ОС, но на самом деле вредоносные программы могут повлиять на любую платформу.

Жесткая связь, созданная командой ln без параметра -s, является указателем на индексный дескриптор файла, так же, как исходное имя в каталоге указывает на индексный дескриптор файла, в отличие от символической ссылки, которая просто указывает на имя файла в каталоге. Жесткие ссылки могут быть использованы только для ссылки на файлы в той же файловой системе. Ссылка через смонтированные файловые системы (разделы) должна быть символической ссылкой. Это ограничивает полезность жесткой ссылки. Жесткие ссылки не распространены, а символические ссылки очень распространены. Если вы делаете ls -l, вы увидите символические ссылки, перечисленные с типом файла (первая буква в перечислении) из 'l', чтобы указать мягкую ссылку, а с именем файла -> истинное местоположение. Имя файла - это имя, которое вы создали по ссылке, в то время как истинное расположение – это путь к файлу и имя файла, на который указывает ссылка.

Файловая система WINDOWS

Раскладка файловой системы Windows, проще, чем в Linux, что делает ее более сложной, когда вы должны что-то найти, потому что вы найдете гораздо больше файлов в любом одном каталоге, чем вы, как правило, находите в Linux. Раскладка для Windows была достаточно стабильной, начиная с Windows 95. Раздел C: примерно эквивалентен Linux / корневой директории. Имена A: и B: зарезервированы для гибких дисков (которые, как правило, не доступны на компьютерах, приобретенных в последние годы) и D: обычно присвоены оптическому приводу компьютера. Могут быть добавлены другие разбиения (или основной жесткий диск может быть разбит на несколько «дисков», каждый со своим собственной буквой).
Под коренной файловой системой (C:), файловая система разделена, по меньшей мере, на три каталога. Операционная система хранится в папке Windows. Вложенная папка этого каталога System32 содержит системные библиотеки и общие файлы, похожие на Linux /usr/lib. Большая часть

прикладного программного обеспечения находится в Program Files. В Windows 7 есть отдельный каталог C:\Program Files(x86) для разделения 64-битного программного обеспечения (Program Files) от старого 32-битного программного обеспечения (Program Files (x86)). В Linux, большая часть прикладного программного обеспечения находится в /usr. Каталоги пользователей, как правило, хранятся в C:\Users (в то время как в Linux, эти каталоги находятся в /home). Там может быть каталог C:\Temp, который похож на Linux каталог /tmp.

Большое отклонение в Windows и Linux происходит в пользовательских каталогах. В C:\Users, папки по умолчанию создаются для различных типов файлов, Рабочий стол, Загрузки, Избранное, Ссылки, Мои документы, Моя музыка, Мои картинки, Мое видео, Поиски. В Linux, по большей части, пользователь решает, где разместить эти элементы. В операционной системе Windows, конкретные папки устанавливаются программным обеспечением, если только они не были перезаписаны пользователем. Есть также мягкие ссылки, так что "Мои документы" на самом деле относится к C:\Users\Имя Пользователя\Мои документы. Предоставление этих каталогов по умолчанию и мягкие ссылки помогают защитить пользователя от необходимости запоминать, где были сохранены файлы, или от того, чтобы понять файловое пространство.

Многие из операций Windows, имеющие дело с файловой системой доступны через GUI программ. Некоторые из них находятся с помощью панели управления, другие доступны через утилиты операционной системы, такие как Norton Utilities и инструменты McAfee. Панель управления включает в себя инструменты для восстановления системы (которая в первую очередь восстанавливает настройки системы, но и может восстановить компоненты файловой системы), резервное копирование и восстановление файловой системы, а также опции папки для контроля отображения при просмотре содержимого папки. И, конечно же, программа файловый менеджер по умолчанию Проводник Windows. Существуют также доступные ДОС команды. Наиболее значимой командой ДОС является chkdsk, утилита, которая служит той же цели, что и fsck в Linux. Другие команды включают дефрагментацию (дефрагментации диска утилитой), diskpart, которая позволяет разметить диск или выполнить соответствующие административные службы на диске (например, назначив ему букву диска или атрибуты), и find, аналогичную программе find в Linux.

При рассмотрении файловой системы Windows, можно почувствовать, что она интуитивно проста в использовании. С другой стороны, подход

Windows, может чувствоваться чрезмерно ограничительного характера. Например, если пользователь желает создать папку CIT 130 для сбора различных файлов данных, связанных с этим курсом, пользователь Linux может создать /home/foxr/cit130. Под этим каталогом, пользователь может поместить все файлы в плоском пространстве, или может создать подкаталоги, например, Картинки, Документы, Поиск и Видео. В операционной системе Windows, пользователь должен создать папки cit130 под каждой из установленных папок для того, чтобы организовать все материалы, которые относятся к CIT 130, но материал распределяется по многочисленным папкам. Только если пользователь достаточно мудр, чтобы переопределить значение по умолчанию, то пользователь сможет установить папку cit130, в которой могут быть размещены все элементы.

Перемещение по системе файлов

Здесь мы посмотрим на конкретные Linux и ДОС команды для перемещения своих файловых систем. В следующих примерах мы видим команды ОС, результаты этих команд и некоторые комментарии, описывающие то, что делают операции. Мы сосредоточимся в первую очередь на Linux, которая является немного более сложной. Освоив Linux, будет проще освоить DOS. Предположим, что мы имеем структуру каталогов, подкаталогов и файлов на рисунке 5.11. Элементы в прямоугольниках представляют собой файлы, а все остальные элементы являются каталогами. Для ДОС, заменить home на users.

Linux

Ниже следует сеанс Linux (команды, введенные в оболочке Linux, и результаты), а также объяснения команд. Комментарии приведены ниже некоторых из команд. Команды Linux появляются после символа $, которые мы будем считать, является приглашением Linux.

Команда / Результат	Объяснение
$ PWD /home/foxr	печать рабочего каталога
$ cd CIT130	Перейти в подкаталог CIT130
$ ls HW LABS	перечислить содержимое

В этом каталоге есть два пункта, которые печатаются

$ ls - l	выполнить длинный список содержимого

drwxr-xr-x 2 foxr foxr 1024 Jan 20 03:41 HW
drwxr-xr-x 4 foxr foxr 4096 Jan 21 17:22 LABS

Длинный перечень содержит более подробную информацию, включая дату и время создания (в военном времени, так 17:22 вместо 5:22 р.м.). На данный момент мы концентрируемся только на первом элементе, который выглядит как случайная последовательность символов. Первоначальный 'd' указывает, что элемент является некоторым каталогом. Остальные буквы и дефис означают права доступа (разрешения). Мы рассмотрим эту тему в главе 6.

$ cd hw	переход в каталог hw
$ cp h1.txt /home/foxr/CIT130/LABS	Копирование файла h1.txt в каталог LABS

Эта команда использует абсолютный путь, то есть мы начинаем на корневом уровне Linux (первая /), чтобы указать каталог LABS.

CD ..	перейти на один уровень вверх (/ home/foxr/CIT130)
$ LABS mv /lab1.txt.	перемещение lab1.txt сюда

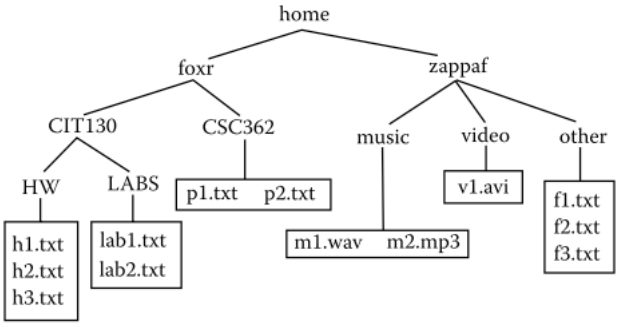


Рис.5.11
Схема для примеров в этом разделе.

. означает "этот каталог". Обратите внимание на то, что спецификация подкаталога LABS выполнена как относительный путь, который начинается в этой точке.

$ ls ../../CSC362	список содержимого каталога CSC362
p1.txt p2.txt	

../../CSC362 – это относительный путь, начиная с этого момента, но двигаясь вверх на два уровня (../..) и на один уровень вниз (CSC362).

$ cd ~/CSC362	перейти в ваш каталог CSC362
$ rm *. *	удалить все здесь

Символ ~ представляет домашний каталог текущего пользователя. Так что в данном случае, ~означает /home/foxr. *, используемый в rm. является подстановочным символом. Подстановочная * означает "ничего". Использование *.* означает, "все, что содержит период". Эта команда затем удалит все файлы, которые имеют период в их имени (файлы с расширениями, как файл p1.txt, а такие как foo не будут выбраны потому, что в нем отсутствует период).
Команда rm установлена выполнить rm -i, что означает "интерактивный режим". Она выведет список каждого пункта и спросит вас, уверены ли вы, что хотите удалить его, по одному за раз, например:

rm: удалить обычный file ‘p1.txt’? y
rm: удалить обычный file ‘p2.txt’? y
rm: удалить обычный file ‘lab1.txt’? n

Ответы, предоставленные пользователем, в данном случае приведет к удалению p1.txt и p2.txt, но не lab1.txt (который был перенесен сюда из каталога CIT130/LABS).

$ cd ..	Переместиться вверх до foxr
$ rmdir CSC362	удалить весь каталог CSC362
rmdir: CSC362 : Directory not empty	Каталог не пусто

Поскольку каталог не пуст, он не может быть удален. Мы могли бы удалить каталог, если бы мы сначала удалили оставшийся элемент lab1.

txt.
С другой стороны, мы могли бы выполнить команду rm -r CSC362. Опция -r означает "рекурсивное удаление". Рекурсивное удаление удаляет все содержимое каталога перед удалением каталога рекурсивно (так что, если каталог имел подкаталог, то сначала удаляется содержимое подкаталога, а затем содержимое каталога, включая сам подкаталог). Так, например, если мы выпустили rm -r ~foxr/CIT130, шаблон удалений будет действовать следующим образом: h1.txt, h2.txt, h3.txt, hw (каталог), h1.txt, lab2.txt (напомним, что ранее были скопированы h1.txt из каталога hw сюда и что мы переместили файл lab1.txt в другой каталог), LABS (каталог), и, наконец, каталог CIT130.
Опция -r может быть очень мощной, но и очень опасной. Лучше никогда не использовать rm -r, пока вы не делаете rm -ir. Часть -i опции заставляет искать подтверждение перед удалением чего-либо. Системному администратору особенно важно никогда не использовать rm -r, если вы абсолютно уверены в том, что вы делаете. Представьте себе, что вы выполнили cd / (перешли к верхней части файловой системы), а затем следует rm –r *.*. Вы хотите удалить все (в том числе саму программу rm)!
Примечание: cp также имеет рекурсивный режим, cp -r, чтобы рекурсивно скопировать все подкаталоги каталога, а mv не имеет рекурсивного режима.

$ cd /	переход к верхней части файловой системы
$ cd home	двигаться вниз к домашней директории
$ ls	перечислить содержимое
foxr zappaf	
$ cd zappaf	перейти в каталог zappaf
$ cp music /*.* ~	копировать файлы из музыки в ваш каталог

Опять же, *. * означает "все с периодом". Таким образом, эта команда копирует все файлы в музыкальном каталоге zappaf, которые имеют период в их имени в ~, что обозначает домашний каталог.

$ mv videos/*.* ~	переместить все

```
mv: не может переместить ‘/home/
zappaf/videos/v1.avi’
в ‘./v1.avi’:
Доступ запрещен
```

Хотя вы можете иметь разрешение на доступ к каталогу zappaf и содержимому, вы не имеете доступа перемещения, потому что перемещение по существу удаляет файлы из каталога этого пользователя. Мы будем исследовать разрешения в главе 6.

```
$ cd ~                          вернуться в свой домашний каталог
$ mkdir STUFF                   создать каталог с именем STUFF
$ ln -s video/v1.avi            создать символическую ссылку
STUFF/ video
```

Символическая ссылка хранится в подкаталоге STUFF (/home/foxr/ STUFF), b ссылка называется видео. Ссылка указывает на или ссылается на /home/zappaf/videos/v1.avi.

```
$ ls STUFF                      список содержимого каталога STUFF
video
```

Команда ls просто перечисляет элементы по имени, без предоставления каких-либо деталей. Если мы хотим более подробно, мы должны использовать опцию ls -l (для "длинного списка").

```
$ ls -l STUFF
lrwxrwxrwx 1 foxr foxr  16 Jan 21 17:13   video -> /home/zappaf/videos/
v1.avi
```

С длинного списка, мы видим, что видео является ссылкой на местоположение /home/zappaf/videos/v1.avi. Символ "l" в начале строки является типом файла (ссылка) и -> справа указывает, куда указывает ссылка.

ДОС

Мы ограничиваем наш взгляд на DOS команды, потому что они похожи и потому, что есть меньше вариантов. Здесь мы пройдемся лишь по части предыдущего примера.

Команда / Результат	**Объяснение**
C:	переключение на раздел/диск C:
C: \> CD Users \ foxr	перейти к каталогу foxr
C: \ Users \ foxr> dir	список содержимого каталогов

Обратите внимание на то, что при переключении каталогов, ваши приглашения изменяются. В списке текущего каталога нет эквивалента Linux команды pwd. Команда dir предоставляет содержимое каталога. Оно извлечено ниже. Некоторые подробности опущены.

Wed 11/23/2011 10:53 AM <DIR> .
Wed 11/23/2011 10:53 AM <DIR> ..
Wed 11/23/2011 10:55 AM <DIR> Desktop

Wed 11/23/2011 10:55 AM <DIR> Documents
Wed 11/23/2011 10:55 AM <DIR> Downloads
...
0 File(s)
15 Dir(s)

C: \ Users \ foxr> cd Documents	перейти к директории Documents
C:. \ Users \ foxr \ Documents> copy * * \ Users \ zappaf	скопировать все в директории zappaf 's
C: \ Users \ foxr \ Documents> cd ..	двигаться вверх на 1 уровень
C: \ Users \ foxr> cd Downloads	переместиться вниз в Downloads
C: \ Users \ foxr \ Downloads> move * * \ Users \ zappaf	Перемещение вместо копирования
C:. \ Users \ foxr> CD \ Users \ zappaf	Переключиться в папку zappaf

Обратите внимание на то, что в DOS не существует эквивалента ~, так что вы должны указать либо относительный или абсолютный путь, чтобы добраться до zappaf. Из \Users\foxr\ Downloads в zappaf, вы могли бы использовать этот относительный путь: CD ..\..\zappaf.

C: \ Users \ zappaf> dir	Список каталогов zappaf

Это теперь должно включать в себя все файлы, обнаруженные в Documents и Downloads для foxr.
Перечисление здесь опущено.

C: \ Users \ zappaf> del * *
C:. \ Users \ zappaf>

Обратите внимание на то, что имело место удаление. Вы не были лишены возможности удаления файлов в zappaf. Это потому, что соответствующие разрешения для защиты каталога zappaf и файлов не были созданы!
Мы закончим этот пример этими комментариями. Как и в Linux, для создания и удаления директорий, у вас есть команды mkdir и rmdir. Можно также создавать жесткие и мягкие ссылки, используя команду mklink. Как вы, возможно, заметили, символ * также работает в DOS. Команды DOS также допускают опции. Команда del может, вынуждена запрашивать у пользователя использование del/P и удалить без запроса с использованием /Q (тихий режим). В отличие от Linux, del не имеет рекурсивного режима.

Файловая система и задачи системного администрирования

Несмотря на то, что отдельные пользователи обслуживают свое собственное пространство файловой системы, существует множество задач, которыми может управлять системный администратор. Первая и наиболее важная задача, конечно же, это убедиться, что файловая система доступна. Это потребует монтирования системы по мере необходимости. Команды Linux mount и umount используются для монтирования нового раздела в файловой системе и размонтирования. Команды требуют, чтобы вы указывали как файловую систему, так и точку ее монтирования. Точка монтирования — это имя, с помощью которого пользователи будут ссылаться на нее. Например, если вы выполняете df –k, вы увидите различные точки монтирования, включая /var и /home. Команда mount -a установит все компоненты файловой системы, которые перечислены в специальном файле /etc/fstab (таблица файловой системы). Эта команда выполняется автоматически во время инициализации системы, но если вы должны размонтировать некоторые разделы, вы можете выполнить эту команду (как корневую), чтобы перемонтировать все. Файл /etc/fstab включает в себя дополнительную информацию, которая может оказаться полезной, например, дисковые квоты, и является ли данный

файл пространством только для чтения или для чтения и записи. Все установленные в настоящее время разделы также указаны в файле /etc/ fstab.

Настройка дисковых квот является сложной операцией, но может оказаться полезной в организациях, где есть много пользователей и ограниченное дисковое пространство. Вы можете установить политику использования дискового пространства (возможно с управлением) и осуществить ее, установив квоты на пользователей.

Другая политика использования может включать в себя, должны ли файлы быть зашифрованы или нет. Linux имеет ряд инструментов для шифрования с открытым исходным кодом, а также существует целый ряд инструментов публичных доменов, доступных для Windows.

Удаленное хранение означает, что часть файловой системы не является локальной для этого компьютера, но вместо этого доступна по сети. В целях поддержки удаленного хранения данных, необходимо расширить файловую систему, чтобы она была сетевой файловой системой. NFS является, пожалуй, наиболее распространенной формой этого, и он доступен в системах Unix/Linux. NFS является очень сложной . Вы можете найти подробную информацию о NFS в некоторых текстах, перечисленных для дополнительного чтения.

Эти последние темы, хотя и полезно, не будут охвачены в этом тексте, но должны быть охвачены в тексте системы администрирования Linux.

ГЛАВА 6. ПОЛЬЗОВАТЕЛИ, ГРУППЫ И ПРАВА ДОСТУПА

В главе 4 была введена идея защиты и безопасности. В целях поддержки этих операционных требований к системе, создаются учетные записи компьютеров. Каждая учетная запись поставляется с определенными правами доступа. Эти права могут быть изменены в операционных системах, таких как Windows, Linux и таким образом, что владелец ресурса может диктовать, кто может использовать этот ресурс. В этой главе рассматриваются учетные записи пользователей, создание учетных записей, групп и создание группы, пароли и права доступа.
Цели обучения данной главы являются:

• Описать роль учетной записи пользователя и группы.
• Ввести механизмы для создания и поддержания пользователей и групп учетных записей в Windows и Linux.
• Объяснить , как изменить права доступа к файлам и каталогов как в Windows и Linux.
• Обсудить управление паролями, включая концепцию сильного пароля.
• Ввести команду SudoLinux.

Напомним, что компьютерная система более чем компьютер; это совокупность устройств, программным обеспечением и пользователей. Если компьютерная система в вопросе должна использоваться более чем одним пользователем, то компьютерная система может предложить некоторую степень защиты. Защита гарантирует, что ресурсы, предоставленные одному пользователю, не мешают другим пользователям или их ресурсам . Ресурсы включают в себя процессы памяти хранения пользовательских, запущенные приложения, общие данные и файлы. Различные операционные системы предлагают различные подходы к защите. Здесь мы рассмотрим наиболее распространенный механизм защиты учетной записи пользователя. Наряду с установлением учетных записей пользователей, мы рассмотрим использование разрешений, так что пользователи могут контролировать доступ к своим ресурсам.
Помимо защиты, учетные записи полезны для ряда других целей.

Они обеспечивают безопасность в том, что только авторизованным пользователям разрешено использовать компьютер и все его доступные ресурсы. Они обеспечивают средства, с помощью которых мы можем собирать статистические данные о вычислении использования. Например, как системный администратор, вы можете проверить, как количество пользователей влияет на производительность процессора, есть ли достаточно основной памяти, и в какой степени пользователи используют файловое пространство . Это может, в свою очередь, обеспечить поддержку с просьбой управление тратить деньги на дополнительные ресурсы. Отслеживание использования компьютера поддерживает как мониторингпроизводительности, так и бухгалтерский учет.

Пользователь является агентом, который использует компьютер. Чаще всего, мы будем рассматривать людей как пользователям. Тем не менее, в некоторых ситуациях, пользователь может быть программным агентом . В большинстве операционных систем, пользователь требует учетной записи для использования компьютера. Учетная запись пользователя представляет собой набор данных, каталогов и привилегий, предоставляемых пользователю, созданной системным администратором.

Как правило, учетная запись пользователя будет связана с именем пользователя и паролем пользователя. Они используются для аутентификации пользователя, так что пользователю может быть предоставлен доступ к операционной системе. Аутентификация чаще всего осуществляется с помощью логина, который пользователь вводит как имя пользователя и пароль, которые затем сравниваются в базе данных, сохраненных самим пользователем имя/пароль. Некоторые более продвинутые системы могут соединяться с биометрическим или KEYCARD доступом, при котором пользователь должен провести свою карту черезкартридер или использовать некоторые биометрические, такие как голосовая печать или отпечатки пальцев. Они гораздо менее распространены. В некоторых случаях биометрическиеиспользуются вместо пароля.

Одной из ключевых проблем с надлежащей защитой компьютерной системы, чтобы пользователи могли использовать надежные пароли. Это довольно простой вопрос для хакера, чтобы разработать программу, которая будет проверять все слова из словаря, как пароли в попытке взлома учетной записи пользователя. Таким образом, сильные пароли являются те, которые не появляются в словаре. Требования к сильным

паролям , как правило, что они содержат, по меньшей мере, один не алфавитный символ или комбинацию прописных и строчных букв, и, по крайней мере, восемь символов в длину. Кроме того, сильные пароли могут требовать, чтобы пользователь изменял часто пароль (каждые 1 до 3 месяцев), без повторения предыдущих паролей. Другие ограничения могут включать в себя, чтобы пароль не повторял символы предыдущих паролей. Например, если ваш пароль был abcdef12, а затем изменить пароль к abcdef13 не может быть допустимо, так как он слишком похож на предыдущий пароль.

Многопользовательские системы требуют учетных записей пользователей, так что операционная система знает, кто текущий пользователь и может совпадать пользователя с соответствующими правами доступа. Система однопользовательское не требует каких-либо учетных записей пользователей, так как предполагается, что тот же пользователь всегда будет использовать эту систему. Сегодня, хотя большинство людей имеют свои собственные компьютеры, она все еще широко распространена для создания учетных записей пользователей по нескольким причинам. Во-первых, большинство операционных систем позволяют удаленный доступ и, следовательно, для защиты от других агентов от попыток использовать ресурсы одного пользователя, устанавливая логин, имеет важное значение. Во-вторых, все системы должны быть введены системным администратором. Даже если есть только один пользователь, который использует систему, необходимо проводить различие между пользователем, действующей в качестве конечного пользователя и действия пользователя в качестве администратора системы имеет решающее значение. Если пользователь хочет, например, удалить много файлов и случайно включает в себя системные файлы в список , то операционная система будет отклонять запрос, если текущий пользователь является обычным пользователем. Но если не было учетной записи пользователя, только учетная запись системного администратора, тогда операционная система подвергла бы сомнению продвижение удаления, возможно, к катастрофической ошибке пользователем.

Есть, по крайней мере, два типа учетных записей пользователей. Типичный пользователь является конечным пользователем, чья учетная запись дает ему доступ к своему собственному файловому пространству и разрешения на использование общих ресурсов, в том числе прикладного программного обеспечения. Их счета, как правило, не дают им доступ к системным ресурсам, в частности, программ системного

администрирования. Другой тип пользователя является системный администратор, известный как корень в Linux и Unix. Корень имеет доступ всвою - каждую программу , каждый файл, каждый ресурс, независимо от того, кто владелец этого ресурса есть. Некоторые операционные системы предоставляют учетные записи промежуточного уровня, где пользователь получает больше ресурсов, чем минимум, не будучи при этом на уровне системного администратора. В Linux и Unix, корень может разрешить другим пользователям иметь некоторые дополнительные права доступа. Это делается с помощью программы Sudo, скрытой в Разное Темы учетных записей пользователей. Как будет описано в разделе Настройка учетных записей пользователей, программному обеспечению также приведены учетные записи пользователей. В случае программного обеспечения учетных записей, доступ на самом деле более ограничен , чем обычные счета конечных пользователей, например, не имея домашнего каталога и/или не имеющих оболочки.

НАЛАДКА учетных записей пользователей

Здесь мы рассмотрим, как создавать учетные записи пользователей и в Linux и Windows. Обратите внимание, что только системный администратор сможет настроить учетные записи пользователей.

Linux

Первоначальная учетная запись пользователя создается при установке операционной системы Linux. Эта учетная запись является необходимой, так что первый пользователь системы не должен войти в графический интерфейс пользователя (ГИП) в качестве корня. Это может быть опасно, так как любое окно, чтобы пользователь имел привилегии суперпользователя, а не обычные пользовательские привилегии. Таким образом, к тому времени вы будете готовы использовать Linux, будет создана уже одна учетная запись пользователя. Оттуда, однако, вам придется создать другие.

В Linux существует два способа настройки учетных записей пользователей: через графический интерфейс и через командную строку. Оба подхода требуют, чтобы человек был системным Linux менеджер пользователем и 500, и студентов числа больше 500. В таком случае, следующий пользователь вы добавляете в ту же категорию, что и ранее добавленного пользователя и, следовательно, вам не хочется просто добавить 1 к номеру последнего пользователя, чтобы получить новый

номер.
На рисунке 6.1, вы можете увидеть, что четыре пользователя уже созданы. Например, первый пользователь имеет имя учетной записи foxr, номер пользователя (UID) из 500, Войти оболочки / bin / Баш, и домашний каталог / дома / foxr. Как системный администратор, вы можете создать политику учетных записей, таких как форма имен учетных записей (общая политика фамилия первая начальная или фамилия первая начальная и число, если это конкретная комбинация была использована, так что, например, Роберт Фокс может быть foxr1). Другие политики будут включать по умолчанию местоположение каталогов, по умолчанию оболочек, и может ли и в какой степени, чтобы указать полное имя человека. Опять же, в университетской среде, преподаватели каталоги могут быть размещены в одном месте и студенческих каталогов в другой. На рисунке 6.2 показано окно добавления нового пользователя. Большинство записей очевидны. Здесь новый пользователь Томми Марс и в настоящее время дано имя пользователя marst. Обратите внимание, что необходимо ввести начальный пароль пользователя и подтвердить его. Это обеспечивает соблюдение надлежащей безопасности таким образом, чтобы учетные записи не создавались без паролей. Тем не менее, это может быть хлопотно для системного администратора, который должен создать десятки или сотни счетов. Как мы увидим позже, подход командной строки к созданию учетных записей предпочтителен в таких случаях. Оболочка логина в систему и значение по умолчанию корневого каталога к/bin/bash и/home/username, соответственно. Системный администратор может изменить то, что появляется как по умолчанию. Кроме того, при входе нового пользователя, вы можете изменить их по мере необходимости. Вход оболочки также включает CSH, Ksh, SH, и TSCH

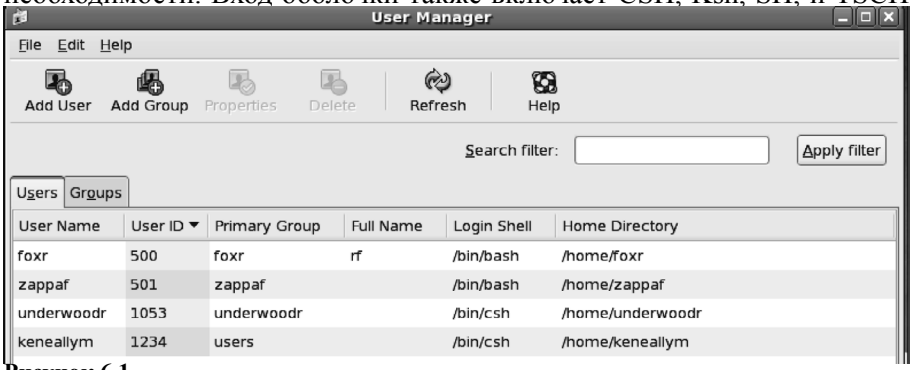


Рисунок 6.1

(они рассматриваются в главе 9), и вы можете также выбрать NOLOGIN, что означает, что пользователь не имеет оболочку входа в систему после входа в систему. Это часто определяется, когда пользователь является частью программного обеспечения. Вы также можете переопределить UID по умолчанию и задать другое значение по желанию. Например, если вы используете один набор чисел для преподавателей и другого для студентов, вы можете переопределить значение по умолчанию для нового студенческого счета.Один из пунктов на Рисунке 6.2 требует пояснения. Обратите внимание на второй флажок "Создать частную группу для пользователя". Когда учетная запись пользователя создается, группа, которая разделяет пользователя,
Пользователи, Группы и Привилегии
Создать новое окно пользователя.
имя учетной записи также создается. Например, пользователь foxr также будет иметь группу foxr. Частная группа будет представлять собой группу, которая содержит ровно один пользователь. Если включить этот флажок выключенным , учетная запись пользователя создается, но частная группа не создается. Мы обсуждаем группы в роли группы.

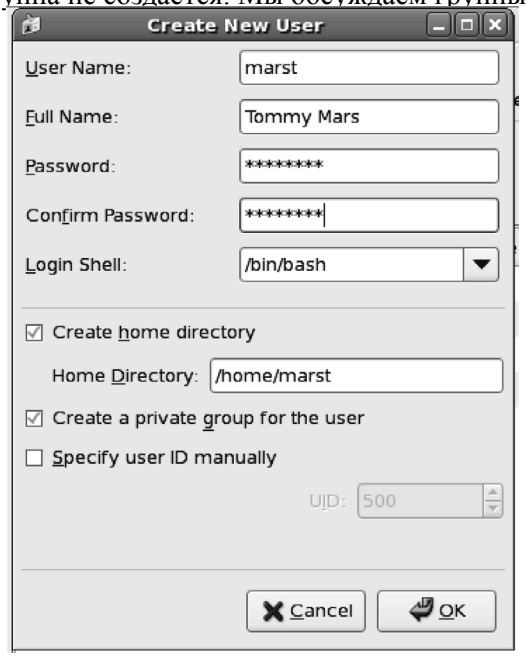


Рисунок 6.2

Удаление или изменение пользователя требуется выбрать пользователя из списка пользователей в окне диспетчера пользователей. После выбора, вы можете либо выбрать Свойства, чтобы изменить пользовательские свойства или Удалить пользователя, чтобы удалить пользователя. Вы можете изменить имя пользователя , полное имя, пароль, каталог или оболочку входа в систему. Можно также указать дату истечения срока действия учетной записи, пароль блокировки (так что пользователь не может изменить его), указать учетную запись expira-информацию (т.е. установить дату, к которой пользователь должен изменить свой пароль), и настроить группы, пользователя. Мы будем говорить об истечении сроков действия паролей в конце этого раздела. Если вы выберите команду Удалить, чтобы удалить пользователя, вы также спросили, является ли домашний каталог пользователя, почтовый файл и временные файлы должны быть удалены или нет. Необходимо сохранить эти каталоги и файлы для архивирования и / или в целях безопасности.

Хотя менеджер графического интерфейса пользователя прост в использовании, многие системные администраторы предпочтут использовать команду useraddвместо командной строки . Причина заключается в том, что сценарий оболочки можно записать, чтобы создать несколько новых учетных записей пользователей автоматически с помощью файла, который содержит список имен новых пользователей. Кроме того, useradd инструкция не требует спецификаций исходного пароля, таким образом, сокращая процесс. Конечно, без начального пароля, учетная запись пользователя может быть легко взломлена , и поэтому некоторый начальный пароль должен быть создан. Тем не менее, отдельный сценарий может позаботиться об этом. Инструкция пароля обсуждается позже.

Useradd инструкция проста в использовании, если вы хотите указать только несколько свойств, как и с графическим интерфейсом. Основной формой useradd является:

useradd [-uUid [-o]] [-g группа] [-G группа, ...] [-d главная] [-s оболочка] [-m] имя пользователя

Обозначения здесь требуют небольшого пояснения. Перечисленное внутри [] является необязательным . Таким образом, единственным обязательным параметром для useraddявляется имя пользователя. Самая простая форма этой команды будет выглядеть следующим образом:

useraddmarst

Если вы не указали какой-либо другой параметр в стороне от имени пользователя, идентификатор пользователя по умолчанию равно 1 больше последнего пользователя, каталог по умолчанию в / главная / имя пользователя, а оболочка по умолчанию оболочки по умолчанию (обычно Баш). -m параметр вынуждает Linux создать домашний каталог. Причина, почему это не является обязательным является то, что некоторые учетные записи не должны иметь домашние каталоги, особенно те, которые создаются для программного обеспечения. Программное обеспечение счета будет объяснено более подробно ниже.
Чтобы указать не по умолчанию идентификатор, используйте -u с последующим идентификационным номером. Параметр -o используется для разрешения, не уникальные идентификационные номера. Существует мало оснований, чтобы разрешить повторяющиеся идентификаторы, и это может стать проблемой. Тем не менее, вариант по-прежнему доступен и может быть использован, например, если вы хотите, чтобы пользователи использовали одну учетную запись, не разделяя один и тот же логин / пароль, или если вы хотите, чтобы дать пользователю несколько имен учетных записей. Такая команда может выглядеть следующим образом:

useradd -u 500 -ofoxr2

Он позволит создать второй счет, который будет использовать один и тот же UID, как foxr (из рисунка 6.1). Чтобы отменить частную группу по умолчанию, -g позволяет указать другую частную группу, тогда как -G позволяет вставить этот пользователь в ряд других групп. Вы можете заметить на рисунке 6.1, что keneallym был включен в группу пользователей по умолчанию, поскольку ему не была предоставлена своя собственная частная группа . Параметры -d и -s соответственно используются для указания домашних каталогов и оболочек, отличных по умолчанию. Два других параметров следует отметить -p, который позволяет определить начальный пароль, и -c, которая позволяет указать имя поля. -c Параметр используется для вставки "комментарий" в файл учетной записи пользователя, / и т.д. / пароль, но комментарий в основном используется для указания имени пользователя.
Рисунок 6.3 демонстрирует useradd команду, используя несколько параметров. В этом случае, мы добавляем пользователя Рут Андервуд. Ее Войти оболочка CSH (C-оболочки), а не Баш. Ее домашний каталог

по умолчанию. Мы даем ей идентификатор пользователя, группы и привилегии выходящего за пределы обычной последовательности, и назначение ее к нескольким группам. Символ # не является частью команды, но вместо того, чтобы подсказка для системы администратору Linux '(корень). Обратите внимание, что мы указываем полный путь к инструкции, потому что / USR / SBIN, вероятно, не в пути суперпользователя. Также обратите внимание, что пространства не размещены между группами (ударников, музыка, cit130).

Как добавить пользователя, информацию об учетной записи пользователя добавляется в файл / и т.д. / пароль. Информация, содержащаяся в этом файле, включает в себя для каждой учетной записи, имя учетной записи, номер удостоверения личности, группы идентификационный номер, имя (поле комментария), домашний каталог, и Логин оболочки. Группа идентификационный номер является идентификационный номер личного счета пользователя. Для реальных пользователей (людей), ID и частных номеров групп, как правило, одинаковы.

Интересно отметить, что отсутствует в / и т.д. / пароль файла пароль пользователя. В прошлом этот файл содержит все пользовательские пароли, которые хранятся в зашифрованном виде. Но / и т.д. / пароль файл может быть прочитан кем-либо. Это создало угрозу безопасности. Несмотря на то, пароли в зашифрованном виде, можно было видеть длину пароля. Например, если мой пароль был длинный с восемью символами, и я заметил, что кого-то есть зашифрованный пароль такой же длины, как мой зашифрованный пароль, я могу сделать вывод, что их пароль был также из восьми символов, и это может помочь мне взломать пароль. Теперь, / и т.д. / пароль файл содержит символ 'x' пароли и все

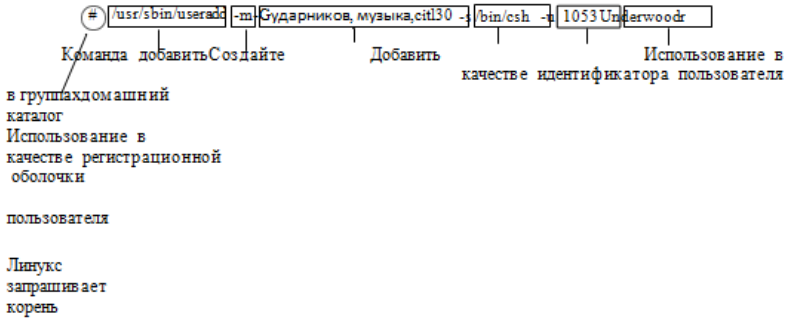


Рис.6.3

Linux пользователь команду добавить.

пароли хранятся в файле / и т.д. / тень. Теневой файл доступен только корень.

На рисунке 6.4 показаны участки и т.д. / пароль файл /. Этот список показывает три сегмента файла паролей. Счета, перечисленные на рисунке, операционной системы счетов, других учетных записей программного обеспечения и учетных записей пользователей. Операционная система счета включает корень (естественно, самый важный счет), BIN, демон, адм и другие учетные записи программного обеспечения Linux, такие как почта, новости, uccp, FTP. Прикладное программное обеспечение счета включает Apache , например. Каждая учетная запись в файле паролей находится на отдельной строке, и в пределах линии, отдельные части информации для каждого пользователя отделены друг от друга двоеточием.

Большая часть программного обеспечения счетов указывает / SBIN / NOLOGIN, чтобы препятствовать тому, чтобы у программного обеспечения была оболочка входа в систему (это делается в целях

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
hsqldb:x:96:96::/var/lib/hsqldb:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
foxr:x:500:500:rf:/home/foxr:/bin/bash
zappaf:x:501:501::/home/zappaf:/bin/bash
underwoodr:x:1053:1053::/home/underwoodr:/bin/csh
keneallym:x:1234:100::/home/keneallym:/bin/csh
marst:x:1235:1235:Tommy_Mars:/home/marst:/bin/bash
```

Рис. 6.4.

Ниже приводятся выдержки из / и т.д. / пароль файл.

безопасности), и есть другие, чем в директории /Главная. Например, корневого каталога является / корень; Л.П., почта, и Apache имеют каталоги под / Вар; и новости есть каталог в / и т.д.. В каталоге / вар в основном используется для хранения программного обеспечения и файлов журналов. Также обратите внимание на то, что все программное обеспечение счетов имеет UIDs меньше, чем 100 (ниже 42 в данном случае), в то время как пользователи человека номера счетов начинаются 500. В каждом случае пароль является лишь x, что указывает на то, что пароль в настоящее время находится в / и т.д. / тени

Все записи также имеют поле комментария. В некоторых случаях поле комментария перечисляет лишь имя пользователя (Root, bin, daemon) и в других случаях истинное имя (FTP пользователя для FTP, NSCDDaemon для ДКНБ, X сервер шрифтов для XFS и ВЧ для foxr). Пользователи zappaf, underwoodr и keneallymимеют пустое поле комментария, которое появляется, как ::, то есть, ничегоне происходит между двумя символами двоеточия.
Записи в / и т.д. / теневого файла несколько загадочно, потому что они зашифрованы. Например, два из пользователей на рисунке 6.4 (foxr и zappaf) могут иметь записи следующим образом:

foxr: $ 1 $ KJlIWAtJ $ aE1YjCp9i5j924vlUXx.V:. 15342: 0: 38: 31 ::: zappaf: $ 1 $ G2nb6Kml $ 2pWqFx8EHC5LNypVC8KFf0: 15366: 0: 14: 7 :::

Это списки имен пользователя, за которым следует зашифрованный пароль. После этого следует последовательность информации об истечении срока действия пароля. В приведенном выше примере, есть четыре номера, но не может быть больше, чем шесть. Эти цифры в следующем порядке:

• Количество дней, прошедших с 1 января 1970 года, что пароль был изменен в последний раз
• Минимальное количество дней, необходимых между изменениями пароля (это может быть 7, если, например, мы требуем, чтобы пароли не менять чаще, чем раз в неделю)
• Максимальное количество дней, что пароль действителен (99999 по умолчанию, здесь пароль foxr действителен в течение более 38 дней, в то время как zappaf действителен только в течение более 14 дней)
• Число дней до предупреждения выдается для предупреждения

пользователя о том, что необходимо изменить пароль
• Количество дней после истечения срока действия пароля, когда учетная запись будет отключена
• Количество дней, прошедших с 1 января 1970 года, когда учетная запись была отключена.

Эти последние две цифры не появляются в приведенном выше примере, потому что они не были созданы для этих двух пользователей.
То, что мы видим здесь, что foxr изменил свой пароль в какой-то момент в недавнем прошлом и zappaf не сделал, поэтому пароль zappaf истекает раньше, чем foxr годов. Через 7 дней, zappaf будет предупрежден для сброса пароля. Вы можете увидеть с помощью простого вычитания (14 - 7 для zappaf и 38 - 31 для foxr), что система предупреждает пользователя 7 дней до истечения срока действия пароля. Если пользователь не сбрасывает пароль, учетная запись будет отключена.
В Linux, команда PASSWD используется для изменения паролей. Для обычного пользователя, команда будет просто PASSWD. Затем пользователю будет предложено их текущий пароль, чтобы убедиться, что пользователь может изменить пароль (представьте себе, если вы отошли от вашего компьютера и кто-то подошел и попытался изменить свой пароль без вашего ведома!). А затем новый пароль, ввести дважды, чтобы подтвердить новый пароль. Программа PASSWDсообщит вам, является ли пароль неверный пароль, потому что это слишком короткий или из-за героев матча пароля или приблизится к соответствию слова из словаря.
Как системный администратор, вы можете изменить PASSWD любого пользователя с именем команды passwd , например, passwd foxr. Как корень, вы не обязаны вводить токовый пароль пользователя перед изменением. Это может быть полезно для создания первоначальных паролей или для изменения массовых паролей из-за изменения политики. Но она также может быть очень опасна , так как вы можете изменить чей-то пароль непреднамеренно, и если вы вводите команду PASSWD без имени пользователя, вы измените корневой пароль.
PASSWD команда имеет много полезных параметров для системного администратора. С их помощью администратор может создавать сроки смены паролей. -fпараметр заставляет пользователя изменить пароль при следующем входе в систему. С -w дней (где дни является положительным целым числом), поле предупреждения устанавливается таким образом, что пользователь получает предупреждение о том, что

много дней, чтобы изменить свой пароль. С -x макс, максимальное поле установлено, что устанавливаетряд дней, что пароль остается в силе до того, как должно быть изменено. В сочетании вы можете установить, что пользователь должен изменить пароль в некоторое количество дней и быть предупрежденным об этом до этого. Например, PASSWD -w 7 -x 30 foxr потребует foxr изменить свой пароль в течение 30 дней, и если пароль не изменяется в течение следующих 23 дней, foxr получает предупреждение. -lпараметр блокирует ввод пароля, так что он не может быть изменен.

Взлом

Термин хакерство предназначен для передачи программиста, кто-то, кто "взламывает " код. На самом деле, хакерство имеет несколько определений. Другое применяется к любителям, которые любят экспериментировать с чем-то новым. Но многие люди считают, что хакер, как человек, который пытается проникнуть в компьютеры путем угадывания паролей пользователей (более правильно, это называется крекинг). Есть много способов для взлома аккаунта пользователя. Мы посмотрим на некоторые из них здесь.

Взлом пароля: перебор различных паролей, а также его отсутствие (только клавиша ввода), словарь слов, имя пользователя или имя супруга пользователя, дети, домашние животные, родители и т.д.

Социальная инженерия: пытается получить пользователь, чтобы дать вам свой пароль, например, путем обращения по телефону и сказал: «Это Боб от ИТ, наш сервер разбился, и для того, чтобы восстановить свою учетную запись, мне нужен пароль." Это также называется фишинг.
Лазание по мусоркам : многие люди пишут свои пароли на стикерах. Быстрый взгляд вокруг рабочего места человека или их мусорка может дать пароль!
IP подмены: изменив свой IP-адрес, вы можете быть в состоянии перехватить Интернет-сообщения, предназначенные для кого-то другого. Вполне возможно (хотя и не очень вероятно), что пароли могут быть найдены таким образом.

Команда usermond используется для изменения учетной записи пользователя. Вы можете использовать эту команду, чтобы изменить

комментарий пользователя, домашний каталог, начальная группа, другие группы (смотрите раздел роль группы), Логин оболочки, идентификатор пользователя, пароль, дата окончания срока действия учетной записи, а также количество дней после истечения срока действия пароля до того, как учетная запись окончательно вышла из строя. Два других параметра позволяют заблокировать или разблокировать пароль пользователя.
Как уже упоминалось ранее, человеческие пользователи и программное обеспечение - приведенные счета. Вы можете спросить, почему программное обеспечение предоставляет отчет. Причина в том,что из-за использования разрешений для управления доступом к файлам и каталогам. Об этом рассказывается в Права доступа. Рассмотрим, однако, что часть программного обеспечения, нужно будет читать и записывать файлы в определенной директории. Мы хотим убедиться , что другие пользователи не могут писать в этот каталог или эти файлы в целях безопасности (мы также хотим ограничить других пользователей от чтения каталога и файлов). Когда пользователь запускает процесс, процесс принадлежит этому пользователю. И поэтому, если файл не был собственностью этого пользователя, то файл не может быть для чтения или записи к этой программе. Если вместо этого, программа имеет свою собственную учетную запись, которая является владельцем файлов, то программа может получить доступ к этим файлам, даже если пользователь, который запустил программу, не может. Таким образом, в Linux существует два типа учетных записей пользователей, для тех человеческих пользователей и тех, для программного обеспечения. Обычно, как было описано ранее, что программные учетные записи не имеют оболочку входа в систему (так как программа никогда не входит) и не может иметь домашний каталог.

Windows

Ранние версии Windows были предназначены для систем с одним пользователем, так что не было никакой необходимости создавать учетные записи пользователей. В WindowsNT, сетевой версии Windows, это изменилось. Теперь учетные записи включены в каждой версии Windows. Как и в случае Linux, начальная учетная запись пользователя создается во время установки операционной системы таким образом, чтобы пользователь компьютера мог войти как без прав администратора. Хотя начальная учетная запись пользователя не требует пароль, настоятельно рекомендуется, чтобы учетная запись имела пароль (и

аналогично для всех будущих счетов созданы).
Чтобы создать дополнительные учетные записи, вы должны действовать в качестве администратора. Это можно сделать через окно учетных записей пользователей, через панель управления. Функция учетной записи пользователя становится доступной через графический интерфейс, как показано на рисунке 6.5.
Подобно LinuxUserManager, у вас есть возможность добавлять, удалять или изменять свойства пользователей. Свойства здесь, однако, ограничены размещения группы. По умолчанию, существует только одна группа администраторов. Тем не менее, вы можете создать дополнительные группы по мере необходимости. На рисунке 6.5 есть три счета: системный администратор учетной записи (администратора); foxr, который также является администратором; и heggeju, кто находится в группе HelpLibraryUpdaters. Эта учетная запись используется членом IT для автоматического обновления системы компьютера по мере необходимости. На вкладке Дополнительно в верхней части окна

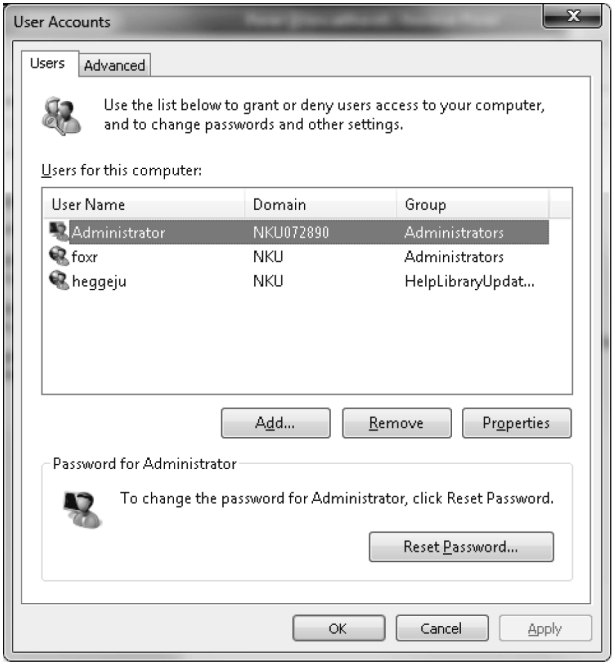


Рис. 6.5
учетная запись пользователя в Windows

UserAccount позволяет одним изменять пароли, добавлять группы и изменять членство в группах. Вполне вероятно, что, если вы работаете в Windows 7 на домашнем компьютере, что вы будете иметь два счета: Administrator и учетной записи, созданной при установке операционной системы. Если имеется несколько пользователей, которые разделяют компьютер, вы можете или не желаете , чтобы дать каждому пользователю отдельный счет. Имея отдельные счета, вы создаете не только пространство файла для каждого отдельного пользователя (под C: \ Users директории), но вы также можете установить права доступа к файлам, так что разные пользователи могут получить доступ к различным файлам и запускать различные программы. Если вы имеете дело с машиной Windows 7, который является общим для пользователей в рабочей среде, то вполне вероятно, что каждый пользователь имеет учетную запись в домене организации (домены не охвачены в этом тексте), и вы можете добавить учетную запись для них на любой из компьютеров в домене.
Пользователи также могут быть добавлены с помощью средства управления компьютером. Щелкните правой кнопкой мыши на значке компьютера ярлык и выберите Управление. С помощью инструмента

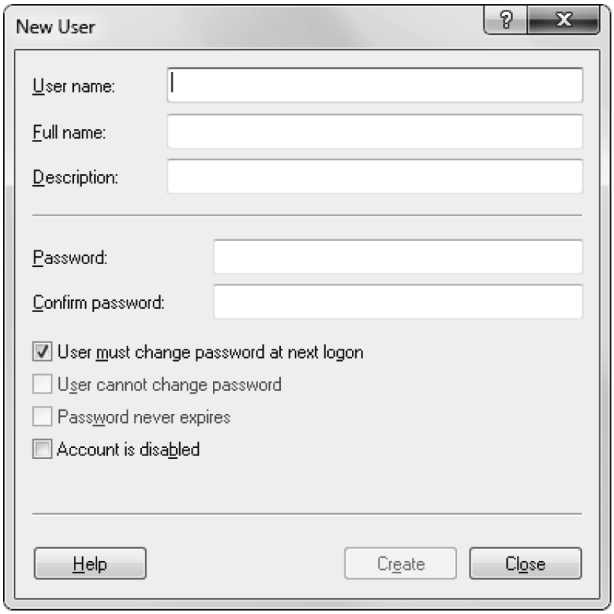


Рис. 6.6
Новое окно пользователя.

управления компьютером, выберите SystemTools и затем Локальные пользователи и группы. Щелкните правой кнопкой мыши на значок Пользователи и выберите Новый пользователь. Появится окно Новый пользователь, как показано на рисунке 6.6. Заметьте здесь, что слово ПАРОЛЬ требуется, в отличие от добавления нового пользователя через окно на рисунке 6.5.

РОЛЬ ГРУППЫ

Отдельный пользователь создает и управляет файлами. Часто желательно, чтобы поделиться файлами с другими пользователями. Тем не менее, есть некоторые файлы, где доступ должен быть ограничен определенным пользователям. Возможность контролировать, кто может получить доступ к файлу и какие типы прав доступа этих пользователей предоставляются, осуществляется с помощью разрешений. В Linux, например, права доступа могут быть установлены для трех различных классов пользователей: владелец файла, тем пользователям, которые разделяют ту же группу, как владелец файла, а весь остальной мир. Это требует, чтобы у нас был объект, называемый группой. И Linux, и Windows, использовать группы таким образом, что доступ не должен ограничиваться лишь владельцем файла и остальным миром .
Группа является всего лишь имя (и номер ID) вместе со списком пользователей, которые являются частью этой группы. Файл может принадлежать как пользователю, так и группе. Права доступа могут быть установлены , так что владелец имеет определенные права доступа, а также члены группы имеют разные права доступа. Таким образом, к примеру, владелец может быть в состоянии читать и писать файл, в то время как члены группы могут только читать файл и все остальные пользователи не имеют доступа к файлу.

И Linux и Windows позволяют создавать группы. В Linux, создание группы можно осуществить с помощью графического интерфейса диспетчера пользователей (смотрите Рисунок 6.1, используется вкладка Group) или через GroupAdd инструкции из командной строки. Как и с пользователями, в группе приведены идентификационные номера. При создании группы, как добавление пользователя, вы можете указать идентификационный номер группы, или он может быть по умолчанию 1 больше последней созданной группы . В операционной системе Windows, создание группы осуществляется через окно ComputerManager. После

того, как группа создана , отдельные пользователи могут быть добавлены к группе.

В Linux, пользователи могут быть добавлены или удалены из групп с помощью инструмента для управления пользователями или из командной строки. Для добавления пользователя в существующую группу, вы можете использовать либо потребительный Радд с параметром -G или usermod (изменить пользователя). Примером может служить usermod -Gcit130 zappaf. Это добавит zappaf к группе cit130. Информация о группе хранится в файле / и т.д. / группы. Этот файл хранит для каждой группы название группы, группы идентификационный номер, пароль для группы (как и с / и т.д. / пароль, это просто появляется как «x»), а также список имен пользователей, которые принадлежат к этой группе. Можно редактировать этот файл, чтобы изменить членство в группах, хотя безопаснее использовать инструкцию usermod. В операционной системе Windows, пользователи могут быть добавлены или удалены из групп с помощью компьютера диспетчера пользователей и групп выбора, как показано на рисунке 6.7.

В Linux, при создании новой учетной записи пользователя, вы можете указать, должен ли быть создан частный счет для этого пользователя. Как правило, это случается. Создание закрытой группы вызывает новую группу, имя которого равно имени пользователя, чтобы быть созданным и добавленным к / и т.д. / файл группы. Единственный член этой группы

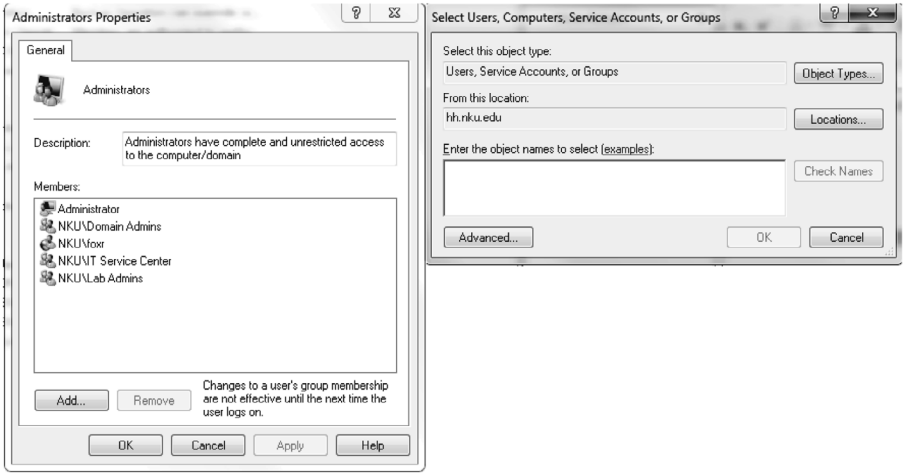


Рисунок 6.7

Добавление (или удаление) пользователей группам в Windows.

будет пользователь. Например, там будет группа под названием foxr с одним членом, foxr и группу под названием zappaf с одним членом, zappaf. Любой файл, созданный пользователем, по умолчанию, принадлежит частной группе этого пользователя. Это до владельца файла для изменения группы владельца. Если foxr должны были создать файл f1.txt, она первоначально будет принадлежать пользователю foxr и группе foxr. Если foxr хотел разделить файл с членами группы cit130, foxr придется изменить группу собственности. Команда ЧаунLinux позволяет владельцу файла изменить владельца на другой пользователь, и команда chgrp позволяет владельцу файла для изменения владельца группы в другую.
Эти две Linux команды работают следующим образом:

Чаунновый_пользователь имя файла команда chgrpновая_группа имя файла

Например, если у вас есть файл foo.txt принадлежит корню, и как корень вы хотите изменить его будет принадлежать foxr, команда будет Чаунfoxrfoo.txt. Вы можете изменить владельца группы, а также, например, в качестве команды chgrpcit130 foo.txt. Обратите внимание на то, что эта команда изменяет группу не foxr своей частной группы, но cit130, предположительно группа из более чем одного пользователя.
Если вы хотите изменить владельца и группу, это может быть сделано с помощью Чаун в одиночку:

Чаунновый_пользователь: новая_группа имя файла

Если вы не являетесь владельцем файла, вы не смогли бы изменить пользователя или группы владельца. Тем не менее, если вы корень, вы можете изменить владельца или группу любого файла.

Списки управления доступом (ACL) используются в некоторых операционных системах для реализации защиты системных ресурсов. Для каждого ресурса есть список, который описывает для каждого пользователя в системе, что их права доступа для этого ресурса. Пользователи, не имеющие доступа, не будут перечислены. Например, какой-то файл f1.txt может иметь ACL из foxr: чтение, запись, исполнение; zappaf: чтение, выполнение; underwoodr: читать. Преимущество ACL является то, что вы можете указать различные права доступа для каждого

пользователя. К недостаткам можно отнести, что списки управления доступом требуют пространства для хранения, которое растет с каждым пользователем, который вы добавляете в список, и довольно трудоемкий характер изменения прав доступа . Например, представим, что новый пользователь, dukeg, добавляется к системе. Мы решили дать dukeg доступ на чтение к каждому файлу в системе. Добавление dukeg: чтение к каждому ACL, если не автоматизирован через какой-то сценарий, может быть неприятной обязанностью для администратора .

Linux упрощает использование ACL, имея три типа разрешений чтения, записи и выполнения, применительно к трем классам пользователей: владелец файла, группы файла, и остальной мир . Эти разрешения размещены на каждой файловой системы и директории. Пользователи имеют возможность изменять некоторые или все разрешения от их значений по умолчанию при создании файла / каталога.

Если файл читается, его можно просмотреть или скопировать. Если файл записываемый, он может быть записан, перезаписан (с изменениями), или удален . Если файл является исполняемым, он может быть выполнен. Как правило, мы бы не беспокоились о выполняемости из nonprograms. Скрипты оболочки являются исключением. Сценарий оболочки не выполняется непосредственно; он вместо того, выполняется интерпретатором. Это должно означать, что сценарий оболочки должен быть доступен для чтения только интерпретатором, но, на самом деле, оно должно быть как для чтения и выполнения. Переводчик не будет являться владельцем файла, ни в группе файла, так что потребуется чтение и выполнение для остальной части мира.

Права доступа к каталогам имеют несколько иное значение, чем права доступа файлов. Если каталог для чтения, то вы можете просматривать его содержимое (то есть сделать Ls). Если каталог не доступен для чтения, вы можете не только не просматривать его содержимое, но вы не сможете просматривать содержимое отдельных файлов в этом каталоге. Если каталог доступен для записи, то вы можете сохранять файлы в этот каталог, а также копировать или перемещать файлы в этот каталог. Если каталог исполняемый файл, то вы можете перейти в него. Если каталог были читаемыми, но не исполняемыми вами, то хотя вы могли бы использовать Ls в каталоге, вы не сможете в него компакт-диск.

Три класса пользователей прав доступа Linux являются владелец ресурса, группа ресурса, и все остальные в мире, которые могут войти в машину Linux. Мы будем ссылаться на них как пользователь (и), группа (г), и другие (о). К сожалению, это немного сбивает с толку, потому что

владелец представлен 'U', а не 'o'. Для каждого из этих трех уровней, есть три права возможные POS-доступа, которые могут быть поставлены: чтение (r), запись (w) и выполнение (x). Эти письма имеют важное значение, как мы увидим ниже. Файлы и каталоги будут по умолчанию определенными правами. Для того, чтобы изменить разрешения, вы должны иметь файл (или быть корень) и использовать команду CHMOD (смена режима). Об этом рассказывается в подразделе Linux ниже.
В операционной системе Windows, доступ к файлам немного отличается. Права доступа являются полный контроль, изменение, чтение и выполнение, чтение, запись и специальные разрешения . Полный контроль дается владельцу, который затем имеет возможность изменять права доступа других лиц. Изменить означает, что файл может быть записан поверх, то есть, команда сохранения будет работать. Чтение и выполнение будут применяться к исполняемому файлу, в то время как чтение будет применяться к невыполняемому . Запись позволяет сохранить заново через сохранить как, или пересохранить . Любой пользователь или группа могут дать любое сочетание вышеуказанных прав доступа. Права доступа могут быть установлены в "Разрешить" или "Запретить". Мы посмотрим на изменение прав доступа Windows, в подразделе Windows.

Linux

Как уже говорилось, Linux имеет три уровня разрешения: владелец (и для пользователя), члены группы (г), и весь остальной мир (о для других). Каждый уровень может быть установлен в любой комбинации для чтения (R), перезаписываемый (w), и исполняемый (x). Вы можете просмотреть разрешения файла (или каталога) при использовании команды -lLs. Опция -l означает "длинную форму". Возмущенные миссии представлены в виде набора из девяти символов, показывая для владельца, если читаемый установлен, доступен для записи, исполняемый файл устанавливается, и то же самое для группы и мира. Например, файл, указанный в качестве rwxr-x, - x означает, что чтение, запись и исполняемый владельцем, считываемые и выполняемые членами группы, и исполняемый только миром. - указывает на то, что конкретное разрешение не установлено. При выполнении LS -l, этот список из девяти символов предшествует десятый символ, который представляет тип файла. Как правило, это либо дефис (файл) или 'd' (каталог), и, как мы видели в главе 5, буква "л" означает ссылку.

На рисунке 6.8 показан типичный длинный список. Список включает в себя, помимо прав доступа, владельца и группы файла, размер файла, дата последнего изменения и имя файла. Обратите внимание, что ~ в конце некоторых имен файлов указывает на то, что это резервный файл. При изменении файла с использованием некоторых редакторов (например, Emacs), старый файл по-прежнему сохраняется, но помечен как старый файл с ~ характер. Поэтому foo2.txt более новая версия и foo2.txt ~ это более старая версия. Сохранение foo2.txt снова будет копировать foo2.txt в foo2.txt ~, чтобы освободить место для новой версии.

Давайте посмотрим на пару пунктов, на рисунке 6.8, чтобы получить лучшее понимание информации, предоставленной длинный список . Во-первых, для настольных ПК имеет права доступа к "drwxr-xt-x". 'D' указывает на то, что это является каталогом. Каталог принадлежит foxr и в частной группе foxr. Foxr пользователь имеет RWX разрешения (чтение, запись, выполнение). Поэтому foxr может компакт-диск в каталог, каталог LS, читать файлы из каталога (предполагая, что эти файлы могут быть прочитаны владельцем), запись в каталог,и пересохранить файлы в каталог (опять же, предполагая, эти файлы могут быть перезаписаны владельцем). Другие пользователи, будь то в частной группе foxr (в которых не было бы, вероятно, не один) или остальную часть мира для чтения и выполнения доступ к этому каталогу, но не доступ на запись. Все остальные элементы, показанные на рисунке 6.8, представляют собой файлы, поскольку их разрешения начинаются с символа '-'. Файл foo2.txt для чтения и записи foxr только. Ни один другой пользователь (кроме корня) не умеет читать и писать этот файл. Файл foo3.txt можно читать и писать, и доступен для записи foxr и пользователей в группе foxr (опять же, нет, наверное, никто иной, чем foxr). Оно предназначено только для чтения по остальному миру .

Для изменения прав доступа, вы должны либо быть владельцем файла или администратором системы. Команда изменение прав доступа является CHMOD (смена режима). Команда получает два параметра: изменения разрешения и имя файла. Изменение разрешения может быть сделано одним из трех способов.

Во-первых, вы можете указать, какие изменения вы хотите, используя обозначения, как это: U + W, G-x. Первая буква в каждой группе представляет тип пользователя: пользовательский (U), группа (г), другие (О), напомнив, что пользователь означает владелец и другие средства, остальная часть мира . Знак плюс указывает на то, что вы добавляете разрешение, а знак минус указывает на то, что вы удаляете разрешение

. Вторая буква в группе – разрешение, добавленное или удаленное. В приведенном выше примере, владелец добавляет writability к себе и удаление executability из группы. Вы можете перечислить несколько изменений разрешений для каждого типа пользователей, таких как через и-Wx, г-RWX. Это позволит удалить как запись и выполнение с правами владельца и удалить все разрешения (RWX) из группы.
Во-вторых, вы можете указать весь набор разрешений, используя буквы (и, г, о, г, ж, х), но назначая разрешения через знак равенства вместо плюс и / или минус. Например, мы могли бы использовать и = RWX, г = гх, О = х, чтобы дать файлу все разрешения для владельца, чтения и выполнения для группы, и выполнения только для мира. Чтобы не указывать разрешения на уровне,вы сразу следуйте знака равенства с запятой , или оставьте это поле пустым для о

```
drwxr-xr-x 2 foxr foxr 1024 Aug 23  2011 Desktop
-rw------- 1 foxr foxr  138 Aug 24  2011 foo2.txt
-rw------- 1 foxr foxr  138 Jun  4 09:09 foo2.txt~
-rw-rw-r-- 1 foxr foxr   20 Jun  4 09:11 foo3.txt
-rw-rw-r-- 1 foxr foxr  276 Jun  4 09:11 foo.txt
-rw-rw-r-- 1 foxr foxr  138 Jun  4 09:10 foo.txt~
-rw-r--r-- 1 foxr foxr   17 Oct  3  2011 script1
```

Длинный листинг в Linux.

(другие). Например, у = RW, г =, о = даст владельцу чтение и запись, и группа и мир не будут иметь никаких разрешений.
Можно комбинировать первый и второй методы до тех пор, пока они соответствовали друг другу. Вы бы, как правило, использовали первый метод, если вы хотите немного изменить разрешения, а второй метод, если вы хотите сбросить разрешения на что-то совершенно другое (или если вы не знали, что разрешения были ранее). Следует отметить, что для обоих этих подходов, буква "а" может быть использована для обозначения всех пользователей (и, г, о). Так, например, + R означает, что каждый получает доступ для чтения, или а = х означает, что все перезагружены, чтобы выполнять доступ.
Третий подход аналогичен второму подходу в том, что вы заменяете все права доступа с новым набором; Однако, в этом случае вы указываете права как трехзначное число. Три цифры будут соответствовать правам, предоставленных пользователю, группе, и другим , соответственно. Цифра будет находиться в диапазоне от 0 до 7 в зависимости от того, какую комбинацию чтения, записи и выполнения вы хотите назначить. Для того, чтобы вывести цифру, подумайте о том, как разрешение файла,

которое появляется с использованием обозначения rwxrwxrwx. Измените каждую группу из 3 (RWX) на три двоичных битов, где бит равен 1, если вы хотите, что право доступа предоставляется и 0, если вы не хотите, что право доступа предоставляется. Например, rwxr-x - x переписываем в виде 111 101 001. Теперь, конвертировать каждый из этих 3-разрядных чисел в десятичной системе (или восьмеричной): 111 7, 101 5, 001 1, что дает нам 751. Таким образом, трехзначное число сопоставления прав доступа rwxr-x - x 751.
Другой способ просмотра 3-оцененный номер, чтобы просто присвоить значения 4 для удобства чтения, 2 для writability и 1 для executability. Теперь сложить значения, которые вы хотите для каждого владельца, группы, и мир, чтобы создать свой 3-оцененный номер. Если вы хотите совместить чтение, запись и исполняемый файл, то это будет 4 + 2 + 1 = 7. Если вы хотите разрешить только для чтения и записи, это будет 4 + 2 = 6. Если вы хотите разрешить только для чтения и исполняемый файл, это будет 4 + 1 = 5. Если вы хотите читаемость только, это будет 4. И так далее. Вы вычислите сумму для каждого владельца, группы и мира. Таким образом, значение 640 будет для чтения и записи для владельца, читаемым для группы, и ничего для мира. Значение 754 означает, что владелец будет иметь все три вида доступа (г, ш, х), группа бы чтение и выполнение, и мир был бы только для чтения доступ.
Ниже приведены некоторые другие общие ценности, вы могли бы найти для файлов и каталогов:

660-чтения и записи для владельца и группы, ничего для мира 755-чтение, запись, исполнение для владельца, чтение и выполнение для группы и мира 711-чтение, запись и выполнение для владельца, выполнение для группы и мира 444-а только для чтения файлов для всех
400-доступ на чтение для владельца, ничего для группы и мира
000-нет доступа к сюрпризам (только системный администратор не может получить доступ в такой файл).

Ниже приведены некоторые значения, которые вы, вероятно, не видите для файлов и каталогов:

557-нет никаких причин, почему мир будет иметь больший доступ, чем владелец 220-а "только для записи" файл не имеет никакого смысла, не имея возможности открыть или прочитать файл, 420-этот файл

считывается только для владельца, но писать только для группы
777-хотя это допустимое разрешение, оно опасно, чтобы дать доступ к мировой записи.

Кроме того, это редко, чтобы не сделать директорию исполняемого файла, в то время как неисполняемые файлы редко даются выполнить доступ. Таким образом, мы склонны видеть 755 или 775 для каталогов и 644 или 664 для файлов.
Каждый из CHMOD, Чаун, и команда chgrpмогут изменять несколько файлов одновременно, перечисляя каждый файл. Например, Чаунzappaf foo1.txt foo2.txt foo3.txt, сменит владельца всех трех файлов пользователя zappaf.
Linux также может быть повышено с помощью функции, называемой Security-EnhancedLinux (SELinux), что позволяет разрешения , которые будут выделены гораздо более тонким зернистым образом. SELinux идет выше, и за разрешениями, обсуждавшихся до сих пор и включает в себя использование списков ACL, чтобы позволить гораздо более высокий уровень контроля и безопасности. SELinux поддерживает полностью отдельный набор пользователей и ролей, чем те, которые мы обсуждали ранее . SELinux гораздо более сложно , чем стандартные права доступа к файлам Linux, а также детали его использования выходят за рамки этого текста.

Windows

Операционная система Windows использует аналогичную идею для разрешения как в Linux. Изменение разрешений не так просто (по крайней мере, когда у вас есть большое количество файлов для изменения) и не обеспечивает достаточно количество гибкости. Чтобы просмотреть уровень разрешения на файл, из файлового менеджера, щелкните правой кнопкой мыши на значок файла и выберите Свойства. Из окна свойств выберите вкладку Безопасность. Вы увидите что-то вроде того, что показано на рисунке 6.9.
На вкладке безопасности на свойства файла, вы увидите различные группы, которые имеют доступ к этому файлу. Они будут включать в себя систему, администраторов и пользователей. В этом случае мы также видим авторизированных пользователей. Это примерно эквивалентно системе администратора и мира. Авторизованные пользователи могут представлять собой группу. Права доступа ограничены полным

контролем, изменение, чтение и выполнение, чтение, запись и специальные. Если вы даете доступ пользователю полный контроль, то вы допускаете пользователю изменять права доступа других. Изменение прав доступа требует, прежде всего, что вы выберите кнопку Редактировать. Из нового всплывающего окна, вы можете добавлять или удалять разрешения, нажав на флажки. Вы можете также добавлять или удалять отдельных пользователей и групп из списка. Таким образом, Windows дает возможность добавлять разрешения для определенных пользователей без необходимости иметь дело с созданием группы. Смотрите Рисунок 6.10.

Тематическое направление РАЗНОЕ учетных записей пользователей

Рассмотрим следующую ситуацию. Вы в настоящее время вошли всвойWindows 7 машину . Вы редактируете несколько файлов. Одному

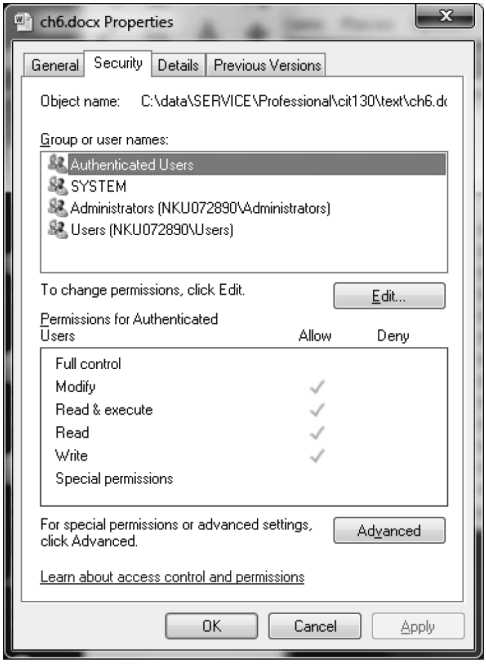


Рисунок 6.9
Вкладка Безопасность на изменение прав доступа.

из членов вашей семьи необходимо быстро получить доступ к файлу собственной и отправить по электронной почте другу. Поскольку этот файл принадлежит этому члену семьи, и они обеспечили разрешения, должен быть видим только их собственный счет, вы не можете просто позволить им использовать компьютер в течение минуты. Вместо этого вам придется сохранить файлы, выйти из системы, пусть они войдут, редактировать свой файл, отправить по электронной почте, выйти из системы, а затем вам придется снова войти в систему и вновь открыть свои файлы. Эта задача может занять несколько минут. Вместо этого Windows 7 имеет пользовательский переключатель объекта (эта функция впервые

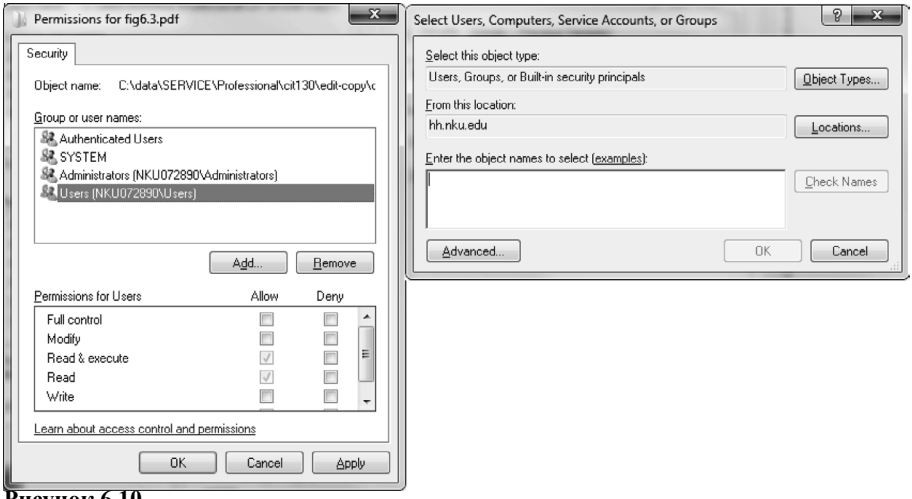


Рисунок 6.10

Изменение прав доступа (слева) и добавление или удаление пользователей и групп (справа).

Рисунок 6.11

Переключение пользователей.

была включена в WindowsXP и доступна в Vista). Для переключения пользователя, перейдите на кнопку Пуск, выберите в меню Завершение работы и выберите пользователя Switch (Рисунок 6.11). Это позволяет прервать свою собственную сессию, пусть ваш член семьи войдет и сделает свою работу, а затем вернется к вашей учетной записи. Когда вы вернетесь к вашей учетной записи, вы найдете ее именно так, как вы его оставили, все ваши окна еще открыт, и ваших программ, работающих под управлением. Это, вероятно, лучше, чтобы сохранить свою работу перед переключением пользователей, однако, в случае, если член вашей семьи случайно или по ошибке закрывает компьютер вниз, а не выход из системы. Объект пользователя коммутатор не сохраняет какой-либо из ваших файлов.

В Linux, переключение пользователей осуществляется гораздо более простым способом. Если вы работаете в окне оболочки, используется при выполнении команды су имя пользователя. Вам будет предложено войти в систему как новый пользователь. Хотя вы теперь новый пользователь с точки зрения прав доступа, что интересно, вы будете по-прежнему в любом рабочем каталоге, где вы были до выдачи команды су. Таким образом, вы можете следить за командой су с компакт-диском ~ команда служит для перехода в домашний каталог нового пользователя. Для выхода из нового сеанса оболочки, типа выхода и вы вернетесь к вашему предыдущему входу в систему (и предыдущий каталог). Если вы выполните команду су без имени пользователя, вы будете пытаться переключиться на корень учетной записи системного администратора. Обратите внимание, что если вы в настоящее время корень, вы можете переключиться на любого пользователя без необходимости вводить свой пароль. Как уже было отмечено ранее, каждая операционная система имеет специальный счет, системный администратор (администраторы в ОС Windows, в корне Linux и Unix, а иногда называют суперпользователем в других операционных системах). Эта учетная запись имеет доступ к любой системе ресурса. В операционной системе Windows есть набор программ и инструментов администратора, которые недоступны для обычных пользователей. В Linux, многие программы администратора доступны только для корня, и они часто встречаются либо в / SBIN или / USR / SBIN. Среди этих программ useradd, userdel, usermod, GroupAdd, groupdel и groupmod (все это в / USR / SBIN). Если вы делаете -lLs на эти файлы, вы увидите, что все они имеют разрешения rwxr-x ---, что означает, что только владелец и члены группы могут читать или выполнять эти программы. Владелец корень и группа является корнем,

поэтому только корень может запустить их. Есть и другие системные программы, которые являются исполняемыми только корнем в обоих из этих каталогов.
Рассмотрим в Linux, которые вы могли бы хотеть, чтобы обеспечить некоторую форму разрешения на подмножество пользователей с помощью GroupAdd. Таким образом, вы хотите создать группу, добавить подмножество пользователей к этой группе, а затем изменить разрешения на файл, давая тем пользователям доступ. Но если вы посмотрите на GroupAdd программы (находится в каталоге / USR / SBIN), вы увидите, что он принадлежит корню и в частной группе суперпользователя. Кроме того, его разрешения "-rxwr-x ---" означает, что он не доступен для среднего пользователя Linux (только корень или кто-то в частной группе суперпользователя). Тем не менее, как системный администратор, вы не хотели бы поощрять людей, чтобы сделать свои файлы открытыми миру в таких случаях, и не кажется, что будет еще один вариант, если пользователь не сможет убедить системного администратора использовать GroupAdd, чтобы создать новую группу . Linux действительно предлагает другой подход, хотя это можно было бы считать компромиссом между тем, чтобы вынуждать пользователей устанавливать опасные разрешения, и наличие системного администратора создавать группы по запросу. Такой подход, чтобы разрешить одному или выбрать группу пользователей возможность запуска конкретных программ системы. Это делается через Sudo программы и файл с именем / и т.д. / sudoers.
Для того чтобы использовать Sudo, существует два этапа. Во-первых, как корень, редактировать файл sudoers. Несмотря на то, что файл уже содержит некоторое количество записей, вы бы добавили свой собственный, вероятно, в нижней части файла. Запись имеет вид:

Имя пользователя (ей) хост = команда

Там может быть более одного пользователя в списке, если вы хотите дать такой же доступ к нескольким пользователям. Usernames будут разделены запятыми. Можно также указать членов группы с помощью

% Или имя_группы% GID (идентификационный номер группы).

Значение для хоста будет либо localhost, всем,либо список имен (компьютер) хоста. Слово localhost означает машина, на которой хранится sudoers файл. Это ограничивает пользователей к тому, чтобы войти в

игру, к этой машине и ограничивает команду, которая будет выполнена только на этой машине. Слово все используется для обозначения любого компьютера в сети. Команда может быть списком команд, разделенных пробелами. Команда должна включать в себя полный путь к файлу, например,

/ SBIN / монтирование или / USR / SBIN / useradd.

Вторым шагом является выдача команды Sudo. Команда Sudo это слово Sudocопровождаемое фактической командой, которую пользователь хочет выдать. Например, если вы получили sudoer привилегию работать useradd, то вы бы выполнили эту операцию

Суда / USR / SBIN / useradd -d / главная / cit130 -Gcit130, студенты -mnewguy

Здесь, в дополнение к команде пользователя добавить (который определяет другое местоположение домашней директории и добавляет нового пользователя к двум существующим группам), команда предшествует Sudo. Программа Sudo, прежде чем пытаться выполнить команду, необходимо, чтобы пользователь мог ввести свой пароль. Если пароль совпадает, то программа Sudo сравнивает имя пользователя и команду записи в / и т.д. / файл sudoers. Если совпадение найдено, то команда выполняется, как если бы пользователь был на самом деле корень; В противном случае пользователь получает сообщение об ошибке, например:

foxr не в файле sudoers. Этот случай будет сообщен .

Доступ к Sudo не должен быть предоставлен по наитию. На самом деле, это не должно быть предоставлено вообще, если пользователь не обучен и организация обеспечивает политику, которая четко определяет роли и требования предоставления определенного доступа системного администрирования. Одно общее исключение из этого правила является следующая запись sudoer:

% пользователей LocalHost = / SBIN / выключение -нтеперь

Команда отключения отключает машину. Эта команда выдается GUI,

когда пользователь выбирает завершение работы. Таким образом, все пользователи должны иметь возможность выключить компьютер с помощью графического интерфейса пользователя или в командной строке. Группа пользователей представляет собой группу всех пользователей, автоматически корректируется всякий раз, когда пользователи будут добавлены в систему.

Разрешение учетных записей пользователей, адаптированные среды, известные как оболочки в Linux, часто упоминается как рабочий стол в Windows. Информация, описывающая подогнанную среду, хранится в домашнем каталоге пользователя. После входа в систему, файл (ы) будет загружен в среде оболочки или GUI. Таким образом, пользователь может создать среду для того, как вещи выглядят, где вещи будут располагаться, и ярлыки для выполнения команд. В операционной системе Windows, этот файл или профиль автоматически сохраняется, как пользователь настраивает рабочий стол. В Linux существует несколько различных файлов, которые могут хранить информацию о профиле. Например, если пользователь использует Bash оболочки в качестве оболочки, пользователь, скорее всего, отредактирует файл .bashrc, хранящийся в их домашнем каталоге. Мы будем изучать Bash более подробно в главе 9.

ГЛАВА 7.
ИСТОРИЯ КОМПЬЮТЕРОВ

Большинство людей принимают компьютеры сегодня как должное, даже не замечая их влияния на общество. Для того, чтобы получить более глубокое понимание нашего цифрового мира, в этой главе рассматривается история вычислительной техники: аппаратных средств, программного обеспечения и пользователей. Эта глава начинается с самых ранних принимающих устройств , затем переходит к развитию электроники, цифровых технологий и компьютеров. С 1940-х годов, компьютеры делятся на четыре "поколения ", каждое из которых изучено . Эволюция компьютерных программ предусматривает три пересекающиеся темы: изменения в программировании, изменения в операционных системах, изменения прикладного программного обеспечения. Глава также кратко рассматривает , как изменились пользователи компьютеров, и эволюционировало аппаратное и программное обеспечение.
Целями обучения данной главы являются:

• Сравнить компьютерное оборудование между четырьмя компьютерными поколениями.
• Описать влияние, которое имели интегральные схемы и миниатюризация на эволюцию компьютерной техники.
• Описать изменения в программировании от языков низкого уровня до языков высокого уровня, включая концепции структурного программирования и объектно-ориентированного программирования.
• Объяснить роль операционной системы и изучить достижения в области программного обеспечения.
• Обсудить изменения в обществе с 1950 года по отношению к использованию компьютера и компьютерных пользователей.

В истории человечества, только некоторые изобретения оказывали такое же влияние на общество, как компьютеры. Возможно сам язык, способность генерировать и использовать электричество, автомобили имели схожее влияние, как компьютеры. В случае автомобиля, воздействие было более значительным в Северной Америке (в частности,

в Соединенных Штатах), чем в других странах, которые полагаются на общественный транспорт, такие как поезда и метро. Но без сомнения, компьютер оказал огромное влияние на большую часть человечества, а также влияние имело место в течение более короткого промежутка времени, чем другие изобретения, так как только крупномасштабные использования компьютеравосходят , возможно, 25-30 лет. В дополнение к быстрому воздействию, которые имели компьютеры, это, конечно, так, что ни один другой человек, ни новшество не улучшилось так резко , как компьютер. Рассмотрим следующие простые сравнения компьютеров с 1950-х годов до их современных коллег:

• Компьютеры 1950-х годов стоят сотни тысяч долларов, в то время как сегодня компьютер можно найти дешево за $ 300 и "простой" компьютер будет стоить не более $ 1200.
• Компьютер 1950-х годов может выполнять тысячи операций в секунду, в то время как сегодня количество команд составляет около 1 млрд. (рост по крайней мере, миллион).
• В компьютерах 1950-х годов было мало оперативной памяти, возможно, несколько тысяч байт; сегодня, основная емкость памяти составляет, по меньшей мере, 4 ГБ (увеличение, возможно, миллион).
• Компьютер 1950-х годов будет занимать значительное пространство, полный большой зал с постройками , и весят несколько тонн; сегодня, компьютер может поместиться в вашем кармане (смартфон), хотя более мощный компьютер общего назначения может поместиться в портфеле и не будет весить не больше, чем несколько фунтов.
• Компьютер 1950-х годов будет использоваться не более чем на несколько человек (возможно, десятков), и там было всего несколько сотен компьютеров, так что общее число людей, которые использовали компьютеры, было несколько тысяч; Сегодня число пользователей в миллиарды.

Если бы автомобили прогрессировали , как компьютеры, мы имели бы автомобили, которые могли бы ускоряться от 0 до миллиона миль в час менее чем за секунду, они получили бы миллионы миль на галлон бензина, они будут достаточно малы, чтобы упаковать и взять с вами, когда вы достигли места назначения, и вместо того, обслуживание вашего автомобиля вы бы просто заменили его на новый.
В этой главе мы рассмотрим, как компьютеры изменились, и как эти изменения повлияли на наше общество. Мы рассмотрим четыре

области изменения: компьютерной техники, программного обеспечения компьютера, пользователей компьютеров, а также общие изменения в нашем обществе. Существует отдельная глава, которая охватывает историю операционных систем. Краткая история Интернета освещена в главе по сетям, несмотря на то, что мы кратко рассмотрим в этой главе, как Интернет изменил наше общество.

Эволюция компьютерной техники

Мы ссылаемся на различные типы эволюции компьютерной техники с точки зрения поколения . Первое поколение произошло примерно с середины 1940-х годов и конца 1950-х годов . Второе поколение не произошло от около 1959 до 1965 года. Затем продолжалась третья генерация до начала 1970-х годов. Мы были в четвертом поколении до сих пор. Перед тем, как рассмотреть эти поколения , давайте кратко рассмотрим историю вычислений перед компьютером.

До Поколения

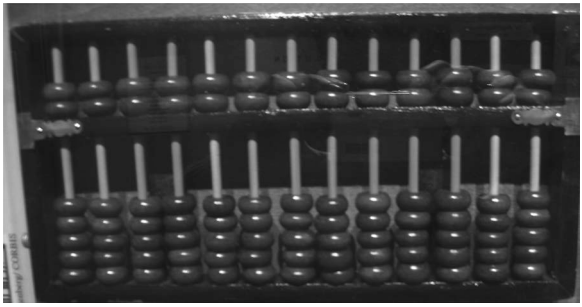
Без сомнения, самая ранняя форма вычислений были пальцы рук и ног людей. Мы используем десятичное, скорее всего, потому что мы имеем 10 пальцев. Чтобы подсчитать, люди могли бы задержать некоторое количество пальцев. Конечно, большинство людей было мало еще в 2000 лет до н.э. или даже 1000 лет до н.э., чтобы рассчитывать. Возможно, древние пастухи подсчитывали своих овец, и матери подсчитывали своих детей . Очень немногие имели много имущества на сумму счета, и математика была очень элементарной . Тем не менее, были и те, кто хотел рассчитывать за 10 или 20, так что кто-то изобрел счетное устройство под названием Abacus (смотрите рисунок 7.1). Бусы представляют собой количество элементов, подлежащих учету. В этом случае Abacus использует базу 5 (пять бусин, чтобы скользить в колонке); Тем не менее, с двумя бортами в верхней части колонны, можно либо рассчитывать 0-4 или 5-9, так что на самом деле, каждый столбец представляет собой силу 10. Абак может иметь три отдельные области для хранения разных чисел гой, например: первый и третий номера могут варьироваться от 0 до 9999, а среднее число может быть от 0 до 99999. Мы не уверены в том, кто изобрел арифметику, или как давно это было изобретено, но это, безусловно, было около тысячи лет назад.
Сама математика была очень сложной на рубеже тысячелетия между

BC и AD, из-за использования римских цифр. Сделайте следующее арифметическую задачу: 42 + 18. В римских цифрах , было бы написано, как XLII + XVIII. Не очень легкая задача решить в этом формате, потому что вы не можете просто выстраивать в очередь число и добавить цифры в столбцах, как мы учили в начальной школе.

Он не был до эпохи Возрождения в Европе, что математика начала продвигаться вперед, а вместе с ней , стремление к автоматизированной вычислительной технике . Только одно, что позволило совершенствование математике, это было новшество арабской системы нумерации (использование цифр 0-9), а не римские цифры. Ренессанс был также период улучшения образования с наличием мест обучения (университетов) и книги.В 1600-е годы, математики ввели такие новые понятия, как алгебра, десятичная система счисления, тригонометрия, геометрия и математический анализ.

В 1642 году французский математик Блез Паскаль изобрел первый калькулятор, устройство под названием Паскалин. Устройство работает таким же образом, как часы. В часах , зубчатое колесо вращается, перемещается с помощью маятника. Шестерня соединяется с другим зубчатым колесом другого размера. Полный оборот одной шестерни приводит следующую передачу, чтобы переместить на одну позицию. Таким образом, первая шестерня будет контролировать "секундную стрелку" часов , следующая передача будет контролировать

fIGurE7.1 An abacus. (Courtesy of Jrpvaldi, http://commons.wikimedia.org/wiki/File:Science_ museum_030.jpg.)



"минутную стрелку" на часах, а другой механизм будет контролировать "часовую стрелку" тактовых импульсов. Смотрите рисунок 7.2, в котором показано, как разного размера шестерни могут соединиться вместе. Для

механического калькулятора, Паскаль сделал два изменения. Во-первых, механизм превратится, когда предыдущий механизм был повернут на один полный оборот, который был бы 10 позиций вместо 60 (за несколько секунд или минут). Во-вторых, шестерни будут перемещены человеческой рукой, а не качанием маятника. Вращающиеся шестерни в одном направлении будут выполнять дополнения, и вращающиеся шестерни в противоположном направлении будут выполнять вычитания . Например, если механизм уже установлен в положении 8, а затем повернуть его на 5 позиций по часовой стрелке, вызовет его до конца в положении 3, но это прошло 0, так что следующая шестерня сместится на одну позицию (от 0 до 1), в результате чего калькулятор со значениями 1 и 3 (8 + 5 = 13). Вычитание может быть выполнено путем поворота в направлении против часовой стрелки.

В 1672 году немецкий математик и логик Лейбниц расширил способность автоматизированного калькулятора для выполнения операции умножения и деления. Калькулятор Лейбница, как и Паскаля, будет использовать вращающиеся механизмы для представления чисел, в результате чего один механизм вращающихся последние 10 вызовет следующую передачу, чтобы повернуть. Тем не менее, Лейбницдобавил дополнительные места хранения (зубчатые колеса), чтобы представить, как много дополнений или вычитаний для выполнения. Таким образом,

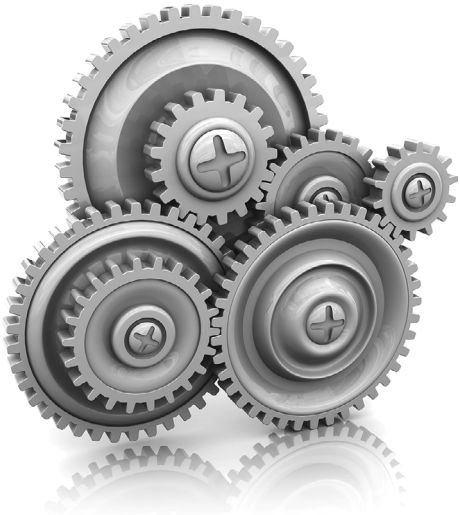
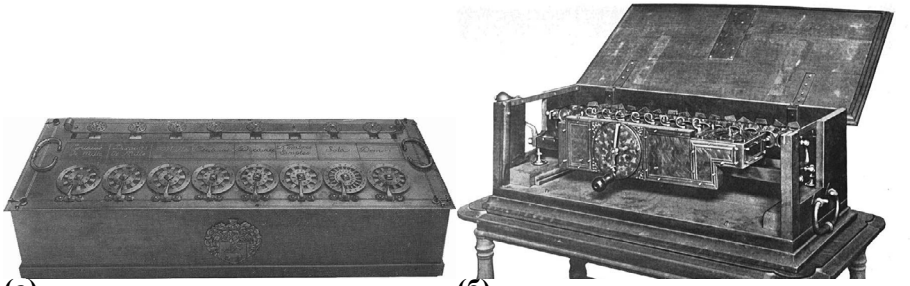


Рисунок 7.2
Механизмы часов. (Предоставлено Шаттерсток / mmaxer.)

то же самое количество может быть добавлено вместе несколько раз, чтобы создать умножение (например, 5 * 4 всего 5 суммироваться 4 раза). Рисунок 7.3 показывает, как Паскаля (а) и (б) калькуляторов Лейбница. Обратите внимание на рукоятку руки в версии Лейбница упростить количество усилий пользователя, а не превращать отдельные шестерни, как и с Паскалин.

В 1801 году мастер-ткач Жаккард изобрел программируемый ткацкий станок. Ткацкий станок представляет собой механическое устройство, которое позволяет потокам быть легко переплетенными . Некоторые из нитей подняты, и поперечная нить проходит под ними (но выше других потоков). Поднятие потоков может быть трудоемким процессом. Крючки используются, чтобы поднять некоторый выбор нитей. Жаккардовый автоматизированный процесс повышения темы с помощью перфокарт. Перфокарты обозначали бы, какие крюки используются при наличии отверстий в карты. Таким образом, ткач может питаться в серии карт, один за один проход нити по всей длине объекта тканый. Значение этого ткацкого станка является то, что он был первым программируемым устройством. "Программа" осуществляет просто данные, которые диктуют, какие крючки активны, но идея автоматизации изменений и с помощью перфокарт для выполнения "программы" инструкции приведет к разработке более сложных программируемых механических устройств. Рисунок 7.4 иллюстрирует Жаккард около конца 1800-х годов. Обратите внимание на коллекцию перфокарт, которые составляют программу, или шаблон проектирования.



(а) **(б)**

Рисунок 7.3

(а) калькулятор Паскаля (развертка Ezrdr взяты из J.A.V.Turck, 1921, Происхождение современных вычислительных машин, западного общества инженеров, стр. 10. с разрешения.) и (б) калькулятор расчета Лейбница. (Сканирование по Chetvorno, взятой из J.A.V.Turck, 1921, Происхождение современных вычислительных машин, западного общества инженеров, стр.133. с разрешения.)

В дополнение к новой теории математики и инновационной технологии, идея двоичной системе нумерации и бинарные операции были введены в 1854 году, когда математик Джордж Буль изобрел двузначную логику, теперь известный как булева логика.[1] Несмотря на то, что это не будет иметь прямое воздействие на механической основе вычислений, это, в конечном счете, иметь большое влияние на вычисления .

В начале 1800-х годов, математик Чарльз Бэббидж рассматривал таблицу логарифмов. Математические таблицы были вручную сделаны группами математиков. Бэббидж знал, что эта таблица будет иметь ошибки, потому

7.4 Фигурный программируемый ткацкий станок Жаккарда . (Mahlum, фотография из Технологического Норвежский музей, Осло, 2008, http://commons.wikimedia.org/wiki/File:Jacquard_loom.jpg~~HEAD=pobj.)

1 Как и большинство открытий в области математики, работа Буля стала продолжением работы других математиков по логике, включая Уильяма Стэнли Джевонса, Августа де Морган, и ЧарльзаСандерса Пирса.

что логарифмы были вычислены вручную через утомительные ряда математических уравнений. Эти таблицы были вычислены с помощью новой идеи, разностных уравнений. Разностные уравнения состоят исключительно из сложения и вычитания, но каждое вычисление может быть весьма активное участие. Бэббидж ударил на идее, что с помощью автоматизированных калькуляторов , можно было бы, возможно, запрограммировать калькулятор для выполнения необходимых вычислений и даже распечатать финальный стол с помощью печатного станка форму вывода. Таким образом, Бэббидж разработал то, что он назвал разностная машина. Было бы, как и любой компьютер, выполнение входа (для приема данных), обработка (разница уравнений), хранение (ориентация различных передач будет представлять номера, используемые в вычислениях), а выход (окончательный набор передач будет иметь цифры на них, которые могут быть напечатаны путем добавления чернил). В 1822 году он начал развивать Разностную машину для вычисления полиномиальных функций. Машины будут паровые . Он получил финансирование от английского правительства по заказу £ 17000 в течение 10 лет.

К 1830 , Бэббидж пересмотрел свои попытки построить разностную машину, когда он пришел к превосходной идее, программируемому устройству общего назначения. В то время как разница двигателя может быть запрограммирована для выполнения одного типа вычислений, его новое устройство будет применяться к большим различным математическим задачам. Он назвал новое устройство как Аналитическая машина. Как разностная машина, аналитическая машина будет использовать перфокарты для ввода. Вход будет включать в себя как данные, так и саму программу. Программа будет состоять не только из математических уравнений, нои ветвление операций, так что производительность программы будет отличаться в зависимости от условий. Таким образом, новое устройство может принимать решения путем выполнения цикла (итерации) операции и операторы выбора. Он попросил, чтобы такой же математик, леди Ада Лавлейс, написал программы для нового устройства. Среди ее первой была программа, чтобы вычислить последовательность чисел Бернулли. Лавлейс закончил свою первую программу в 1842 году, и она теперь считается первым в мире программист . К сожалению, Бэббидж никогда не заканчивал ни один двигатель, оставшийся без денег. Тем не менее, Разностные машины и Аналитические машины были построены с помощью конструкции Бэббиджа. На самом деле, в 1991 году, студенты в Великобритании

построили аналитическую машину, используя компоненты, доступные в 1830-х годах. Стоимость была около 10 миллионов $. Рисунок 7.5 представляет собой чертеж разностной машины на выставке в Музее науки в Лондоне.

"Компьютер" Бэббиджа работает в десятичной системе счисления, так же, как и предыдущие устройства калькулятора и был механический характер: физические компоненты движущихся были использованы для расчета. Шестерня поворачивается для выполнения сложения и

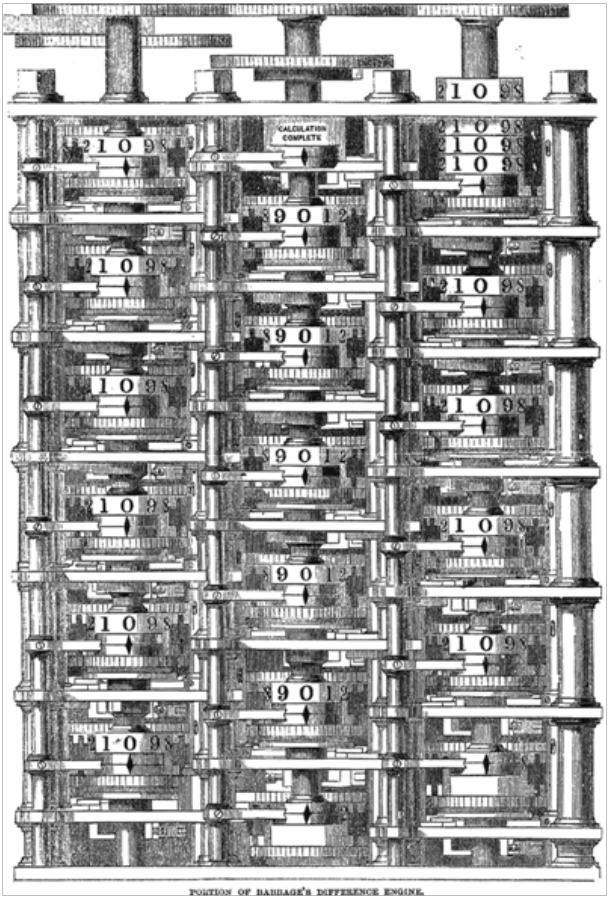


Рисунок 7.5
Разностная машина (От ХарпераNewMonthlyMagazine).

вычитания. К концу 1800-х годов и в 1900-х годов , электричество заменило силу пара для привода вращения механических элементов. Другие механические элементы и аналоговые элементы (в том числе в одном случае, количество воды) были использованы вместо громоздких зубчатых колес. Но к 1930-х и 1940-х годов, релейные выключатели , которые были использованы в телефонной сети, должны были заменить громоздкие механические компоненты. Чертеж релейного выключателя показан на рисунке 7.6. Типичный релейный выключатель составляет около 3 см2 в размерах (менее дюйма и полтора). Релейный выключатель может переключать состояния быстрее, чем зубчатое колесо может вращаться таким образом, что производительность вычислительного устройства улучшило бы также. Релейный выключатель будет находиться в одном из двух положений и, таким образом, компьютеры перемещаются из десятичной системы в двоичную, и теперь упоминается как цифровые вычислительные машины.

Первое поколение

Большинство аналоговых (десятичное) и цифровых вычислительных машин вплоть до середины 1940-х годов были специальные станки-предназначенные только для одного типа вычисления (хотя они были программируемыми, что конкретное вычисление может различаться).

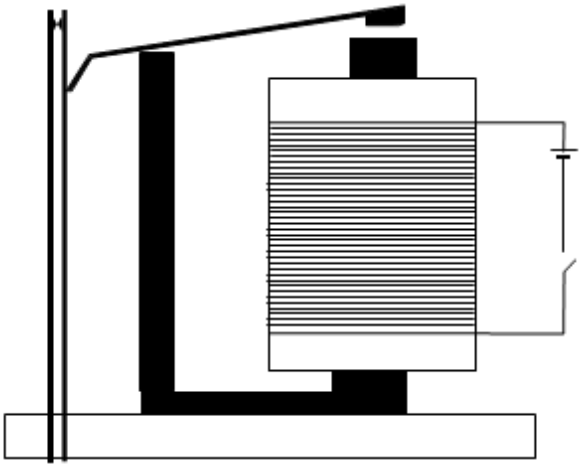


Рисунок 7.6

Электромагнитный релейный выключатель. (Взято из http://commons.wikimedia.org/wiki/~~HEAD=pobj Файл:Schema_rele2.PNG.)

Таблица 7.1 Ранние компьютеры

Название	Год	Национальность	Комментарии
ЦузеZ3	1941	Немецкий	Binary с плавающей точкой, электромеханический, программируемый Атанасофф-ягодный 1942 США Binary, электронные, nonprogrammable
Колосс Марк 1	1944	Великобритания	Binary, электронные, программируемые
Гарвардский (Mark 1)	1944	США	Десятичный, электромеханический, программируемый
ColossusMark 2	1944	UK	Binary, электронные, программируемые
Цузе	1945	немецкий	Z4 Binary с плавающей точкой, электромеханический, программируемый

К их числу относятся устройства вычислений интегральных уравнений и дифференциальных уравнений (заметим, что это отличается от ранее упомянутых разностных уравнений).

К тому времени, когда началась Вторая мировая война, была гонка строить лучше, быстрее, и более пригодные для использования компьютеров.И были необходимы эти компьютеры, чтобы помочь в вычислении ракетных траекторий. Интересная историческая справка, что первые компьютеры не были машины, они были женщины, нанятые британским правительством для выполнения расчетов траектории ракеты вручную! Таблица 7.1 содержит описание некоторых из ранних машин с начала 1940-х годов.

Союзники также надеялись построить компьютеры, которые могли взломать немецкие коды. Несмотря на то, что закончено после Второй мировой войны, ENIAC (электронный цифровой интегратор и компьютер) был первым цифровым, общего назначения, программируемым компьютером, и он потерял весь интерес к аналоговым вычислительным машинам. Что отличает ENIAC от компьютеров в таблице 7.1 является то, что ENIAC был общего назначения, тогда как в таблице 7.1 были либо специального назначения (может работать только программы определенного типа) или не были электронными, но электромеханическими . Общего назначения компьютер может

предположительно выполнить любую программу, которая может быть записана для этого компьютера.

Построенный в Университете Пенсильвании, Эниак впервые был известен общественности в феврале 1946 года. Компьютер стоил около $ 500 000 (из 1940-х годов деньги) и состоял из 17,468 вакуумных трубок, 7200 кристаллических диодов, 1500 реле, 70000 резисторов, 10000 конденсаторов и миллионы ручных спаев. Он весил более 30 тонн и занял 1800 ft2. Ввод данных осуществляли с помощью перфокарт, а программирование осуществлялось подсоединением вместе различных электронных компонентов с помощью кабелей, так что выход одного компонента будет использоваться в качестве входных данных для другого компонента (рисунок 7.7). Выход был произведен с использованием автономной счетной машины . Хранение ENIACбыло ограничено до около 200 цифр. Интересно отметить, что хотя компьютер был цифровой компьютер (что обычно означает бинарное представление), то ENIACвыполняет десятичные вычисления. Хотя ENIAC претерпел некоторые обновления в 1947 году, это было в непрерывном использовании с середины 1947 года до октября 1955 года. ENIAC, с использованием вакуумных ламп для хранения, транзисторов и других электронных устройств для вычисления, удалось вычислить по ставке 5000 операций в секунду. Тем не менее, зависимость от вакуумных трубок, а также трудности в программировании путем подключения компоненты вместе с помощью кабеля, привело к очень ненадежной производительности. На самом деле, самое длинное времяENIACоколо 116 часов.

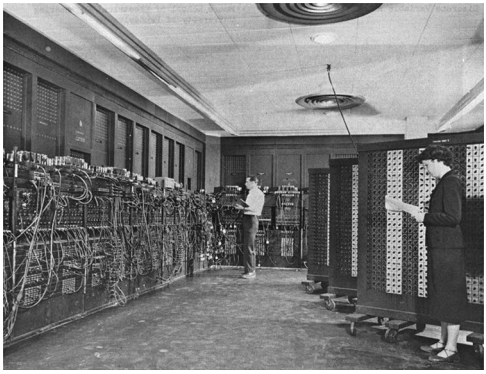


Рисунок 7.7

Программирование электронного цифрового интегратора и компьютера (ENIAC). (Фото предоставлено http://commons.wikimedia.org/wiki/File:Eniac.jpg~~pobj, автор неизвестен.)

Эниак и другие лабораторные компьютеры, как это представляет собой первое поколение компьютерной техники, все из которых были единственные в своем роде машины. Они классифицируются не тольков период времени, но и зависимость от вакуумных ламп, реле выключатели, а также необходимость программирования на машинном языке. К 1940-транзисторы используются в различных электронных приборах . Примерно в 1959 году были разработаны первые компьютеры, которые использовали компоненты транзисторов вместо вакуумных трубок. Транзисторы предпочитали вакуумные лампы по ряду причин.

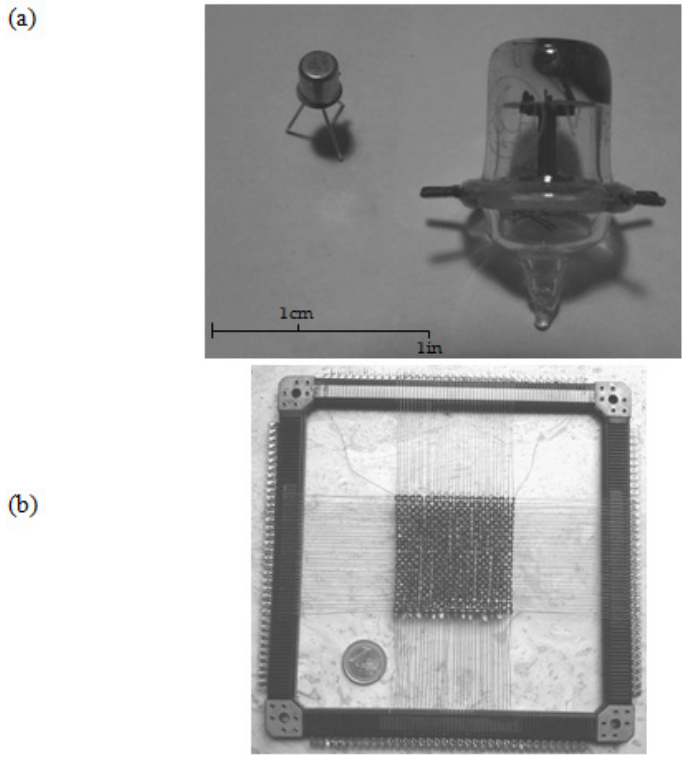


Рисунок 7.8.
Вакуумная трубка и транзистор (а) (Предоставлено Эндрю Кевин Пуллен, HTTP: // Коммонс
.wikimedia.org / вики / Файл:. 955ACORN.jpg) и память на магнитных сердечниках (б) (предоставлено HandigeHarry, http://commons.wikimedia.org/wiki/File:Core_memory. JPG~~HEAD=pobj).

Они могут быть массового производства и, следовательно, были намного дешевле. Вакуумные трубки источали много тепла и имели короткий срок годности, возможно, несколько тысяч часов, в то время как транзисторы могут длиться до 50 лет. Транзисторы используют меньше энергии и были гораздо более надежными.

Второе и третье поколения

Примерно в то же время, память на магнитных сердечниках внедрялась. Память магнитного сердечника состоит из мелких железных колец из металла, помещенных в рамку из проволочной сетки. Каждое кольцо хранит один бит, при наличии магнитного тока вращаются по часовой стрелке или против часовой моды.
Именно эти новшества, которые возвестили в новом поколении компьютеров, теперь называют вторым поколением. Рисунок 7.8 иллюстрирует эти две новые технологии. Рисунок 7.8a обеспечивает сравнение между вакуумной трубкой и транзистором (обратите внимание на разницу в размерах), а рисунок 7.8b показывает коллекцию магнитной памяти ядра. Совокупность магнитных сердечников и проводов представляют собой память, где каждое кольцо (на пересечении горизонтальной и вертикальной проволоки) хранит один бит. Провода используются для указания ядра, где осуществляется доступ. Ток течет вдоль одной из множества проводов, так что ядра могут сохранить свои заряды. Другой набор проводов используется для отправки новых битов для выбора ядра или для получения значений из отдельных ядер. Логика компьютера (управление выборки-выполнения цикла, и выполнение арифметических и логических операций) может быть достигнуто за счет коллекций транзисторов. Например, НЕ операция может быть сделано с двумя транзисторами, И или ИЛИ операция с шестью транзисторами и сложения схемы 1-бит с 28 транзисторов. Таким образом, потребовалось бы сто транзисторов, чтобы построить простой процессор.
Устраняя вакуумные трубки, компьютеры стали более надежными. Памяти магнитного сердечника, хотя и очень дорогая, позволяет компьютерам иметь большие основные размеры памяти (сотни или тысячи байтов, свыше миллиона байт). Кроме того, размер компьютера был снижен из-за уменьшения размеров аппаратного обеспечения. С маленькими компьютерами , физическое расстояние, что электрический ток должен проходить между компонентами, которые меньше, и, таким образом, компьютеры стали быстрее (меньше расстояние, тем

меньше время, затраченное на поездки, что расстояние). Кроме того, компьютеры стали легче программировать с инновационных новых языков программирования (смотрите раздел Эволюция компьютерных программ).

Другие компьютеры были изготовлены и приобретены таким образом, что компьютеры не были уже ограничены государственными лабораториями или научно-исследовательскими лабораториями университетов. Внешнее запоминающее устройство переходит от медленной и громоздкой магнитной ленты для дисков и дисков барабанов. Компьютеры этой эпохи в основном называли мэйнфреймы-компьютеры , построенные вокруг твердого металлического каркаса. В общем, второе поколение нашло дешевле, быстрее, проще программировать, и более надежные компьютеры. Тем не менее, это поколение было недолгим. На рисунке 7.9 показаны компоненты ЭВМ IBM 7094 (от 1962), включая многочисленные ленточные накопители катушки на катушку для хранения.
В течение 1950-х годов, кремниевый чип был введен. К 1964 году первые силиконовые чипы использовались в компьютерах, возвещая в третьем поколении. Чипы, известные как печатные схемы или интегральные схемы (ИС), могут включать десятки транзисторов. Микросхема будет образцом транзисторов травлением на поверхности куска кремния,

Рисунок7.9.
Компания IBM 7094 ЭВМ. (From National Archives and Records Administration, record # 278195, authorunknown, http://arcweb.archives.gov/arc/action/ExternalIdSearch?id=278195&jScript=true.)

который будет проводить электричество, отсюда термин полупроводник. Штыри позволили бы IC, или чип, который должен быть прикреплен к розетке, так что электрический ток может течь от одного места в компьютере через цепь и выход в другое место. Рисунок 7.10 показывает как офорты, которые составляют IC и сам чип с булавками, чтобы вставить чип в материнскую плату. Чип, показанный на рисунке 7.10b, представляет собой типичный чип с конца 1960-х годов.

ИС будет заменять и громоздкие транзисторы и магнитные сердечники памяти, так что чипы будут использоваться как для вычислений и хранения . ИС занял меньше места, так что опять расстояние, что ток должен течь, был снижен еще больше. Более быстрые компьютеры были

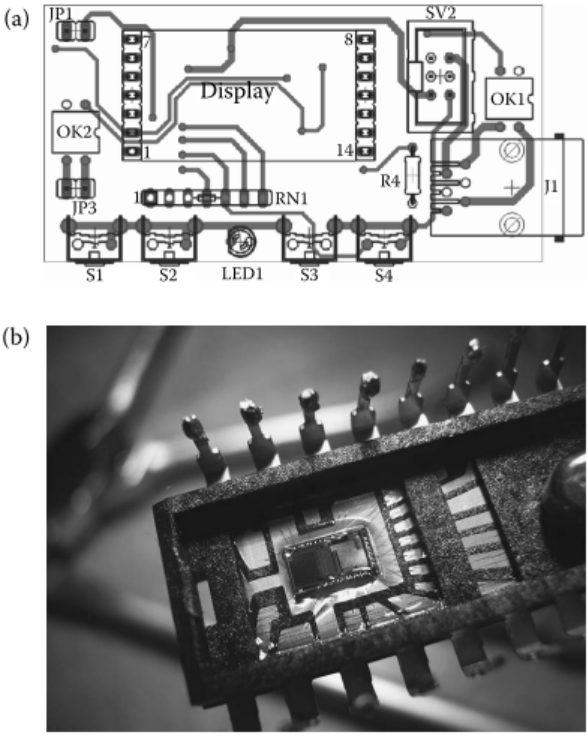


Рисунок 7.10.
Интегральная схема (а) (Фото любезно предоставлено Мартина Броз, http://commons. wikimedia.org/ вики / Файл:.Navrh_plosny_spoj_soucastky.png) и кремниевый чип (б) (предоставлено Xoneca, HTTP: // Обще .wikimedia.org / вики / Файл:. Integrated_circuit_ optical_sensor.jpg) Display – дисплей

результатом. Кроме того, ИС может быть массовым производством, поэтому стоимость изготовления компьютера была сокращена. Теперь, даже малогабаритные организации могли бы рассмотреть вопрос о приобретении компьютера. Мэйнфреймы по-прежнему производятся при затратах, возможно, $ 100 000 или более. Теперь, однако, компьютерные компании также производят миникомпьютеры по сниженной цене, возможно, по цене от $ 16 000. В мини-ЭВМ были существенно уменьшенные мэйнфреймы, они использовали один и тот же тип процессора, но уменьшил число регистров и обработку элементов , снижение памяти, и так далее, так что они будут поддерживать меньшее количество пользователей. Мэйнфрейм мог бы использоваться большой организацией сотен или тысяч людей, в то время как миникомпьютер мог бы использоваться небольшой организацией десятками или сотнями пользователей.

В течение третьего поколения, компьютерные компании начали производить семейства компьютерами. Идея заключалась в том, что любой компьютер в данной семье должен иметь возможность запускать одни и те же программы без необходимости изменять программный код. Во многом это было связано с компьютерами одного и того же семейства, используя один и тот же процессор, или по крайней мере процессоров, которые имеют тот же набор инструкций (машинный язык). Компьютер

Рисунок 7.11.
Немой терминал, около 1980 года (адаптировано из Wtshymanski, http://en.wikipedia.org/~~HEAD=pobj вики / Файл:.Televideo925Terminal.jpg)

семьи родили области разработки программного обеспечения, как кто-то может написать код для всей семьи и продавать эту программу, чтобы потенциально десятки или сотни клиентов.
Организации теперь закупают компьютеры с ожиданием того, что многие сотрудники (десятки, сотни, даже тысячи) будут его использовать. Это создало необходимость некоторых механизмов, посредством чего сотрудники могли получить доступ к компьютеру удаленно без необходимости обращаться к самой компьютерной комнате. Компьютерные сети были введены, что позволит отдельным пользователям подключаться к компьютеру через *немой* терминал (рисунок 7.11). Немой терминал был всего лишь устройством ввода и вывода, у него не было никакой памяти или процессора. Все вычисления и хранение имело место на самом компьютере. Операционные системы были усовершенствованы для обработки нескольких пользователей одновременно. Развитие операционной системы также описано в эволюции компьютерного программного обеспечения.

Четвертое поколение

Следующее важное нововведение состоялось в 1974 году , когда IBM выпустила процессор однокристальный. До этого момента, все процессоры были распределены по нескольким десяткам, возможно, чипы (или в более ранние дни, вакуумные трубки и релейных переключателей или транзисторов). Создавая однопроцессорной чип, известный как микропроцессор, можно было бы построить небольшой компьютер вокруг одного чипа. Эти компьютеры были названы микрокомпьютерами . Такой компьютер будет достаточно мал, чтобы стоять на столе человека. Это положило начало в самое последнее поколение, четвертое поколение. Это было новшество микропроцессора, который привел первых персональных компьютеров в середине 1970-х годов. Эти компьютеры были немного больше, чем устройства человека, увлеченного своим хобби, практически без деловых или личных возможностей. Эти ранние микрокомпьютеры были проданы в качестве составных частей, и поставил бы любитель компьютер вместе, возможно размещение компонентов в деревянном ящике, и есть компьютер к телевизору или принтеру. Самый ранний такой компьютер был Mark 8. AppleComputersбыл основан в 1976 году, и их первый компьютер , AppleI, был также продан в виде комплекта, что люди будут строить в коробке. В отличие от Марка 8, AppleI стал сенсацией (рисунок 7.12).

Хотя первые микрокомпьютеры были маленькими, вычислительное использование было любительской игрушкой, со временем они становились все более и более популярными, и, таким образом, были заинтересованы в их улучшении. К концу 1970-х годов, как микропроцессоры и емкости памяти компьютерабыли улучшены , что позволяет более способных микрокомпьютеров, в том числе те, которые могут выполнять рудиментарные компьютерные графики . Скромная обработка текстов и бухгалтерского программного обеспечения, введенного в конце 1970-х годов, сделали эти компьютеры полезные для малого и среднего бизнеса. Позже, программное обеспечение было введено, которые могли бы заполнить нишу рынки, такие как настольной издательской системы, музыки и искусства и образования. В сочетании с графическим интерфейсом пользователя (ГИП), персональные компьютеры стали привлекательным вариантом не только для бизнеса, но

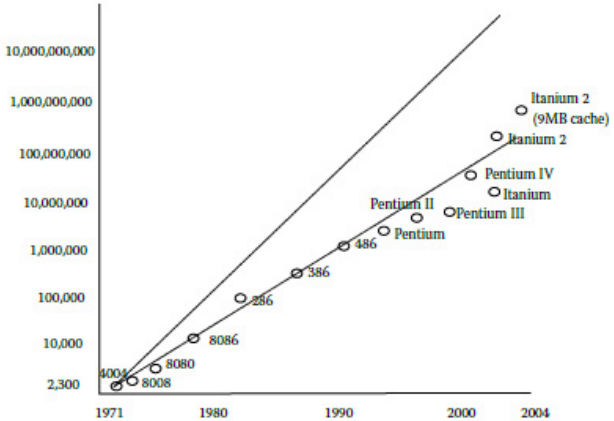
Таблица 7.2 Линейные по сравнению с экспоненциальным ростом

1000	1000
1200	2000
1400	4000
1600	8000
1800	16,000
2000	32,000
2200	64,000
2400	128,000

Рисунок 7.12.
AppleI построил в деревянный ящик. (Фото предоставлено Alpha1, http://commons. wikimedia.org/~~HEAD=pobj вики / Файл:.Apple_1_computer.jpg)

и для домашнего использования. С введением коммерциализированного Интернета, компьютер стал больше, чем бизнес или учебным пособием, а сегодня это, конечно, как в общем домашнем хозяйстве в качестве автомобиля. Наиболее существенное изменение, которое произошло с 1974 года, можно резюмировать одним словом: миниатюризация . Компьютеры третьего поколения состоятиз несколько плат, соединенных между собой в шасси. Каждая плата содержала множество микросхем, и каждый ICсодержал несколько десятков транзисторов (до нескольких сотен до конца 1960-х годов). К 1970, это было возможно уменьшить на тысячу транзисторов , которые будут размещены на одном кристалле. С течением времени , тенденция миниатюризации транзисторов продолжалась с экспоненциальной скоростью.

Большинство улучшений в нашем обществе происходят с низкой скоростью, в лучшем случае предлагают линейное увеличение производительности. Таблица 7.2 демонстрирует разницу между экспоненциальным и линейным ростом. Показатель линейного улучшения увеличивается на коэффициент, примерно на 20% за определенный период времени. Это приводит к удвоению производительности по сравнению



Транзистор для различных чипов Intel на протяжении многихлет. Верхняя линия показывает уровень, если количество транзисторов удваивается каждые 18 месяцев, нижняя линия показывает уровень, если количество транзисторов удваивается каждые 24 месяца

Рисунок 7.13.

Набросок миниатюризации – закон Мура . Шкала слева возрастает экспоненциально, а не линейно. (Взято из Wgsimon, http://commons.wikimedia.org/wiki/File:Moore_~~HEAD=pobjLaw_diagram_% 282004% 29.jpg.)

с первоначальной производительностью , в течение примерно пяти периодов времени. Экспоненциальное улучшение удваивается каждый период времени, что приводит к увеличению, что порядки величины равны большеза тот же период. В таблице, увеличение по сравнению с 1000 до 128,000 происходит всего за семь периодов времени.

Что это значит по отношению к нашим компьютерам? Это означает, что в течение многих лет, число транзисторов, которые могут быть размещены на чипе, увеличилось на несколько порядков величины, а не по линейному закону. Улучшения в наших компьютерах были драматичны в довольно короткие сроки. Это был Гордон Мур, один из основателей Intel, который первым заметил это увеличивая явление "количество транзисторов" . В докладе 1965 года , Мур отметил, что тенденцией с 1958 по 1965 было то , что число транзисторов на кристалле удваивается каждый год. Это явление получило название «закон Мура». Мур предсказал, что эта тенденция будет продолжаться в течение, по крайней мере, еще 10 лет. Мы находим, на самом деле, что степень миниатюризации является удвоение транзистора, рассчитывать примерно каждые 18 до 24 месяцев, и что эта тенденция продолжается с 1965 года до настоящего времени. График на рисунке 7.13 иллюстрирует прогресс, отметив число транзисторов на нескольких различных процессорах , выпущенных в период с 1971 по 2011 год. Следует еще раз подчеркнуть, что закон Мура является наблюдением , а не физический закон. Мы привыкли полагаться на тенденции в миниатюризации, но там, конечно, никакой гарантии, что эта тенденция будет продолжаться вечно. На самом деле, было много инженеров, которые считали, что темпы роста должны привести к замедлению работы к 2000. После того, как мы достигли 2000, и закон Мура продолжает реализоваться, инженеры чувствовали, что 2010 год увидит конец этой тенденции. Немногие инженеры сегодня считают, что закон Мура будет продолжаться в течение более чем еще несколько лет или десятилетий, и поэтому есть много активных исследований, расследующих новые формы полупроводникового материала, кроме кремния.

Что это может означать, если закон Мура должен был подвести нас? Подумайте, как часто вы покупаете автомобиль. Скорее всего, вы покупаете автомобиль, потому что ваш текущий автомобиль не дорогой достойный, возможно, из-за аварий, износ от чрезмерного пробега, или неспособность сохранить автомобиль до стандартов (хотя в некоторых случаях, люди покупают новые автомобили, чтобы отпраздновать акции и т.д.). Теперь рассмотрим компьютер. Существует мало износа от

использования и составные части редко терпят неудачу (жесткий диск является одним устройством в компьютере с движущимися частями и, вероятно, потерпеть неудачу гораздо раньше, чем любой другой). Желание приобрести новый компьютер почти полностью появляется из-за устаревания.

Что делает компьютер устарело? Потому что новые компьютеры лучше. Как они лучше? Они быстрее и имеют больше памяти. Почему ? Потому что миниатюризация привела к большему количества транзисторов и, следовательно, более способных компонентов, которые быстрее и имеют большие хранения. Если мы не в состоянии продолжать увеличивать число транзисторов, новые компьютеры будут немного лучше или не лучше, чем современные компьютеры. Таким образом, люди будут иметь меньше необходимости покупать новый компьютер каждые несколько лет. Мы вернемся к этой мысли позже в этой главе. Закон Муране привело нас к чрезвычайно мощным процессорам современных компьютеров. Конечно, уменьшение размеров компонентов на чипе означает, что время, необходимое для электрического тока, чтобы путешествовать, продолжает уменьшаться, и таким образом мы получаем ускорение в наших процессорах. Тем не менее, большее значение имеют архитектурные новшества к процессору, введенные компьютерными инженерами. Они стали доступны в основном потому, что там больше места на самом чипе, чтобы приспособить большее количество схем. Между избыточным количеством миниатюризации и архитектурных инноваций, наши процессоры буквально в миллионы раз мощнее , чем у ENIAC. У нас также есть основные возможности памяти 8 ГБ, ряд, который считался бы невозможным , как совсем недавно, в 1980-е годы. Многие архитектурные новшества были введены на протяжении многих лет с середины 1960-х годов. Но это период последних 20 лет, который видел наиболее значительные продвижения. Одним из самых важных нововведений является конвейерный процессор. В трубопроводе, выборки-цикл выполнить данную выполняется во взаимном перекрывании на нескольких инструкций. Например, в то время как инструкция 1 выполняется, инструкция 2 декодируется и обучение 3 пересылается . Это приведет к трем инструкциям, которые в то же время находятся в некотором состоянии исполнения. CPU не выполняет все три одновременно, но каждая команда проходит часть выборки-выполнения цикла. Трубопроводы могут изменяться по длине от трех стадий, как описано здесь, чтобы хорошо более десятка (некоторые из современных процессоров имеют 20 или более ступеней).

Конвейерный процессор очень похож на автомобильный сборочный конвейер . В сборочной линии, многочисленные рабочие работают на разных машинах одновременно.
Другие новшества включают параллельную обработку, на кристалле кэш-памяти, зарегистрировать стеклам, аппаратное обеспечение, спекулирует о том, следует ли принимать ветвь, и совсем недавно, несколько ядер (несколько процессоров на одном чипе). Все эти понятия, изучаемые в области информатики и вычислительной техники.
Воздействие микропроцессора не может быть преувеличено . Без него мы бы не имели персональные компьютеры, и поэтому мы не будем иметь даже меньшие вычислительные устройства (например, смартфоны). Тем не менее, четвертое поколение не ограничивается только новшеством , выдвинутое миниатюризацией . Мы также видели огромные улучшения в вторичных устройств хранения данных. Емкость жесткого диска достигла и превысила 1 Тб (то есть 1 байт лев трлн. , в которую вы могли бы хранить 1 миллиард книг текста или около 10000 высококачественных компакт-дисков или более 1000 фильмов с низким разрешением). Флэш-диски являются также обычным явлением сегодня и дают нам мобильность для передачи файлов. Мы также видели введение долговечных батарей и ЖК-технологии, чтобы предоставить нам мощный ноутбук и примечание- книги компьютеров. Кроме того, широкополосная беспроводная технология позволяет нам общаться практически в любом месте.

СТАНКИ, изменившие мир

В 1992 году, ПБС в эфир из пяти частей серии на компьютере под названием Машина, которая изменила мир. Это должно быть очевидно при чтении этой главы, что многие машины изменили наш мир, наша технология прогрессировала и развивалась. Вот список других машин стоит упомянуть.

Z1 построенный Конрадом Цузе в Германии, она предшествовала ENIAC, хотя частично механический характер, используя телефонные реле, а не вакуумные трубки.
UNIVAC I-построенный создателями ENIAC для Remington Rand, это был первый коммерческий компьютер, начиная с 1951 года.
Первый CDC 6600-в мире суперкомпьютер был фактически выпущен в 1964 году, а не 1980-е года !

Этот мэйнфрейм превзошел следующий самый быстрый компьютер, в 3 раза.

Altair 8800, хотя знак 8, был первым персональным компьютером, это был Altair 8800, что компьютерные любители изначально вышли. Этот компьютер был продан как компьютерный комплект для сборки. Хотя его популярность была ничто по сравнению с Apple I, который последовал спустя 5 лет.

IBM 5100-первый коммерческий доступный портативный компьютер (известный как портативный компьютер в те дни). Машина весила 53 фунтов и стоимость от $ 9000 и $ 19000! IBM Simon-B 1979 году NTT (Nippon по телеграфии и телефонии) выпустила первые сотовые телефоны. Но первый смартфон был IBM Simon, выпущенный в 1992 году. Его возможности были ограничены мобильным телефоном, пейджером, факсом, а также несколько приложений, таких как календарь, адресная книга, часы, калькулятор, блокнот, и адрес электронной почты.

Таблица 7.3 Эпохальные сравнения

Год	Автомобильная эпоха	Вычислительная эпоха Год	Год
6500 до н.э.	Колесо	Abacus	4000? До н.э.
1769	Паросиловое транспортное средство	Механический калькулятор	1642
1885	Изобретение автомобиля	Программное устройство	1801
1896	Первая автомобильная смерть (пешехода сбила машина со скоростью 4 миль в час)	Первый механический компьютер разработан (аналитический двигатель)	1832
1904	Первая автоматическая коробка передач	Первый цифровой, электронный, компьютер общего назначения (ENIAC)	1946
1908	Сборочная линия позволяет массовое производство	Второе поколение компьютеров (дешевле, быстрее, надежнее)	1959
1911	Первые компьютеры электрического зажигания	Третье поколение (ИС, дешевле, быстрее, компьютерные семьи)	1963

1925	Около 250 шоссе, доступных в Соединенных Штатах	ARPANET (первоначальное воплощение Интернета)	1969
1940	Одна четверть всех американцев имеют собственный автомобиль	Четвертое поколение компьютеров (микропроцессоры, персональные компьютеры)	1974
1951	Автомобили достигают 100 миль в час при разумных затратах	Первый жесткий диск для ПК	1980
1956	Межгосударственная система шоссе правомочно (взял 35 лет, чтобы закончить)	Выпущен IBMPC	1981
1966	Первые тормоза антиблокировочной системы	Macintosh (первый GUI)	1984
1973	Национальные ограничения скорости, установленные в Соединенных Штатах, начинается энергетический кризис	Столлман вводит GNUS	1985
1977	Введены парковки для инвалидов	Доступ в Интернет в домашних хозяйствах	1990-е годы
1997	Разработан первый гибридный двигатель	Дешевые ноутбуки, смартфоны	2000-е годы

Сегодня у нас есть портативные устройства, которые являются более мощными, чем компьютеры от 10 до 15 лет назад. Наши настольные и портативные компьютеры в миллионы раз мощнее , чем самые ранниекомпьютеры 1940-х и 1950-х годов . И тем не менее, наши компьютеры стоили нам всего лишь нескольких сотен до тысячи долларов, достаточно мало денег, что люди часто выбрасывали свои компьютеры, чтобы купить новые в течение времени, всего за несколько лет. Эта глава начинается с сопоставления автомобиля и компьютера. Мы закончим этот раздел, взглянув на некоторые этапы в автомобильной промышленности, по сравнению с этапом в компьютерной индустрии (таблица 7.3). Обратите внимание на то, что эти этапы в улучшенных

автомобилях заняли большее количество времени, чем этап в вычислении, в то время как этапы в автомобиле, по большей части, не завезли автомобили, которые являются порядками величины, больше, поскольку улучшения в компьютерной отрасли.

Эволюция программного обеспечения для компьютеров

Самые ранние компьютеры были запрограммированы на машинном языке (напомним, из главы 2, что программа на машинном языке представляет собой длинный список 1S и 0s). Это были инженеры, то есть создание и поддержание компьютеров в день, которые были программирования на компьютерах . Они были также пользователи, единственные люди, которые будут запускать программы. Ко второму поколению, лучшие языки программирования были произведены, чтобы сделать задачу программирования проще для программистов. Программисты часто были еще инженеры, хотя не обязательно те же инженеры. На самом деле, программисты могут быть деловые люди, которые хотели, чтобы программное обеспечение выполняло операции , которое их текущее программное обеспечение для выполнения операций, которое их текущее программное обеспечение не может выполнить. В середине 1960-х годов, сотрудниками IBM было принято решение прекратить производство своего собственного программного обеспечения для ряда своих комплектующих. В результате организации, которые приобрели компьютеры IBM, пришлось пойти куда-нибудь в другое место для покупки программного обеспечения. Так были введены компании по разработке программного обеспечения, и зародилась индустрия программного обеспечения.

История программного обеспечения не является почти столь же волнующей, как история аппаратного обеспечения. Тем не менее, на протяжении многих десятилетий, было несколько очень важных нововведений. Они кратко рассматриваются в данном разделе. Специализируясь на ИТ, вы должны будете составлять программы, но вам, скорее всего, не придется беспокоиться о многих из проблем, которые возникли в процессе эволюции программного обеспечения, потому что ваши программы в большинстве случаев будут короткими скриптами. Тем не менее, эта история показывает то, как мы пришли к тому, где мы находимся в отношении к программированию .

Ранние компьютерные программы были написаны машинным кодом, написаны для конкретной машины, часто инженерами, которые

сами конструировали и запускали компьютеры. Ввод программы не был простым набором его текста, без подключения областей памяти к вычислительным схемам с помощью кабеля, так же, как раньше телефонный оператор соединял звонки. Как только программа была выполнена, приходилось соединять схему заново во всем компьютере для запуска следующей программы. Напомним, что первые компьютеры были отчасти ненадежны из-за короткого срока службы и ненадежности вакуумных трубок. Но добавьте к этому сложную природу машинного кода программирования и метод ввода программы (через кабель), и вы оказываетесь в очень сложной ситуации. В качестве исторической справки, первый программист не мог заставить работать свою программу, даже, несмотря на то что , вакуумные трубки работали, его логические функции были правильными, и программа была введена правильно. Оказалось, что каким-то образом мотылек попал в переключатель реле и так, оно не пропускало ток. Это, по общему мнению, и привело к термину «баг», который используется для обозначения ошибки. Теперь мы используем термин отладки программы не только для обозначения устранения ошибок в программе, а просто о какой-либо форме поиска неисправностей! Смотри рисунок 7.14.

При этом напомним, что ввод / вывод компьютерного устройства (I /

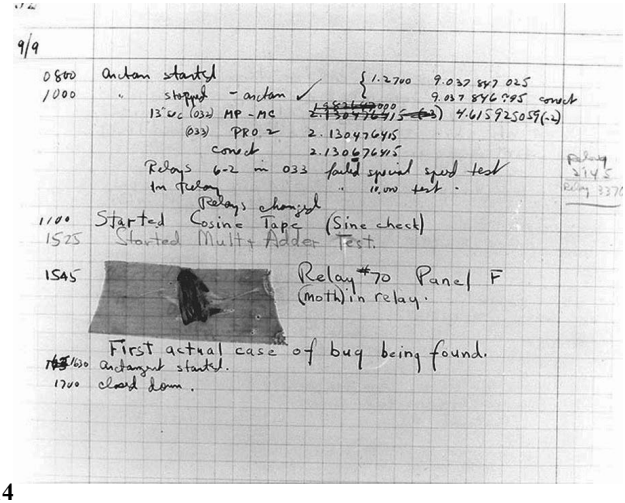


Рисунок 7.14
Первая компьютерная ошибка. (Предоставлено Военно-морским историческим центром США, фотография интернет библиотеки NH 96566-KH.)

О), был ограничен главным образом для считывания с перфокарт или магнитной ленты и записи на магнитную ленту. Любой произведенный вывод сохранялся на ленту, которая демонтировалась из ленточного устройства, устанавливалась на принтер, а затем распечатывалась. Так как компьютер запускал только одну программу за раз, то все вводы и выводы были ограничены таким образом, и не было никакой необходимости в операционной системе компьютера. Программист использовал любой код ввода-вывода в самой программе.

Рутина программирования стала намного проще после нескольких инноваций. Первая была идея программы транслятора. Эта программа брала другую программу в качестве входных данных и выводила версию машинного кода, который затем мог использоваться на компьютере. Оригинальная программа, часто известная как «исходный код», не могла быть выполнена в своем первоначальном виде. Первые трансляторы были известны как «ассемблеры», которые переводили программу ассемблирования в машинный код. Язык ассемблирования, хотя и проще, чем машинный язык, по-прежнему требуется очень подробные, точные и низкоуровневые инструкции (вспомним пример из Главы 2). К 1959 году перевод языка улучшился до такой степени, что конвертированный язык мог быть записан способом более похожим на английский, с гораздо более мощными программными конструкциями. Усовершенствованный класс транслятора был известен как «компилятор», и языки назывались «языками высокого уровня». Первый из этих трансляторов был создан

```
C code
     for(i=1;i<=x;i++)
          sum=sum+i;

 Код языка ассемблирования                          Машинный код
01231407    mov    dword ptr [i], 1        C7 45 EC 01 00 00 00
0123140E    jmp    main+69h (1231419h)     EB 09
01231410    mov    eax, dword ptr [i]      8B 45 EC
01231413    add    eax, 1                  83 C0 01
01231416    mov    dword ptr [i], eax      89 45 EC
01231419    mov    eax, dword ptr [i]      8B 45 EC
0123141C    cmp    eax, dword ptr [x]      3B 45 F8
0123141F    jg     main+7Ch (123142Ch)     7F 0B
01231421    mov    eax, dword ptr [sum]    8B 45 E0
01231424    add    eax, dword ptr [i]      03 45 EC
01231427    mov    dword ptr [sum], eax    89 45 E0
0123142A    jmp    main+60h (1231410h)     EB E4
```

Рисунок 7.15
Сравнение высоко уровня, кода ассемблирования и машинного кода.

для языка под названием FORTRAN. Идея FORTRAN в том, что программист во многом указывал математические формулы, используя алгебраические обозначения, наряду с командами ввода, вывода и контроля. Команды контроля (циклы, выборы) были очень похожи на язык ассемблирования, но остальная часть языка читалась больше, как английский и математика. Название языка происходит от ТРАНслятора ФОРмул. FORTRAN который в первую очередь предназначался для математического / научного программирования. Язык для описания коммерческих задач был также создан примерно в то же время под названием COBOL (Универсальный язык для коммерческих задач). Другие языки были разработаны для поддержания исследований в области искусственного интеллекта (LISP), моделирования (Simula), а также работы со строками (SNOBOL).

Давайте сравним код высокого уровня, ассемблированного к машинному коду. Рисунок 7.15 предоставляет простой оператор Си, которая вычисляет сумму всех целых значений от 1 до введенного значения, x. Код С представляет собой один оператор: цикл for. Тело цикла само по себе является оператором однократного присваивания.

Когда одна команда С ассемблируется в коде ассемблируемого языка (для процессора Intel x86), код имеет длину в 12 команд. Это показано на рисунке в виде трех колонок информации. Первая колонка содержит адрес памяти, хранящий команды. Вторая колонка является, по сути, самой работой, представленной, мнемокодом (аббревиатурой). Третья колонка содержит операнды (данные), которые обрабатывает команда.

Например, первая команда перемещает значение 1 в область, на которую указывает переменная, поданная в виде “dword ptr”. Четвертая команда является, пожалуй, самой легкой в понимании, “add eax”, 1 добавляет значение 1 в регистр данных под именем “eax”. Каждая команда ассемблера конвертируется в одну команду машинного кода. В этом случае машинный язык (справа на рисунке) команды приведены в шестнадцатеричной системе. Так, например, первая команда “mob” (перемещение данных), состоит из 14 шестнадцатеричных значений, показанных в парах. Нужно только изучить этот простой код С, чтобы понять, каким может быть зашифрованный язык ассемблера. Мнемокод языка ассемблера может дать указания на то, что делает каждая операция, но машинный код почти полностью смутный для понимания. Вместо этого, код С сообщает нам с помощью английских слов, таких как “for”, наряду с переменными и математической системой символов.

Разыскиваются Программисты!

Программист и инженер ПО довольно новые термины в нашем обществе. Первый университетский факультет программирования был в Университете Перьдью в 1962 году Первый кандидат наук в области программирования был получен Университете Пенсильвании в 1965 г. Аж 1970-ых годах, программирование было введено во многих университетах. В нашем языке до 1968 года даже не было термина «программная инженерия».

На сегодняшний день, большинство программистов получают ученые степени по программированию, хотя существуют и некоторые программы ученых степеней по инженерии ПО. Но кем были программисты, до появления специалистов по информатике? Как ни странно, так же, как история Джо из главы 1, в программиста превратился специалист ИТ, ранние программисты были самоучками. Это были инженеры и математики, или они были администраторами фирма или бухгалтерами. Если бы вы знали, как программировать, вы могли бы сменить специализацию и стать программистом.

Сегодня компьютерные программисты — это вымирающий вид. Мало компаний заинтересованы в приеме на работу кого-то, кто знает, как программировать, но не имеет существенных знаний основ в программировании. Итак, сегодня, когда вы видите программирование в перечне работ, то, скорее всего, от вас потребуется ученая степень по информатике, информационных системах, компьютерной инженерии, или ИТ.

Хотя язык ассемблера для программиста расшифровать не является невозможным, все еще остается проблема, разобраться в нем. Код языка высокого уровня, независимо от того, каком языка, состоит из английских слов, математических символов, и знакомого синтаксиса, такого как точка с запятой для завершения команд. Язык программирования С не был написан до 1968 года; однако, другие языки высокого уровня той эпохи (FORTRAN, COBOL и т.д.) были все же гораздо легче для понимания, чем любой язык машинного кода или ассемблера. Переход к языкам программирования 204 Информационные технологии высокого уровня представляет собой одно из наиболее значительных достижений в области программирования, потому что без него, разработки программного обеспечения будет не только проблемой, сырой язык программирования будет ограничивать размер разрабатываемого программного обеспечения. Вряд ли команда из 20 программистов может производить программу на миллионы линий , если они вынуждены писать на машинном языке или языке ассемблера.

В середине 1960-х годов, компьютеры становились все более доступными. Кроме того, было использовано больше ресурсов ввода и вывода и компьютерные сети позволяли пользователям подключаться к компьютеру с немых терминалов или удаленных локаций. И теперь, это было не только инженеры, которые используют компьютеры и программирование. В организации, которая имела компьютер, любой сотрудник мог получиться

пользователем компьютера. Эти пользователи, возможно, и не поняли комплектующего компьютера, ни как программировать компьютер. С учетом всех этих добавленных приложений, потребовалась программа, которая могла позволить пользователю вводить простые команды для запуска программ и перемещать файлы данных таким образом, чтобы пользователь не должен был, по сути, писать полные программы. Это привело к первым операционным системам.

В начале 1960-х годов, операционная система была названа «резидентным монитором». Она всегда будет в памяти, доступной для призыва любым пользователем. Она была известна как «монитор», потому что она отслеживала запросы пользователей. Запросы были в значительной степени ограничены для запуска программы, указывая местоположение ввода (который ленточный накопитель или диск, какой файл (ы)), и место назначения вывода (принтер, файл на диске, магнитной ленте файл и т.д.). Однако, с течением 1960-х годов и больше пользователей смогли получить доступ к компьютерам, резидентный монитор должен был стать более сложным. К середине 1960-х, резидентный монитор был назван операционной системой – программой, позволяющей пользователю управлять компьютером. Операционная система будет отвечать за планирование программ (поскольку несколько программ могут быть запрошены несколькими пользователями), выполнение программы (запуск и мониторинг программы во время выполнения, завершение программы, по окончанию работы), и пользовательский интерфейс. Операционная система будет вынуждена обрабатывать запросы нескольких пользователей одновременно. Работа программы сначала была выполнена на основании мультипрограммирования, а потом, временном разделении (теперь называется многозадачностью).

Операционные системы также обслуживают защиту пользователя (гарантируя, что один пользователь не нарушает ресурсы, принадлежащие другим пользователем) и сетевую безопасность. На протяжении 1960-х и 1970-е годов, операционные системы были основаны на тексте. Таким образом, даже если пользователь не должен был понимать компьютерные комплектующие или быть в состоянии программировать, от пользователя требовалось понимание, как использовать команды операционной системы. В таких системах, как VMS (Система виртуальной памяти), которая работает на компьютерах DEC (Корпорации Диджитал Эквипмент) компьютерах VAX и JCL (Язык управления заданиями), который работает на базовом блоке IBM, команды могли быть изощренными и сложными, как с языком программирования.

С развитием персонального компьютера, могла применяться более простая операционная система. Одной из самых популярных была, так называемая MS-DOS, дисковая операционная система. Команды были в значительной степени ограничены работой диска (или хранилища) - запуск программы, сохранение файла, перемещение файла, удаление файла, создание каталогов, и так далее. Как ни странно, хотя название операционной системы DOS, она может быть использована для любого дискового или ленточного хранилища! Следующее новшество в операционных системах не появлялось до 1980-х годов. Тем не менее, в то же время ....

Уроки, извлеченные программистами в 1960-е годы, привели к новой революции в 1970-е годы известной, как структурное программирование. Команды перехода, известные как GOTO использовались в большом количестве ранних языков. Оператор GOTO позволил программистам передать управление из любого места в программе, куда-нибудь еще в программе. Для программиста, эта свобода может быть замечательной вещью, пока вы не должны понять программу, чтобы изменить ее или исправить ошибки. Причина заключается в том, что команда GOTO создает то, что теперь называется «спагетти-кодом». Если вы должны были отследить программу, вам пришлось бы следовать инструкциям в последовательном порядке. Тем не менее, с использованием оператора GOTO, внезапно после любой команды, вам, возможно, придется переместиться в другое место в программе. Отслеживание программы начинает выглядеть как тарелка спагетти. В структурном программировании, программист ограничен конструкциями управления высокого уровня, такими, как циклы с предусловием для циклов и команд «если-иначе», и не допускается использование более примитивного оператора GOTO. Это открыло новую эру языков высокого уровня, С и Паскаль являются одними из наиболее известных.

В 1980-х годах, еще одно новшество вышло в свет в мире программирования. Вплоть до середины 1980-х годов, программист, который хотел смоделировать некоторую сущность в мире, будь то физический объект, такой как автомобиль, или абстрактный объект, такой как технический документ, использовал отдельные переменные. Переменные описывали характеристики объекта. Для автомобиля, например, переменные могут включать в себя возраст, расход бензина, тип автомобиля, количество миль и текущую ценность по Синей книге (журнала спецификаций). Лучший подход моделирования существовал для определения классов сущностей, называемых объектами. Объекты

могли быть созданы программой, каждый, будучи уникальным и моделирования другой физический объект (например, учитывая класс автомобиля, мы могли бы генерировать четыре различных автомобиля). Объекты будут затем взаимодействовать друг с другом и с другими типами объектов. Разница между объектно-ориентированного подхода и старым подходом, основанным на переменной, является то, что объект – самостоятельная сущность, который будет запрограммирован для обработки своих собственных внутренних методов, а также сообщений, полученных от других объектов. И с помощью указанных классов, программист может расширить язык путем определения дочерних классов. Посредством технологии, так называемой наследованием, программист может взять предыдущий класс и создать более конкретный класс из него. Это обеспечивает степень повторного использования, в которой программисты могут использовать классы других программистов без необходимости самостоятельно заново изобретать код. Понятие наследования показано на рисунке 7.16, где класс велосипеда является основой для нескольких более конкретных типов велосипедов. Например, все велосипеды в иерархии представляют собой способы транспортировки, которые содержат два колеса, приводятся в движение человеческим педалированием. Тем не менее, существуют подтипы велосипедов, такие как гоночный велосипед, женский велосипед или велосипед с учебными колесами.

Объектно-ориентированное программирование (ООП) было впервые введено на языке Smalltalk в 1970-х и начале 1980-х годов. В середине 1980-х годов, вариант объектно-ориентированных возможностей Small-

Рисунок 7.16
Объектно-ориентированное наследование (Основано на Шаттерсток/фотоизображение).

talk был включен в новую версию C, которая называется C ++. C ++ стала настолько популярной, что другие объектно-ориентированные языки программирования (OOPL) были введены в 1990-х годах. Сегодня почти каждый язык программирования имеет возможности объектно-ориентированного программирования.

Рука об руку с развитием OOPLs стало введение первой оконной операционной системы. Для того, чтобы продемонстрировать использование объектно-ориентированного программирования, исследователями искусственного интеллекта в Калифорнийском Ксерокс Пало Альто (Xerox Parc) построили оболочку на основе окон. Идея заключалась в том, что окно будет моделироваться как объект. Каждое окно будет иметь некоторые общие черты (размер, расположение, цвет фона) и операции, которые будут работать на любом окне (его перемещение, изменение его размера, его разрушение). В результате получилась оконная операционная система, которую они использовали, чтобы помочь улучшить свои исследования. Они не думали особо над продажей их изобретения, но они были готовы продемонстрировать его посетителям. Два таких посетителя, Стивен Джобс и Стивен Возняк (изобретатели персональных компьютеров Apple), нашли это нововведение необычным и предсказали, что это может изменить программирование. Джобс и Возняк провели много лет, реализуя свою собственную версию. Джобс участвовал в разработке наличия ГПИ у первого персонального компьютера, Apple Lisa. Но когда он был выпущен в 1983 году за 10 000 $ за единицу продукции, было их продано, очень мало. На самом деле, проект Лиза стал такой неразберихой, что Джобс был вынужден покинуть проект в 1982 году, и вместо этого он перешел на другой проект Apple, Macintosh. Выпущенный в 1984 году за $ 2500, Макинтош имел огромный успех.[2]

Оконный интерфейс позволяет людям пользоваться компьютером, направляя указатель на пункты меню или с помощью движения нажатий и перетаскивания. С оконной системой, нет необходимости изучать язык операционной системы, такие как VMS, DOS или Unix, но вместо

2 Для тех из вас, кто достаточно пожилого возраста , вы могли бы вспомнить первый рекламный ролик Apple Macintosh. Проветривание во время Super Bowl XVIII в январе 1984 года за $ 1,5 млн, реклама показала ряд одинаково одетых людей, которые ходят через однообразные прихожие путей и сидят в зале. Они слушали "Большого брата" - подобие фигуры. Молодая спортсменка в белой майке выбежала в коридор и бросила молоток в экран. При ударе, экран взорвался, и реклама закончилась надписью "... вы увидите, почему 1984 не будет похож на" 1984 "" Реклама руководилась режиссером Ридли Скоттом. Вы можете найти ролик на YouTube.

этого, теперь можно управлять компьютером интуитивно практически без уроков. Таким образом, это был графический интерфейс, который открывает использование компьютера для всех. Сегодня, графическое программирование и ООП являются мощными инструментами для программиста. Почти все программное обеспечение на сегодняшний день является графическим по своей природе, и многое из производимого программного обеспечения происходит от объектно-ориентированного языка программирования.

Еще одним нововведением в программировании является использование интерпретатора, а не компилятора. Компилятор требует, чтобы компоненты программного обеспечения были определены до начала компиляции. Это может быть сложным, когда программный проект состоит из нескольких десятков до сотен отдельных компонентов, все из которых должны быть написаны до компиляции. В таких языках, как Java, например, сначала нужно написать какие-классы, которые будут призваны другим классом. Тем не менее, если вы просто хотите проверить идею, вы не можете сделать это с неполным кодом.

Язык программирования Лисп, разработанный в конце 1950-х годов для исследований искусственного интеллекта, использовал интерпретатор. Это позволило программистам протестировать одну команду за раз, таким образом, они могли построить свою программу по частям.

Основное различие между скомпилированным языком и интерпретируемым языком является то, что интерпретируемый язык работает внутри специальной среды, называемой интерпретатором. Программист вводит команду, интерпретатор преобразует эту команду на машинный язык и выполняет ее. Команда может применять результат предыдущих команд, ссылаясь на переменные, установленные предыдущими командами, хотя это не обязательно. Таким образом, интерпретированное программирование опирается на понятие сессии. Если программисту удается выполнить нескольких команд, которые идут рука об руку, то программист может обернуть их в программу. В компилированном языке, вся программа написана, прежде чем он может быть вписан, что необходимо выполнения. Есть много причин насладиться интерпретированным подходом к программированию; однако, создание эффективного кода не является одной из них. Поэтому компилируется большинство крупных программных проектов.

В качестве ИТ-администратора, ваша работа, скорее всего, требует, чтобы вы, время от времени, писали свой собственный код. К счастью, поскольку эффективность не обязательно будет проблемой (ваш код

будет относительно небольшим, возможно, всего лишь несколько команд на программу), вы можете использовать интерпретируемую среду. В операционной системе Linux, оболочка содержит свой собственный интерпретатор. Таким образом, написание программы является вопросом размещения ваших Linux команд в файл. Эти небольшие программы называются сценариями (скриптами), а процесс программирования называется созданием сценариев командной оболочки. В операционной системе Windows, вы можете написать DOS файлы для пакетной обработки для подобных результатов.

Разработка скриптов, однако, выходит за пределы системного администрирования. Небольшие скрипты часто пишутся в сети управления, администрирования веб-сервера и администрирования серверов баз данных. Скрипты также являются инструментом выбора для веб-разработчиков, которые будут писать небольшие скрипты для запуска на веб-сервере (серверные скрипты), или в веб-браузере (клиентские сценарии). Серверные скрипты, например, используются для обработки данных, введенных в формах веб-страниц или для создания динамических веб-страниц, извлекая информацию из базы данных. Клиентские сценарии используются для взаимодействия с пользователем, например, путем обеспечения того, чтобы форма была заполнена правильно или через компьютерную игру. Многие из интерпретируемых языков сегодня могут служить в качестве языков сценариев для этих целей. Языки сценариев включают Perl, PHP, Ruby, Python и asp. Мы рассмотрим программирование и некоторые из этих языков написания скриптов более подробно в главе 14.

Эволюции КОМПЬЮТЕР ПОЛЬЗОВАТЕЛЬ

Так же, как развивались компьютерное оборудование и программное обеспечение, так и компьютерный пользователь. Самые ранние пользователи компьютеров были инженерами, которые и конструировали, и программировали компьютеры. Эти пользователи имели высокоспециализированные знания в области электроники, электротехники и математики. Были только сотни таких людей, работающих на нескольких десятках лабораторных машинах. К 1950м годам пользователи прогрессировали и включили программистов, которые больше не были (в обязательном порядке) инженерами, конструируемые компьютеры.

В 1960-х и в 1970-е, пользователи включили сотрудников организаций,

которые владели или приобрели время на компьютере. Эти пользователи, как правило, были очень образованными, но, возможно, не учеными по информатике и инженерами. Некоторые пользователи были компьютерными операторами, работа которых включала такие задачи, как монтаж лент на ленточные накопители и работу с компьютерными комплектующими в чистых комнатах. Но большинство пользователей, вместо этого, работали из своих офисов на немых терминалах. Для этой группы пользователей, их навыки не должны включать в себя математику и электронику, ни программирование, хотя в большинстве случаев они были обучены командам операционной системы своих компьютерных систем, чтобы они могли вводить команды операционной системы для выполнения своих задач. Ниже приведены некоторые примеры из VMS, операционной системы для компьютеров VAX DEC, которые были очень популярными 1970-е и 1980-е годы (отрывок из учебника по VMS).

```
ASSIGN DEV$DISK:[BOB]POSFILE.DAT FOR015
COPY SYS$EXAMPLES:TUT.FOR TUT.FOR
DEL SYS$EXAMPLES:TUT.FOR
PRINT/FORM = 1/QUEUE = SYS$LASER TUT.FOR
SHOW QUEUE/ALL SYS$LASER
```

В течении 1970-х, персональный компьютер позволял почти каждому использовать компьютер, независимо от их знаний. Для того, чтобы контролировать ранние компьютеры, пользователи должны были знать, что-то из операционной системы, поэтому опять же, они писали команды. Вы уже рассматривали некоторые команды MS-DOS ранее в тексте. Хотя команды нельзя легко запомнить, их гораздо меньше для освоения, чем в более сложных операционных системах мейнфреймов, таких как VMS. С выпуском Apple Macintosh в 1984 году, однако, использование команд операционной системы устарело. Вместо того, чтобы вводить загадочные команды в командной строке, пользователь управляет компьютером с помощью графического интерфейса. Все программное обеспечение Macintosh было основано на ГПИ, впервые, пользователь не придется иметь какие-либо специальные знания для использования компьютера. Он был интуитивно легким. Операционная система Microsoft Windows была введена через год, а к 1990-ым, почти на всех компьютерах была доступна операционная система с графическим интерфейсом на оконной основе и прикладное программное обеспечение. В интересном повороте событий, часто это были молодые поколения, которые легко

учились пользоваться компьютером, в то время как старшее поколение, те, кто вырос, думая, что компьютеры были огромными, дорогими, и пугающими, больше всего колебались, чтобы научиться использовать компьютером.

Самыми последними разработками в области использования компьютерной техники является внешний вид и устройство ввода данных с чувствительным сенсорным экраном, такие как смартфоны и планшетные ПК. Впервые введенный различными продуктами компании Apple, сенсорный экран обеспечивает еще более интуитивный контроль над компьютером, чем более традиционный ГПИ в оконном стиле. Скроллинг, постукивание, введение через клавиатуру, и использование своих пальцев, для перехода, возможно, является следующим поколением операционных систем. Уже объявлено, что Microsoft планирует адаптацию внешнего вида и чувствительности для их следующего поколения настольной операционной системы Windows 8.

Сегодня удивительно найти кого-то, кто не знает, как пользоваться компьютером. Необходимый уровень квалификации остается на низком уровне для использования компьютера. На самом деле, с таким количеством смартфонов на планете, примерно половина населения планеты можно считать пользователями компьютеров.

Чтобы понять компьютерные основы, нужно знать некоторую основу компьютерной грамотности. Чтобы понять более сложные понятия, такие как установка программного обеспечения, монтаж комплектующих, мониторинг производительности, и так далее, необходимы даже большие знания. Большую часть этих знаний можно получить, прочитав инструкции или журналы. Это также можно изучить,

Наблюдая, как друг или член семьи делает что-то подобное, некоторые пользователи учатся методом проб и ошибок. Сегодня, из-за огромного влияния, которое компьютеры оказали на наше общество, большая часть этих знаний легкодоступна в Интернете, поскольку есть веб-сайты, которые создают компьютерные компании и частные лица, которые помогают пользователям учиться. Только самые узкие знания по ремонту комплектующих, системного / сетевого администрирования, разработка программного обеспечения и вычислительной техники требует обучения и / или школу.

ВЛИЯНИЕ НА ОБЩЕСТВО

Вы находитесь в автомобиле, который движется вниз по улице. Вы

достигли стоп-сигнала и ждете, когда он переключится. Ваш GPS указывает на то, что вы должны идти прямо. Вы слушаете спутниковое радио. В то время как горит сигнал стоп, вы набираете сообщение другу на вашем смартфоне (теперь запрещено в некоторых штатах). Вы находитесь на пути в кино. Ранее вы смотрели прогноз погоды по телевизору, который говорил Вам, что дождь был неизбежен. В то время как вы даже не прикасались к вашему компьютеру сегодня, вы погружаетесь в области компьютерных технологий.

• Ваш автомобиль содержит несколько процессоров: карбюратор с системой топливного впрыска, антиблокировочная тормозная система, приборная панель, все это программируемые устройства с ИС.
• Ваш GPS является программируемым устройством, которое получает входные данные от сенсорного экрана и от орбитальных спутников.
• Ваше спутниковое радио работает благодаря спутникам на орбите, запущенным ракетой и компьютером, и запрограммированных для доставки сигналов в конкретные области страны.
• Ваш смартфон представляет собой уменьшенный компьютер.
• Фильм, который вы собираетесь посмотреть, был написано почти наверняка людьми с помощью текстовых процессоров. Фильм, вероятно, отредактирован в цифровой форме. Специальные эффекты были, вероятно, добавлены с помощью компьютерной графики. Список, как компьютеры были использованы в производстве фильма, скорее всего, будет длинным, и даже не учитывает маркетинга или рекламу фильма.
• Погода была предсказана благодаря спутникам (опять-таки, выведенных на орбиту ракетой и компьютером), а модели погоды были запущены на компьютерах.
• Даже фонарь стоп-сигнала, вероятно, контролируется компьютером. Видите, датчики в дороге?

В нашем современном мире практически невозможно избежать взаимодействия с компьютером. Вам придется переехать в отдаленный район и целенаправленно избавить себя от технологии для того, чтобы удалить компьютеры из вашей жизни. И, тем не менее, вы все еще можете почувствовать их влияние через счета, почтовое отделение, походы в загородные магазины, а также от своих соседей.
Использование компьютера можно найти в любой и каждой карьере. От А до Я, будь то бухгалтерия, реклама, авиаперелеты, вооруженные силы, искусство, или зоология, компьютеры используются, чтобы помочь нам,

и даже сделать для нас много работы.
Но воздействие не ограничивается только нашим использованием компьютеров в качестве рабочей силы. Они являются первоначальной основой для экономических операций. Почти все наши покупки происходят в электронном виде:
Инвентаризация, взимание с кредитных карт, денежные начисления. Компьютеры доминируют над нашими формами развлечения, является ли это создание фильма, телевизионных программ, музыки или искусства, или это просто носитель, с помощью которого мы рассматриваем / слушаем / смотрим искусство. Более существенно то, что компьютеры в настоящее время находятся в центре наших коммуникаций. Помимо очевидного использования сотового телефона для телефонных звонков, мы используем наши мобильные устройства, чтобы увидеть друг друга, чтобы отслеживать, где люди (да, даже шпионить друг за другом), чтобы прочитать последние новости, даже играть в игры.
И тогда есть Интернет. Через Интернет, мы делаем покупки, мы инвестируем, мы учимся, мы читаем (и смотрим) новости, мы ищем развлечения, мы поддерживаем контакт с семьей, друзьями и незнакомыми людьми. Глобальный характер Интернета в сочетании с доступностью, что люди имеют в современном обществе, сделало возможным иметь голос всем и каждому. Блоги и доски для постов находят всех, кто хочет поделиться своим мнением. Социальные сети позволили нам поддерживать дружеские отношения удаленно и даже завести новых друзей и любовников.
Интернет также обеспечивает межкультурные связи. Теперь мы можем увидеть, как это жить в других странах. Мы можем увидеть историю и исторические места тех стран. Мы можем наблюдать за собрания или развлечения, проводимые в этих странах через YouTube. И с социальными сетями, мы можем узнать, почти мгновенно, какие новости происходят в этих странах. В качестве простого примера, в течение 2011 года проходила арабская весна. И в то время как она разворачивалась, весь мир наблюдал за этим.
Для того, чтобы увидеть влияние компьютера, рассмотрите следующие вопросы:

1. Когда в последний раз вы написали кому-то почтовое письмо? По какой причине, кроме отправки поздравительной открытки или платежа?
2. Когда в последний раз вы ходили к другу в гости без предварительного звонка по своему мобильному телефону или письма по электронной

почте?
3. Когда в последний раз вы читали новости исключительно из газеты?
Вы будете изучать историю операционных систем, с особым акцентом на DOS, Windows и Linux, а также истории Интернета в главах 8 и 12, соответственно. Мы повторно рассмотрим языки программирования и некоторые из их эволюций в главе 14.

ГЛАВА 8.ИСТОРИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Глава 7 охватывала эволюцию аппаратных средств и программного обеспечения. В этой главе основное внимание уделяется эволюции операционных систем с особым акцентом на развитие Linux и Unix, и отдельно, разработке ПК на базе операционных систем графическим интерфейсом. Изучение Linux охватывает два отдельных потока: во-первых, разнообразие дистрибутивов Linux, которые существуют сегодня, и во-вторых, воздействие сообщества разработчиков открытого ПО на развитие Linux.
Целями обучения данной главы являются

• Описание, как операционные системы эволюционировали.
• Обсуждение влияния, которое Linux получило в мире вычислительной техники.
• Сравнение различных дистрибутивов Linux.
• описание роли, которую сыграло сообщество разработчиков открытого ПО.
• Прослеживание разработки операционных систем для ПК.

В этой главе мы посмотрим на историю и развитие различных операционных систем (ОСС), концентрируясь в основном на ОС Windows и Linux. Мы высветим эти две операционные системы, потому что они являются одними из самых популярных в мире, и они представляют две отдельные философии программного обеспечения. Windows, продукт компании Microsoft, является операционной системой, разработанной в качестве коммерческой платформы и, следовательно, является собственностью компании, с версиями, выпущенных в попытке побудить или заставить пользователей модернизироваться и тем самым тратить больше денег. Linux, которая развилась из ОС Unix, охватывает движение Open Source. Он бесплатный, но более того, содержание ОС (исходный код) открыто доступно, так что разработчики могут усиливать или изменять код и производить свое собственное программное обеспечение для ОС-до тех пор, пока они соблюдают соглашение по лицензированию

Open GNUS (GL). Перед тем, как рассматривать любую ОС Windows или Linux, мы посмотрим на эволюцию более ранних ОС.

ПЕРЕД LINUX И WINDOWS

Как уже говорилось в главе 7, ранние компьютеры не имели операционных систем. Пользователями этих компьютеров были программисты и инженеры, которые конструировали компьютеры. Предполагалось, что их программы будут работать с малым количеством или совсем без ресурсов (возможно, через доступ к перфокарте для ввода, с выходным сигналом, сохраняющимся на магнитной ленте). Операции чтения с перфокарт и записи на магнитную ленту должны были быть вставлены в сами программы, и, таким образом, были написаны пользователями.
Некоторые изменения привели к необходимости создания ОС. Первым было развитие языковых переводчиков. Чтобы запустить программу, программист должен сначала скомпилировать (или ассемблировать) программу из своего исходного языка на машинный язык. Программисту придется запрограммировать компьютер на несколько различных шагов. Во-первых, программа перевода (компилятор или ассемблер) должен был бы быть введен с ленты или перфокарты. Во-вторых, исходный код должен быть введен, снова с ленты или перфокарты. Программа перевода будет работать, выполняя загрузку исходного кода, переводя исходный код, и сохраняя полученный исполняемый код на магнитную ленту. Затем исполняемая программа будет введена с магнитной ленты, с введенными данными снова ж таки с карты или ленты. Эта программа будет выполнена и вывод будет направлен на магнитную ленту для последующей печати. Без ОС, каждый из входных и выходных задач должны быть написаны как часть программы. Для упрощения, резидентный монитор был создан для обработки этих операций на основе нескольких команд, а не десятков или сотен программных инструкций. Смотри рисунок 8.1, который иллюстрирует на языке JLC (JCL используется на мэйнфреймах IBM) инструкции для копирования файлов из одного места в другое.
Еще одно изменение, которое привело к необходимости ОС, наличие компьютеров. Во втором поколении, компьютеры были менее дорогими, в результате чего больше компьютеров было продано и больше организаций, имело к ним доступ. В связи с увеличением количества компьютеров пришло увеличение количества пользователей. Пользователи, начиная со второго поколения, не обязательно были программистами или инженерами. Теперь, обыкновенный сотрудник

может использовать компьютер. Операционная система представляет собой механизм, который позволил людям использовать компьютер без необходимости того, чтобы понять, как запрограммировать компьютер.
Кроме того, улучшение компьютерных комплектующих привело к ОС двумя способами. Во-первых, улучшенная скорость работы привела к тому, что программы требуют меньше времени для работы. В первом поколении, пользователь компьютера был также инженером и программистом. Переход от одной программы и, таким образом, одного пользователя к другому занимал много времени. С помощью программы, на выполнение которой нужно меньше времени, желание улучшить этот переход привел к ОС. Кроме того, повышение надежности компьютеров (отказавшись от короткого срока службы вакуумных трубок) позволило выполняться более длинным программам до завершения. С более длинными программами пришла необходимость обработки дополнительных ресурсов программирования и, следовательно, к большему спросу на ОС.
ОС становились более сложными с развитием мультипрограммирования и разделения времени (многозадачности). Это, в свою очередь, позволило десяткам или сотням, или тысячам пользователей использовать компьютер одновременно. Это увеличение в использовании, в свою очередь, требует, чтобы операционные системы обрабатывали все больше и больше задач, и поэтому ОС стали еще более сложными. И, как описано в главе 7, оконная ОС в четвертом поколении изменила то, как пользователи взаимодействуют с компьютерами. Ниже приводится краткое описание некоторых из ранних операционных систем и их вклад.
Самой ранней ОС, используемой для "реальной работы" была GM-NAA ввода / вывода, написанная General Motors для использования на компьютерах с мэйнфреймами IBM 701. Ее главная цель состояла

```
//IS198CPY  JOB (IS198T30500),'COPY JOB', CLASS = L, MSGCLASS = X
//COPY01    EXEC  PGM = IEBGENER
//SYSPRINT  DD  SYSOUT = *
//SYSUT1    DD  DSN = OLDFILE, DISP = SHR
//SYSUT2    DD  DSN = NEWFILE,
//              DISP = (NEW, CATLG, DELETE),
//              SPACE = (CYL, (40,5), RLSE),
//              DCB = (LRECL = 115, BLKSIZE = 1150)
//SYSIN     DD  DUMMY
```

Рисунок 8.1
Команды JCL

в том, чтобы автоматически выполнять новую программу после того, как предварительное выполнение программы было прекращено. Она использовала пакетную переработку, в которой ввод был предоставлен с программой. Это была коллекция расширенного резидентного монитора, написанного в 1955 году, наряду с программами, которые могут получить доступ к устройствам ввода и вывода, подключенным к мэйнфреймам.
Atlas Supervisor в 1957 году разрешила одновременный доступ пользователей для Манчестерского университета. Она считается первой настоящей ОС (а не резидентным монитором). Параллельная обработка не будет в целом доступны до начала 1960-х годов (см CTSS). Кроме того, Atlas была одной из первых, которая предложила виртуальную память.
BESYS была ОС Bell, разработанная в Bell Laboratories в 1957 году для мэйнфреймов IBM 704. Она может обрабатывать входной сигнал от и перфокарт и от магнитной ленты, а также выводить информацию на любой принтер или магнитную ленту. Она был создана для компиляции и запуска программ на FROTRAN.
IBSYS, начиная с 1960, была выпущена компанией IBM с IBM мэйнфреймами 7090 и 7094. Команды ОС были встроены в программы, вставляя $ перед любой командой ОС, чтобы отличить ее от команды FORTRAN. Такой подход позволил впоследствии использовать для реализации инструкции JCL (Рисунок 8.1) для программ мэйнфреймов IBM 360 и IBM 370.
CTSS, или совместимая система с разделением времени, выпущенная в 1961 году в Массачусетском институте вычислительного центра технологий (MIT), была первой истинной ОС с разделением времени (многозадачная). В отличие от Atlas Supervisor, один из компонентов CTSS отвечал за проведение пользовательских процессов, предлагая каждому долю процессорного времени (отсюда название разделение времени).
EXEC 8, в 1964 году, была подготовлена для мэйнфреймов Remington Rand's UniVac. Это примечательно, потому что она была первой успешной коммерческой многопроцессорной ОС (то есть операционная система, которая работает на нескольких процессорах). Она поддерживает несколько форм управления процессами: порционное, разделения времени и обработки в реальном времени. Последняя означает то, что, как ожидается, процессы будут работать немедленно, без задержек и завершаться в течение заданного срока, предлагая взаимодействие с пользователем в реальном времени.
TOPS-10, также с 1964 года, была выпущена компанией Digital Equipment

Corporation (DEC) для их серии мэйнфреймов PDP-10. TOPS-10 является иной известной ОС с разделением времени, поскольку она представила общую память. Общая память позволит нескольким программам общаться друг с другом через память. Чтобы продемонстрировать использование совместно используемой памяти, была разработана многопользовательская компьютерная игра на основе Star Trek под названием DECWAR.

MULTICS, с 1964 года, ввела динамическое связывание программного кода. В программной инженерии, она является общей для программистов, чтобы призвать библиотеки подпрограмм (куски кода, написанного другими программистами, скомпилированы и сохранены в библиотеке). При использовании динамического соединения между кодами, эти фрагменты загружаются в память только при необходимости. MULTICS первая предложила это. Сегодня мы видим подобный подход в ОС Windows с использованием файлов "DLL". MULTICS была ОС, разработанной на основе модулей таким образом, что она могла поддерживать множество различных аппаратных платформ и была масштабируема, так что она все еще была эффективной, когда в систему были добавлены дополнительные ресурсы. Кроме того, MULTICS представила несколько новых концепций, включая списки контроля доступа для контроля доступа к файлам, разделяя память процессов и файловую систему (т.е., рассматривая работу процесса памяти, как если бы это было файловое хранилище, так как с Unix и Linux с помощью каталога процессов), и иерархическая файловая система. Хотя MULTICS была не очень коммерчески успешной ОС, она оставалась в эксплуатации в течение более десяти лети, и все еще использовалась некоторыми организациями до 1980 года. Это была одна из самых влиятельных ОС, хотя бы, потому что она послужила основой для последующей ОС Unix.

OS360 для мэйнфрейма IBM 360 (и более поздних версий, OS370 для IBM 370) была выпущена в 1966 году и была первоначальной партией ОС, а затем добавила мультипрограммирование. Кроме того, какая-либо одна задача может быть выполнена с помощью многозадачности (по существу, была многопоточной в том, что процесс может быть многозадачным, но не было никакой многозадачности между процессами). OS360 использовала JCL для получения команд ввода / вывода. OS360 разделяла ряд новшеств, введенных в другие операционные системы , таких как виртуальная память и иерархическая файловая система, но и представила свой собственный виртуальный метод доступа к хранилищу (который позже станет основой для хранения баз данных),

а также способность порождать дочерние процессы. Другой элемент OS360 был объектом передачи данных, что позволило ОС иметь связи с любым типом терминала. Из-за популярности мэйнфреймов IBM 360 и 370, OS360 стала одной из самых популярных операционных систем своего времени. Будучи модифицированной для IBM 370, OS360 (переименованная в System / 370) сама имела несколько модификаций. Сегодня OS360 остается популярной экспериментальной ОС и доступна для свободного скачивания.
UNICS (позже, UNIX) была разработана AT & T в 1969 году (смотрите следующий раздел). Кроме того, в 1969 году Программа управления Авиалинией (ACP) IBM была отделена от остальной части системы автоматизации авиакомпании IBM, которая обрабатывала бронирования авиабилетов. После отделения, ACP, позже известна как TPF (Устройство обработки трансакций) была ОС обработки транзакций для базы транзакций данных авиакомпании для обработки таких задач, как обработка кредитных карт, а также гостиниц и бронирование аренды автомобилей. Хотя она и не инновационная сама по себе, это дает пример ОС специально для задачи, а не аппаратной платформы.
В начале 1970-х годов, большинство операционных систем просто расширили возможности, введенных в течение 1960-х годов. Заметной ОС, VM используемой IBM, и выпущенной в 1972 году, позволяет пользователям создавать виртуальные машины.

История UNIX

ОС Unix датируется концом 1960-х годов. Это одна из самых мощных и портативных операционных систем. Ее мощность исходит от различных особенностей: администрирования файловой системы, сильных сетевых компонентов, безопасности, установки программного обеспечения на заказ, а также возможности определять свои собственные внутренние программы, оболочки и скрипты для адаптации окружения. Часть мощности Unix исходит от его гибкости предложения командной строки для приема входных сигналов ОС. Командная строка позволяет пользователю или системному администратору для ввода команд с большим разнообразием вариантов. Она портативна, потому что она написана с помощью C, языка, который может быть скомпилирован для почти любой платформы, и она была перенесена на ряд самых разных типов компьютеров, включая суперкомпьютеры, мэйнфреймы, рабочие станции, персональные компьютеры и ноутбуки. На самом деле, Unix

стала настолько популярной и успешной, как теперь ОС для Macintosh (хотя Mac оконная система занимает первое место).
Unix был создан в Лаборатории AT & T Bell. Оригинальное использование ОС заключалось на PDP-7, чтобы сотрудники могли играть в игру на этой машине (под названием Space Travel). Unix была написана на ассемблере и не была портативной. В самом начале это была не успешная ОС; на самом деле, было не так много систем вообще. После первоначального внедрения, два сотрудника лабораторий Bell, Кен Томпсон и Деннис Ритчи, начали укрепление Unix путем добавления средства работы с файлами (копирование, удаление, редактирование, печать) и интерпретатора командной строки так, что пользователь может вводить команды одну за другой за раз (а не через серию перфокарт называемых "работа"). К 1970 году ОС была официально названа Unix.
К началу 1970-х годов, Unix была переработана для запуска на PDP-11, гораздо более популярном, чем машины PDP-7. Перед тем как она была переписана, Ритчи первым разработал и внедрил новый язык программирования, C. Он специально разработал язык, который мог бы реализовать ОС, а затем он использовал C, чтобы переписать Unix для PDP-11. Другие сотрудники были вовлечены в переписывание Unix и добавили такие функции, как трубы (они обсуждаются в главе 9). Операционная система была затем распространена по другим фирмам (за плату, не бесплатно).
К 1976 году Unix была перенесена на несколько компьютеров, и была заинтересованная в Unix группа, которая непрерывно повышала обсуждения и поддержку ОС. В 1976 году Томпсон взял академический отпуск из Bell Labs, чтобы преподавать в Университете Калифорнии в Беркли, где он и студенты UCB разработали версию стандартного распределения Unix Беркли, теперь известную как BSD Unix. Версия BSD 4.2 станет чрезвычайно популярным релизом. В течение этого периода, Unix, также приняла протокол TCP / IP, таким образом, что компьютеры могли обмениваться данными по сети. TCP / IP был протоколом, используемым ARPAnet (что стало Интернетом), и Unix был там, чтобы обеспечить это.
В 1980-е годы, растущая популярность Unix ослабевала, и она стала операционной системой по выбору для многих компаний, приобретающих мэйнфреймы и мини-ЭВМ. Несмотря на то, что она была дорогой и сложной, это был, пожалуй, лучший доступный выбор. В 1988 году изобретение открытого программного обеспечения (OSF) было основано с явным намерением, что программное обеспечение и ОС

были разработаны свободно. Термин "бесплатный" не то, что вы могли бы ожидать. Основатель, Ричард Столлман, сказал бы "бесплатно, как в свободе не пиво". Он стремился, что бы сообщество пользователей Unix тратило время на разработку новой операционной системы и поддержки программного обеспечения, чтобы любой мог получить его, использовать его, изменить его, и опубликовать модификации. Результат - любое разработанное и выпущенное может быть выпущено только под той же "бесплатной" концепцией. Это требовало, чтобы весь код был доступен бесплатно в качестве исходного кода. Он разработал GPL (GNUS[1] Гражданскую лицензию), которая заявила, что любое программное обеспечение, разрабатываемое от программного обеспечения GPL, должно было также опубликовываться под лицензией GPL, и, таким образом, быть в свободном доступе. Это вызвало разрыв в общине Unix - те, кто готов работать бесплатно, и те, кто работал с целью получения прибыли.

OSF до сих пор существует и до сих пор производит бесплатное программное обеспечение, в то время как несколько компаний производят и продают свои собственные версии Unix. Контроль AT & T над Unix был передан Novell, а затем позже в Santa Cruz Operation, в то время как другие компании-производители Unix включают Sun (которая производит версию под названием Solaris), Hewlett Packard (который продает HP-UX), IBM (которая продает AIX), и Compaq (которая продает Tru64 Unix). Unix популярен сегодня, но, возможно, не так, как Linux.

ИСТОРИЯ LINUX

Linux очень похож на Unix, но был основан на свободной "игрушечной" ОС под названием Minix. Minix была доступна в качестве образца ОС от автора учебника Эндрю Таненбаума. Университет Хельсинки студент Линус Торвальдс хотел глубоко исследовать ОС, и хотя ему нравилось учиться у Minix, он в конце концов признал ее неудовлетворительной, поэтому он принялся писать свою собственную. 26 августа 1991 года он написал в группу новостей comp.os.minix Usenet, чтобы люди знали о его намерениях и увидеть, если другие могут заинтересоваться. Его первичное заявление указывало, что он разрабатывает свою собственную версию Minix для процессоров Intel 386/486 (клонов IBM

1 GNU расшифровывается как GNU не Unix. Это рекурсивное определение, что на самом деле ничего не значит. Столлман пытается передать то, что операционная система не была Unix, а Unix-подобная операционная система, которая будет состоять исключительно из свободного программного обеспечения.

AT). Он хотел заниматься этим как хобби, и его первоначальное письмо было попыткой вербовки других любителей, которые могли бы быть заинтересованы в игре с ней. Его первоначальная реализация содержала как интерпретатор Bash и C компилятор текущего проекта GNU, GCC. Bash является оболочкой Bourne Again, вариация оригинальной оболочки Bourne, сделанной доступной для Unix. Bash является очень популярной оболочкой, и вы узнаете об этом в главе 9. Он также заявил, что эта первоначальная реализация была полностью свободна от любого кода Minix. Тем не менее, этой ранней версии ОС не хватало драйверов устройств для запуска любого диска, кроме жестких дисков AT.

Торвальдс выпустил версию .01 в сентябре 1991 года и версию .02 в октябре, получив ряд замечаний от пользователей, которые скачали и тестировали неоперившуюся ОС. 5 октября, чтобы объявить о выпуске новой версии Торвальдс опубликовал последующий сообщение для той же группы новостей. Здесь он упоминает, что его операционная система может запускать различные программы Unix, включая Bash, GCC, сделать (мы рассмотрим сделать в главе 13), СЭД (упоминается в главе 10), а также сжатие. Он сделал исходный код доступным для других не только скачать и играть, а работать над кодом, чтобы добиться более широкими возможностями.

К декабрю Торвальдс выпустил версию .10 даже, несмотря на то, что ОС была еще очень сильно сырой. Например, у нее не было возможности входа, загруженного непосредственно в оболочку Bash и только поддерживала жесткие диски IBM AT . Версия .11 поддерживала нескольких устройств и после версии .12, Торвальдс считал, что операционная система была достаточно стабильной и достаточно полезной, чтобы оправдать номер версии .95. Интересно отметить, что на данный момент, Торвальдс услышал в ответ от Таненбаума критику концепции Торвальдса ОС с "монолитным" ядром, отметив, что "Linux является устаревшим." Таненбаум, казалось бы, ошибался в своих комментариях. Выпуская свою первую полномасштабную версию, Линус Торвальдс решил следовать Общей Гражданской Лицензии GNU. Это позволило любому получить бесплатную копию ОС, вносить изменения и публиковать эти изменения бесплатно. Тем не менее, Торвальдс также решил разрешить другим выпускать на рынок версии Linux. Это решение иметь и бесплатную версию (с исходным кодом, доступным для модификации) и коммерческую версию оказывается случайной идеей.

В течение нескольких лет, группа сторонников Linux исчислялась сотнями тысяч. Коммерческие версии были разработаны, в то время как

сообщество пользователей продолжало вносить свой вклад в программное обеспечение, чтобы сделать его более полезным и желательным. Графические пользовательские интерфейсы были добавлены, чтобы сделать Linux более полезной для общего программирования.
На сегодняшний день, как Unix так и Linux являются чрезвычайно популярными форматами ОС. Оба доступны для широкого спектра машин от карманных устройств до ноутбуков и персональных компьютеров до мэйнфреймов суперкомпьютеров. На самом деле, Linux может работать на всех следующих машинах / процессорах: Sparc компании Sun, Alpha компании Compaq, процессорах MIPS, ARM процессорах (во многих портативных устройствах), x86 от Intel / Pentium, процессорах Power-PC, компании Motorola 680x0 процессорах, RS6000 компании IBM, а также других. Одна из причин, почему ОС настолько популярны в том, что большинство серверов в Интернете работают на некоторых версиях Unix или Linux.
Вот несколько интересных фактов о Linux:

• в мире самый мощный суперкомпьютер Sequoia от IBM, работает на Linux, и 446 из 500 лучших суперкомпьютеров работают на Linux.
• Девяносто пять процентов серверов, используемых в голливудских анимационных студиях, работают на Linux.
• Google запускает свои веб-серверы в Linux.
• ОС для Android от Google и Maemo Nokia построены поверх ядра Linux.
• Следует отметить, что 33,8% всех серверов работают на Linux (по состоянию на 2009 г.), тогда как только 7,3% работают на сервере Microsoft.
• Доступное программное обеспечение для Linux включают Mozilla Firefox и OpenOffice, которое является бесплатным и Acrobat Reader и Adobe Flash Player, которые являются частной собственностью.
• Есть более чем 300 ответвлений Linux, применяемых на сегодняшний день.
• Версия Debian Линукса состоит из свыше 280 миллионов строк кода, которые, при разработке на коммерческой основе, будут стоить больше, чем 7 миллиардов долларов; оригинальное ядро Linux (1.0.0) имело 176,250 строк кода.

С другой стороны, по состоянию на январь 2010 года, чуть более 1% всех рабочих столов работали на Linux. Помимо популярности Linux, его кроссплатформенные возможности и возможность получения

версий Linux и прикладного программного обеспечения бесплатно, есть несколько других привлекательных характеристик Linux. Во-первых, Linux является стабильной ОС. Если вы являетесь пользователем Windows, вы, несомненно, столкнулись, с разочарованием, видя крах приложения программного обеспечения без предварительного уведомления и особых причин. Существуют также случаи, в которых необходима перезагрузка Windows, так как операционная система была активна слишком долго (причины необходимости перезагрузки Windows, включают в себя поврежденные страницы ОС и фрагментированного пространства подкачки). Linux практически никогда не нуждается в перезагрузке . На самом деле, Linux только имеет тенденцию останавливаться, если есть сбой оборудования или пользователь закрывает систему. Все, чего не хватает обновлению системы не требует перезагрузки. Кроме того, Linux не подвержен компьютерным вирусам. Хотя большинство вирусов поражают ОС Windows, если кто-то намеревался написать вирус для Linux, то, скорее всего, его ждет неудача, благодаря управлению памятью Linux, которое гарантирует защиту от каких-либо нарушений памяти (смотрите главу 4). * Кроме того, Linux может предоставить более эффективную поддержку программного обеспечения. В качестве эксперимента, Oracle сравнили производительность Oracle 9i при запуске на Windows 2000 по сравнению с Linux. Выполнение Linux на 25-30% эффективнее. По результатам опроса посетителей на LinuxCon 2011, наиболее популярным ответвлением Linux была Ubuntu (34%), а затем Fedora / Red Hat (28%). Из опрошенных почти половина используется Linux дома, на работе, и / или в школе, в то время как 38% использовали Linux на работа и 14% дома.

ОТЛИЧИЯ И РАСПРЕДЕЛЕНИЯ

На рисунке 8.2 представлен график различных релизов между оригинальной ОС Unics через релизы Unix, MacOS и Linux (до 2010 года). Смотри рисунок 8.3 для рассмотрения более значимых релизов дистрибутива. Каковы различия между Unix и Linux? Каковы различия между различными ответвлениями Linux (Debian, Red Hat и т.д.)? Почему существует так много релизов? Почему это так смущает?

Рассмотрим, как Linux и Unix отличаются. Если вы посмотрите на две операционные системы на уровне оболочек, они будут очень похожи. Они имеют почти идентичные команды (например, Ls, CD, RM, PWD), и большинство из этих команд имеют пересекающиеся или идентичные

параметры. У них есть многие из тех же доступных оболочек и наиболее распространенные графические пользовательские интерфейсы (GUI) работают на обеих платформах (KDE и Gnome). Они имеют очень похожие структуры каталогов верхнего уровня. Более того, Linux и Unix, надежны (по сравнению с Windows), портативны для многих различных типов аппаратных средств, и малого размера (по сравнению с ОС типа Windows). Каковы различия тогда?

Хотя Linux[2] является открытым исходным кодом, и многие из программных продуктов, написанных для Linux и Unix с открытым исходным кодом, коммерческие версии Unix не являются открытыми, и большинство версий Unix являются коммерческими продуктами. Существуют стандарты, установленные для различных выпусков Unix, которые помогают поддерживать согласованность между версиями. Это не гарантирует, что программный продукт, написанный для одной версии Unix, будет работать на другой, но это действительно помогает. Для Linux, проект Linux Standards Base начался в 2001 году и является постоянной попыткой помочь создать стандарты для всех Linux релизов. Стандарты описывают то, что должно или не должно содержаться в ответвлениях. Например, стандарт диктует, какая версия компилятора C++ GNUS и какие библиотеки C ++ должны быть включены. В 2011 году компилятор Java был исключен из стандарта. Тем не менее, программисты Linux не должны следовать стандартам Linux Base. Это может привести к приложениям программного обеспечения, которые не могут работать на данном ответвлении, патчи, которые требуются для одного выпуска, а не другого, а также аргументы командной строки, которые работают в одном ответвлении, которые могут не работать на другом. Это может очень разочаровать и программистов, и пользователей.

Другие различия между Linux и Unix являются тонкими и, возможно, не заметными для конечных пользователей. Некоторые различия рассматриваются системными администраторами, так как файлы конфигурации будут отличаться (разные расположения, разные имена, разные форматы). Программист может видеть глубокие различия. Например, ядро Linux редко выполняется в резьбовой форме, тогда как в Unix, потоки ядра являются общими.

Как насчет релизов Linux? Как вы можете видеть на рисунке 8.3, существует множество различных форм, или ответвлений, Linux, а также различные «побочных продуктов», которые много раз выпускались в

2 Linux не претендует на звание свободного от вирусов, но года исследований привели к операционной системе, способной победить многие из механизмов, используемых вирусами для своего распространения.

течение последних 20 лет. Например, как показано на рисунке, SLS / Slackware Linux была использована для разработки S.u.S.E Linux, которая сама по себе была использована для разработки OpenSUSE.
Основные два распределения Linux являются Debian и Red Hat, которые наряду с Slackware (SLS), были самыми ранними дистрибутивами. Хотя на рис 8.3 показаны основные распределения Linux, существуют десятки

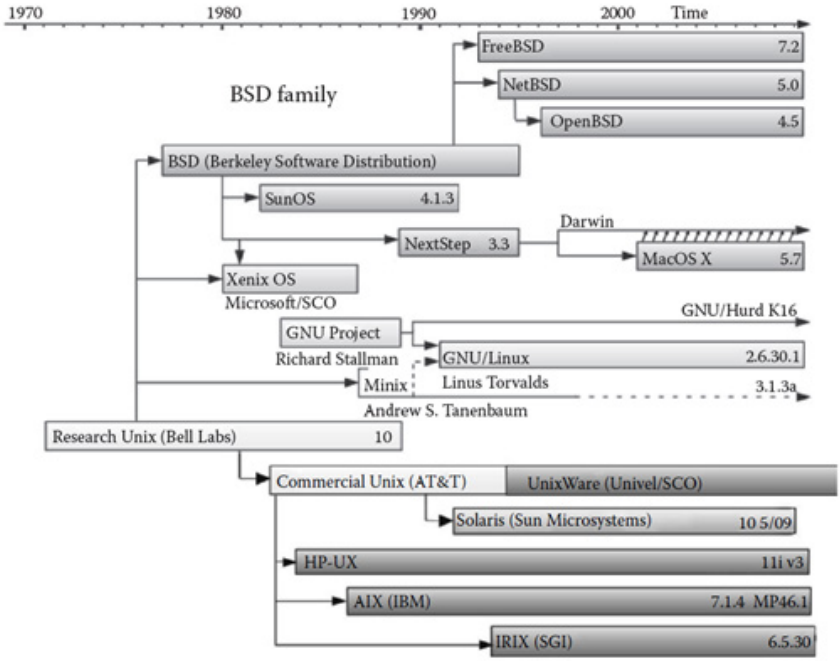
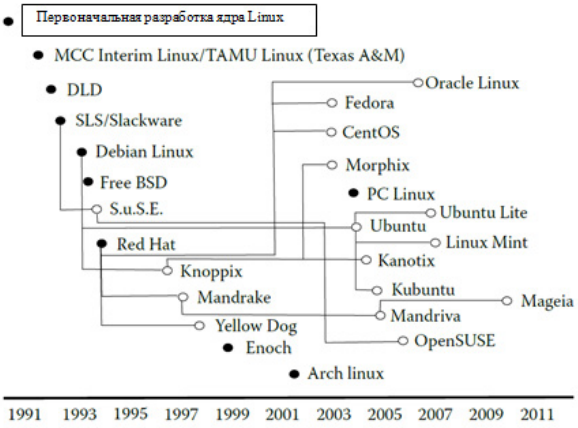


Рисунок 8.2
график времени Unix / Linux. (Фото предоставлено http://commons.wikimedia.org/wiki/File:Unix.png, автор неизвестен.)

Перевод: BSD family – семействоBSD
Free BSD – свободный BSD
NetBSD- сетевой BSD
Open BSD – открытыйBSD
BSD – Программнаялицензия университета Беркли
SunOS – ОС Sun
Next step – следующийшаг
Research UNIX (bell labs) – изучениеОС UNIX (лаборатории BELL)
Commercial UNIX (коммерческаяОС UNIX)

других релизов, которые не показаны.
Есть слишком много различий в различных ответвлениях, чтобы все перечислять здесь. На самом деле, есть сайты, основной причиной существования которых является перечисление таких различий. Здесь мы ограничимся только четырьмя выпусками: Red Hat, Debian, Ubuntu и SUSE.

• Red Hat выпущена в 1994 году Бобом Янгом и Марком Юингом, она является ведущей версией Linux с точки зрения разработки и применения. Его можно найти в любом месте от встраиваемых устройств до интернет-серверов. Red Hat (ОС) является открытым исходным кодом. Тем не менее, Red Hat Enterprise Linux доступна только для покупки, хотя она поставляется с поддержкой. Red Hat поддерживает приложения программного обеспечения с открытым исходным кодом, и в интересном развитии, приобрела собственное программное обеспечение и сделала его доступным с открытым исходным кодом. Некоторые из нововведений Red Hat включают их пакет программного обеспечения для управления, RPM (Red Hat Package Manager), SELinux (защита доступная в Linux), а также программное обеспечение промежуточного слоя JBoss (ПО промежуточного слоя делает возможным мультипроцессорность между прикладным программным обеспечением и операционной системой). Двумя продолжениями из Red Hat являются Fedora и CentOS, оба из которых были впервые выпущены в 2004 году и являются бесплатными



Изображение 8.3
Ключевые релизы ответвлений Linux

и открытыми исходными кодами. CentOS Linux в настоящее время является наиболее популярным ответвлением для веб-серверов.

• Debian-Выпущенный в 1993 году Яном Мердоком, Debian была задумана, чтобы быть полностью некоммерческой, в отличие от Red Hat, для которой организации могут приобрести и получить поддержку. Debian может похвастаться более чем 20000 пакетов программного обеспечения. Debian приводится в качестве наиболее стабильной и наименьшим количеством багов любого Linux. Debian также работает на большем количестве архитектур (процессоров), чем любая другая версия Linux. Тем не менее, эти два преимущества не обязательно могут быть столь же хорошими, как они звучат, так как для поддержки его разработчики ОС более консервативны в своих выпусках. Каждый выпуск ограничен в новых разработках и возможностях. На самом деле, основные релизы Debian выходят гораздо более медленными темпами, чем другие версии Linux (каждые 1 - 3 года). Debian, как Red Hat, может работать во встраиваемых устройствах. Debian представил APT (Advanced Package Tool) для управления программным обеспечением. В отличие от Red Hat, то ОС Debian можно запускать со съемных средств массовой информации (известный как Live CD). Например, можно загрузить Debian с любого компакт-диска или USB-устройства, вместо того, Debian, который установлен на жестком диске. Помимо Ubuntu (смотрите ниже), есть несколько других продолжений Debian, в первую очередь MEPIS, Linux Mint, и Knoppix, все из которых могут быть запущены с загрузочного диска или USB.

• Ubuntu был выпущен в 2004 году Марком Шаттлвортом и основан на неустойчивой версии Debian, но отошел от Debian, так что они не полностью совместимы. В отличие от Debian, Ubuntu имеет регулярные выпуски (6 месяцев) с длительными релизами поддержки, время от времени, (каждые 3 до 5 лет). Ubuntu, больше, чем другие Linux-релизы, выпускает темы для рабочего стола и помощь для пользователей, которые хотят перейти от Windows, к Linux. Доступны 3-D эффекты, и Ubuntu предоставляет отличную поддержку для документации в Wiki-стиле. Шаттлворт является мультимиллионером и делился Ubuntu со всеми, в том числе установочными компакт-дисками бесплатно (эта практика была прекращена в 2011 году). Как и в других выпусках Debian, Ubuntu может быть запущена с загрузочного диска или USB-устройства.

• SUSE-обозначает Software und System-Entwicklung (немецкое название для программного обеспечения и развития систем), она также сокращается, как SuSE. Эта версия Linux основана на Slackware, версии

Linux на основе немецкой версии Softlanding Linux Systems (SLS). Так как SLSвымерла, SuSE это тогда существующий потомок СЛС. Первое ответвление SuSE вышло в 1996 году, а компания SuSE стала крупнейшим дистрибьютором Linux в Германии. SuSE вышла на рынок в США в 1997 году, а в других странах вскоре после этого. В 2003 году Novell приобрела SuSE за $ 210 млн. Хотя SuSE вышла независимо от версий Linux, перечисленных выше, она все же включила аспекты Red Hat Linux, включая менеджера пакетов Red Hat и файловую структуру Red Hat. В 2006 году Novell подписала соглашение с Microsoft, так что SuSE сможет поддерживать программное обеспечение Microsoft Windows. Это соглашение создало некие разногласия с Фондом свободного программного обеспечения (FSF, обсуждается в следующем разделе).

Все еще запутанны? Вероятно, так.

ПРОДВИЖЕНИЕ ОТКРЫТЫХ ИСХОДНЫХ КОДОВ

Разработка Linux является более недавнее событие в истории Unix. Ранее те, кто работал в Unix, следовали двумя разными путями. С одной стороны, люди работали в BSD Unix, версия, которая была разработана в UC Berkeley от работы, финансируемой Агентством оборонных перспективных исследовательских разработок (DARPA). Пользователи BSD со всего мира, но в значительной степени были группой хакеров с Западного побережья, помогли отлаживать, поддерживать и улучшать Unix с течением времени. Примерно в то же время в конце 1970-х или в начале 1980-х годов, Ричард Столлман, исследователь искусственного интеллекта в Массачусетском технологическом институте, разработал свой собственный проект, известный как GNU. * Проект GNU Столлмана заключается в разработке программного обеспечения для ОС Unix таким образом, что программное обеспечение было бы свободным, и не только с точки зрения стоимости, но и с точки зрения способности людей для разработки программного обеспечения в дальнейшем.
Видение Столлмана превратилось в движения свободного программного обеспечения. Идея заключается в том, что любое программное обеспечение, разработанное в рамках этого движения, будет находиться в свободном доступе, и люди будут иметь возможность свободно изменять его. Таким образом, программное обеспечение должно быть доступным в формате исходного кода (то, что было редко, если когда-либо и было). Тот, кто изменил исходный код, может затем мог внести свой вклад в

новый код обратно в проект. Пользователям Unix, которые следовали за этой группой, пытались разработать собственную Unix-подобную операционную систему под названием GNU[3] (или иногда GNU / Linux). Следует отметить, что, хотя проект Столлмана начался еще в 1983 году, стабильная версия GNU никогда не была выпущена.

В 1985 году Столлман основал FSF. FSF, и вся понятие движения свободного программного обеспечения, сочетает в себе и социологические и политические взгляды – пользователи должны иметь свободный доступ к программному обеспечению, чтобы запустить его, чтобы изучить его, чтобы изменить его, и распространять копии с или без изменений.
Это свободное движение программного обеспечения, которое привело к Инициативе Открытого исходного кода (OSI), организации, созданной в 1998 году для поддержки программного обеспечения с открытым исходным кодом. Тем не менее, OSI не то же самое, что и FSF; на самом деле, существуют серьезные разногласия между ними (в основном вызвано самим Столлманом). Под понятием движения свободного программного обеспечения, все программное обеспечение должно быть бесплатным. Продажа программного обеспечения рассматривается как этически и морально неправильно. Само программное обеспечение является реализацией идей, и идеи должны быть доступны для всех; таким образом, программное обеспечение должно быть доступным для всех. Для некоторых, особенно тех, кто начал или участвуют в OSI, эта точка зрения имеет чрезмерно ограничительный характер. Если чьей-то карьерой является создание программного обеспечения, конечно же, человек имеет право зарабатывать деньги на этом. Свободное движение программного обеспечения вместо этого будет поддерживать хакеров, которые будут незаконно копировать и использовать защищенное авторским правом программное обеспечение.
Сообщество с открытым исходным кодом балансирует между крайностями чисто коммерческим программным обеспечением (например, Microsoft) и чисто свободным программным обеспечением (FSF), посредством того, что программное обеспечение сделало вклад в развитие сообщества с открытым исходным кодом и должно оставаться открытым. Тем не менее, тот, кто использует программное обеспечение с открытым исходным кодом для разработки коммерческий продуктов,

3 GNU расшифровывается как GNU не Unix, таким образом, рекурсивное определение, которое, несомненно, применяется к Столлману, так как рекурсия является обычным инструментом для программирования искусственного интеллекта.

должен иметь доступ к свободному праву копирования и продажи его до тех пор, пока вклады, сделанные другими по-прежнему свободно доступны. Это привело к развитию GNU Генеральной общедоступной лицензии (GPL). Хотя GPL[4] была написана Столлманом для его проекта GNU, она на самом деле регулярно используется обществом с открытым кодом. В ней говорится, что данное название программного обеспечения находится в свободном доступе для распространения и модификации, пока что угодно модифицированное сохраняет лицензию GPL. Столлман назвал GPL отсутствием авторского права, вместо авторских прав. GPL, позволило Linux иметь влияние, которое она оказала. Около 50% Red Hat Linux 7.1 содержит GPL, как и многие из программных продуктов, написанных для среды Linux. На самом деле, GPL приходится почти на 65% всех проектов свободного программного обеспечения. *

Сегодня сообщество с открытым исходным кодом процветает. Тысячи людей посвятили время для работы над программным обеспечением и внесли новые идеи, новые компоненты в программном обеспечении, а также новые программные продукты. Без философии с открытым исходным кодом, сомнительно, что Linux достиг бы того успеха, который имеет, потому что не будет никакого глобального сообщества, которое будет делать вклады в новый код. Сообщество с открытым исходным кодом не только внесло свой вклад в Linux, но Linux является самым крупным проектом и наиболее заметным успехом сообщества. Это также привело к тому, что некоторые из компаний пересматривают свои коммерческие подходы. Например, сама Microsoft имеет несколько проектов с открытым исходным кодом.

ИСТОРИЯ WINDOWS

История Windows, в каком-то смысле отражает историю персональных компьютеров, так как Windows, и его предшественник, MS-DOS, были самой популярной ОС для персонального компьютера на протяжении десятилетий. Тем не менее, чтобы полностью понять историю Windows, мы также должны рассмотреть другую операционную систему, Macintosh. Хотя сначала , мы рассмотрим MS-DOS.

MS-DOS был впервые выпущен в 1981 году, чтобы сопровождать выпуск

4 Лицензия GPL, которая обсуждается здесь, на самом деле вторая и третья версии, GPLv2 и GPLv3, выпущенные в 1991 и 2005 годах, соответственно. GPLv1 не обязательно поддерживают идеологию FSF, потому что это позволило людям распространять программное обеспечение в виде двоичного кода, и, таким образом, ограничить некоторые возможности изменить его.

ПК IBM, первого персонального компьютера, выпущенного IBM. IBM решила, при проектировании своего первого персонального компьютера для рынка, использовать открытую архитектуру; они использовали все готовые компоненты, чтобы построить свой компьютер, в том числе микропроцессор Intel 8088. Это, по сути, предложило другим компаниям скопировать архитектуру и выпустить аналогичные компьютеры, которые считали Клонами ПК IBM (или аналоги). Так как эти компьютеры все имели одинаковый процессор, все они могли работать с одними и теми же программами. Таким образом, с доступной MS-DOS для ПК IBM, она также будет работать на клонах ПК. Популярность ПК клонов IBM была в значительной степени потому, что (1) компьютеры были дешевле, чем у многих конкурентов, так как компании (кроме IBM) не должны вкладывать много в развитие, и (2) было произведено больше программного обеспечения для этой платформы, потому что было больше компьютеров такого типа на рынке. По мере роста их популярности, росла и доля MS-DOS на рынке.

ИЗУЧЕНИЕ НЕСКОЛЬКИХ ОПЕРАЦИОННЫХ СИСТЕМ

Как ИТ-профессионал, вы должны знать несколько операционных систем. Тем не менее, вам не нужно покупать несколько компьютеров для решения этой задачи. Здесь мы рассмотрим три подхода к наличию нескольких систем на вашем компьютере.

Первый из них является самым простым. Если у вас есть Apple Macintosh под управлением ОС версии x, вы можете получить UNIX в любое время, когда вы захотите, открыв окно Xterm. Она доступна в соответствии с утилитами. Окно Xterm запускает ядро Mach, которое основано, по меньшей мере, частично, на версиях FreeBSD и NetBSD UNIX. Это, грубо говоря, не UNIX, но это выглядит очень похоже, почти во всех отношениях.

Другой подход, независимо от того, какую платформу компьютера вы используете, это установка виртуальной машины. Виртуальная машина может имитировать большинство других типов машин. Таким образом, вы устанавливаете Vm на вашем компьютере, вы загрузите компьютер в обычном режиме, а затем запустите программное обеспечение Vm, которое загружается на вашей виртуальной машине. Через виртуальную машину, вы можете запускать несколько различных операционных систем. Недостатки такого подхода заключаются в том, что вы должны будете установить ОС, которая может потребовать, чтобы вы приобрели

ее у поставщиков (в то время как Linux, как правило, бесплатна, ОС Windows и Mac такими не являются). Кроме того, виртуальная машина занимает много ресурсов для эффективной работы, и таким образом вы можете обнаружить, что ваша система замедляется при запуске виртуальной машины. Многоядерный процессор устраняет эту проблему. Наконец, если вы используете машину с Windows, и вы не хотите использовать виртуальную машину, то вы ограничены двойной загрузкой. Компьютер с двойной загрузкой — это компьютер, который имеет доступными две (или более) ОС. При загрузке компьютера, загрузчик Windows, может быть приостановлен, и вы можете передать управление загрузчику Linux. Два самых популярных загрузчики являются Lilo (Linux Loader) и GRUB (Grand унифицированный загрузчик). Таким образом, Windows и Linux разделить ваше пространство на жестком диске, и, хотя по умолчанию для загрузки установлен Windows, вы можете перенастроиться для загрузки ОС Linux в любое время. В отличие от Macintosh или подхода виртуальной машины, тем не менее, вам придется закрыть одну ОС, чтобы открыть другую.

DOS была однозадачной, текстовой (не GUI) операционной системой. Команды были в основном те, которые работают в файловой системе: изменение каталогов, создание каталогов, перемещение файлов, копирование файлов, удаление файлов, и запуск / управление программами. В качестве однозадачной операционной системы, операционная система будет работать, оставляя подсказку на экране для пользователя. Пользователь должен ввести указанную команду. Операционная система будет выполнять эту команду.

Если команда пользователя должна была запустить программу, то программа будет работать до тех пор, пока не будет завершена и контроль вернется в напоминании к пользователю. Единственным исключением из этой однозадачной природы было, если программа требует некоторого ввода / вывода, в этом случае программа будет прервана в пользу ОС, или пользователь сделал что-то, что вызвало прерывание, например, набрав ctrl + c или CNTRL + Alt + Del. Большая часть программного обеспечения будет также на основе текста, хотя многие программные продукты, используемые (текстовые) меню, которые могут быть доступны с помощью клавиш со стрелками и некоторые комбинации клавиш управления или функциональных клавиш.

В 1970-е годы, исследователями в Xerox Palo Alto (Xerox Parc) был разработан графический интерфейс для их ОС. Они сделали это, главным образом, чтобы упростить свои собственные исследования в области

искусственного интеллекта, не принимая во внимание достижение, которые привели бы к коммерческому продукту. Стив Джобс и Стив Возняк, люди, которые стоят за компьютерами компании Apple, ездили в Xerox PARC и поняли, что оконная операционная система может существенно повлиять на неоперившийся рынок персональных компьютеров.

Apple, начал разработку новой ОС вскоре после этого. Стив Джобс пошел работать над Apple Lisa, который должен был быть первым персональным компьютером с графическим интерфейсом. Тем не менее, из-за перерасходов развития, проект отставал, и Джобса исключили из проекта. Затем он присоединился к группе Apple Macintosh. Хотя Lisa перегнал Macintosh и появился первым, он имел очень плохие продажи отчасти из-за чрезвычайно высокой стоимости (10 000 $ за единицу). Macintosh будет продан 1 год спустя и сразу прижился. Операционной системой Apple Macintosh считается первой коммерческой на основе графического интерфейса ОС из-за успеха Mac. На стадии разработки Macintosh, Apple наняла Microsoft, чтобы помочь в разработке прикладного программного обеспечения. Билл Гейтс, который занимается разработкой MS-DOS, понял значение в ОС Windows и начал с Microsoft разрабатывать аналогичный продукт.

Хотя MS-DOS была выпущена в 1981 году, он был в стадии разработки в течение большей части 1970-х годов. Во второй половине 1970-х годов, Microsoft начал работать над Windows, первоначально названный менеджер интерфейса. Он должен был располагаться поверх MS-DOS, а это означает, что вы должны сначала загрузить компьютер с MS-DOS, а затем вы запустите GUI поверх него. Из-за своей зависимости от MS-DOS, это означает, что пользователь может на самом деле использовать графический интерфейс, или вернуться к командной строке DOS всякий раз, когда это необходимо. Windows 1.0 был выпущен в 1985 году, через 1 год после выпуска Apple Macintosh. Macintosh не имел строки напоминаний вообще; вы должны были выполнить все, начиная от графического интерфейса пользователя.

Microsoft следуют за Windows 1.0 с Windows 2.0 в 1987 и Windows 3.0 в 1990 году В этот момент, Windows была разработана для запуска на процессоре Intel 80386 (или просто 386). В 1992 году Windows 3.1 был выпущен и в то время, стал самой популярной и широко используемой из всех операционных систем, в первую очередь ориентированных на процессор Intel 486. Все эти ОС были построены на MS-DOS, и все они были мультипрограммными системами в том, что пользователь может

принудительно переключаться из одного процесса (окна) к другому. Это является одной из форм кооперативной многозадачности. Ничто (за исключением версии Windows, 2.1 при запуске со специальным аппаратным и программным обеспечением) не будет выполнять истинную (конкурентную) многозадачность.
Примерно в то же время, Microsoft также выпустила Windows NT, 32-разрядную, сетевую ОС. Помимо того, что она быстрее из-за ее 32-разрядной природы, Windows NT был разработан для клиент-серверных сетей. Это была также первая версия Windows, чтобы поддерживать конкурентную мультизадачность. Windows NT продолжал оказывать существенное влияние на организации.

Следующий 1 шаг в развитии в истории Windows, пришел с выпуском Windows 95. Windows 95 была первой ОС со встроенными возможностями Интернета, объединяя некоторые особенности Windows 3.1 с Windows NT. Как и в Windows NT, Windows 95 также имела конкурентоспособную многозадачность. Еще одно новшество в Windows 95 была способность к готовности работать. Готовность к работе означает, что пользователь может легко устанавливать новое оборудование на лету. Это означает, что компьютер может распознавать устройства по мере их добавления. Windows 95 был разработан для процессоров Intel Pentium.
В быстрой последовательности, несколько новых версий Windows, были выпущены в 1998 (Windows 98), 2000 [Windows 2000 и Windows Millennium Edition (Windows ME)], 2001 Windows XP (версия на основе NT) и 2003 (Windows Server). Кроме того, в 1996 году Microsoft выпустила Windows CE, версию Windows, для мобильных телефонов и других "уменьшенных " устройств (например, устройства системы навигации). В 2007 году Microsoft выпустила оконную систему нового поколения, Vista. Несмотря на то, Vista, содержит ряд новых функций безопасности, она сталкивается с проблемами низкой производительности. И, наконец, в 2009 году Windows 7 была выпущена.
По интересной иронии судьбы, из-за большой доли компании Microsoft на рынке, Apple старается изо всех сил остаться в бизнесе. В 1997 году Microsoft инвестировала 150 миллионов долларов в Apple. В результате оказало существенное влияние на Apple Computers. Во-первых, к 2005 году, Apple Macintosh перешли от процессора питания ПК на основе RISC на процессоры Intel. Во-вторых, из-за изменений, это позволило компьютерам Macintosh запускать большинство программного обеспечения, которое может выполняться в Windows.

Macintosh сохранили свои окна на основе операционной системы, известной как ОС Mac, теперь в версии X. В отличие от Windows, которая первоначально была основана на MS-DOS, Mac OS X построена поверх ОС Unix. Если кто-то желает, можно открыть оболочку Unix и работу с командной строкой. Windows, все еще содержит MS-DOS, чтобы можно было открыть подсказку DOS.

ГЛАВА 9. ОБОЛОЧКА BASH И РЕДАКТИРОВАНИЕ

Для большинства пользователей компьютеров, в том числе вводных студентов ИТ, пользование Linux будет новым опытом. На самом деле, использование командной строки является незнакомой территорией для большинства. Эта глава посвящена оболочки Bash, которая доступна во всех дистрибутивах Linux. В частности, глава охватывает целый ряд особенностей, которые оболочка Bash предлагает сделать командную строку взаимодействия проще. Темы включают редактирование командной строки, использование псевдонимов и список истории. Кроме того, интерпретатор Bash исследуется. Глава завершается рассмотрением двух текстовых редакторов, VI и Emacs.
Цели обучения данной главы являются

• Введение ярлыков в Bash, которые упрощают ввод командной строки.
• Описание роли интерпретатора Bash.
• Введение команды перенаправления Linux.
• Введения механизмов для персонализации оболочки Bash.
• Обсуждение важности текстовых редакторов в Linux.
• Предоставление инструкции о том, как использовать редакторы VI и Emacs.
В этой главе мы сосредоточимся на некоторых конкретных темах Linux. Во-первых, мы рассмотрим оболочку Bash подробно. Мы смотрим на запись командной строки и редактирования, переменные, псевдонимы, и как персонализировать оболочку Bash. Функции, которые мы рассмотрим, все поддерживают пользователя Linux в использовании командной строки, а не графического пользовательского интерфейса (GUI). Мы закончим главу рассмотрением двух общих программ редакторов текстов, используемых в мире Unix / Linux: VI и Emacs. Хотя эти темы редакторов могут выглядеть совсем не связаными с Bash, вы обнаружите, что некоторые из нажатий клавиш, которые вы используете в Emacs такие же, как те, которые вы будете использовать в командной строке для редактирования команд.

ОБОЛОЧКИ

Оболочка является интерпретируемой средой. Это означает, что вы вводите команды и, при нажатии клавиши ввода, эти команды интерпретируются. Интерпретация команды требует, чтобы команда переводится с его входной формы в исполняемую форму. Интерпретированная программа переведена и выполняется строка за строкой, в то время как более традиционная скомпилированная программа переведена полностью, а затем выполняется. Из-за сложности в интерпретатора командной строки Linux, мы можем вводить сложные команды Linux, которые включают в себя несколько команд, перенаправление ввода и вывода, а также сокращенные операции, которые разворачиваются в более сложных инструкциях.

Оболочка Linux / Unix является специализированной. Пользователи могут добавлять операции быстрого доступа, называемые псевдонимы, используйте сокращенные операции, такие как ~ (что указывает на домашний каталог пользователя), определить переменные значения, которые будут использоваться в командах, и даже переопределить внешний вид ввода самого пользователя. Адаптацию окружающей среды можно сделать из командной строки или путем редактирования различных файлов. Есть целый ряд различных вещей, которые могут быть адаптированы, как мы увидим в следующем разделе.

По мере того как оболочка на основе текста, взаимодействуя в оболочке может быть сложной задачей. Большинство пользователей не знакомы с базовыми командами операционной системой (ОС), потому что они только работают с операционной системой через графический интерфейс. С другой стороны, взаимодействие между собой на уровне оболочки, не только обеспечивает большую степень гибкости и контроля над операционной системой, она также экономит ресурсы, потому что сама оболочка не требует, например, сложных графических команд или отслеживания мыши. Можно подумать, оболочки — это более примитивный способ получить доступ к компьютеру, но на самом деле вы общаетесь с компьютером на более низком уровне, потому что вы избегаете слоя GUI.

В Linux / Unix, оболочка автоматически открывается, если вы получаете доступ к компьютеру удаленно без графического интерфейса пользователя. Это, например, тот случай, когда вы используете Telnet или SSH для входа в компьютер. Вы также можете открыть оболочку, запустив программу окна терминала. В CentOS, например, вы можете

открыть окно терминала, когда вы щелкните правой кнопкой мыши на рабочем столе или выберите окно терминала из меню Приложения. Вы также можете создать новое окно терминала из командной строки, например, путем выдачи команды гном-терминал, чтобы открыть окно терминала Gnome, или набрав Xterm, чтобы начать окно X. Оконная система X старше Gnome. Окно терминала Gnome допускает взаимодействие с графическим интерфейсом, так как он имеет ряд меню, в то время как окно Xterm отсутствует (см. Рисунок 9.1). Пользователь может открыть любое количество терминальных окон, каждое является его собственной оболочкой, своя интерпретированная среда. Вы даже можете запускать различные оболочки в каждом окне, хотя оболочки по умолчанию будет Логин пользователя оболочки. Вы также можете начать новый сеанс оболочки из внутри работающей оболочки. В таком случае новая оболочка (то, что мы могли бы именовать как внутренней оболочкой сеанса) перекрывает, что было задано ранее. При выходе из внутренней оболочки, вы возобновите с вашей предыдущей сессии.

Одна из самых ранних Unix оболочек называлась Bourne Shell (sh), впервые выпущенная в 1977 г. Более поздние оболочки включали оболочку C (CSH), оболочка Tenex C, также, так называемая, оболочка T (Tcsh), оболочка Korn (ksh) и оболочка Almquist (ash). Есть целый ряд других, менее известных или малоиспользуемых оболочек, таких как Busybox, оболочки Hamilton C, fish, оболочка Perl и zoidberg.

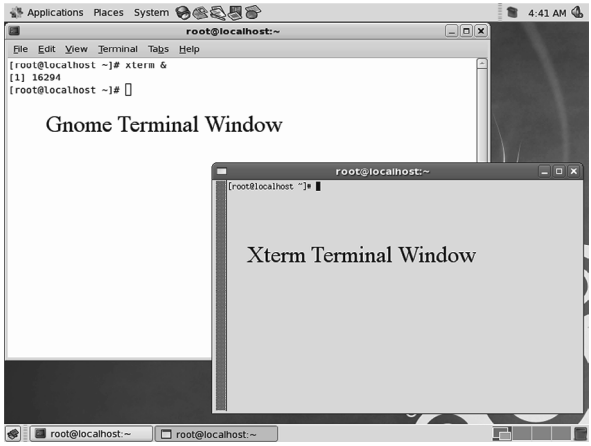


Рисунок 9.1
Терминальные окна

Имена sh, CSH, Tcsh, ksh и ash являются фактическими именами программ оболочки. Можно, например, запустить оболочку C, набрав команду CSH из командной строки. Это создает новую сессию оболочки, как описано в предыдущем абзаце. Оболочки Bash (bash) на самом деле известна как Bourne Again Shell, на основе оболочки Bourne. Мы будем называть ее просто Bash. В следующих трех разделах мы рассмотрим оболочку Bash подробно. Следует отметить, что многие из функций, доступных в Bash, также доступны во многих других оболочках, перечисленных выше.

ФУНКЦИИ РЕДАКТИРОВАНИЯ ОБОЛОЧКИ BASH

Оболочка Bash была написана в качестве замены для оболочки Bourne. Bash была написана, чтобы сопровождать GNU ОС, попытка сообщества с открытым исходным кодом для построения Unix-подобной операционной системы. Позднее она был использована в качестве оболочки по умолчанию для Linux и ОС Mac X. Bash имеет многие из тех же функций из оригинальной оболочки Bourne, но добавляет функции, которые были созданы для таких оболочек, как CSH и KSH. Это также упрощает некоторые детали синтаксиса, необходимые в оболочке Bourne, так что, несмотря на то, что эта оболочка больше, ее проще использовать. В этом разделе мы рассмотрим несколько функций, доступных в Bash, которые делают ввод и редактирование команд проще. Это

- История
- Редактирование командной строки
- завершение Tab
- Псевдонимы (alias)
- раскрытие скобок
- тильды
- Символы

В следующем разделе мы рассмотрим другие особенности оболочки Bash.

Каждый раз, когда вы вводите команду в текущем сеансе Bash, он записывается в том, что называется списком истории. Список истории доступен , так что вы можете легко вспомнить ранее введенную команду. Для просмотра списка истории, введите команду history. Рисунок 9.2 показывает пример списка истории. История впервые была доступна в CSH, но оказалась настолько полезной и популярной, что она был с тех

пор добавлена в большинстве оболочек.
Есть множество способов, чтобы вспомнить команду из списка истории. Из командной строки, набрав клавишу со стрелкой вверх или ctrl + p [1] будут извлекать предыдущую команду. Продолжая нажимать клавишу вверх или ctrl + p будете продолжать шагать назад по списку истории. Так, например, если вы ввели 10 команд, показанных на рисунке 9.2, нажимая CNTRL + P три раза, оставляете "PWD" в командной строке. Нажатие Enter затем выполняет эту команду. Для того, чтобы двигаться вперед по списку истории, как только вы сделали шаг назад, нажмите клавишу со стрелкой вниз или ctrl + n (p для перехода к предыдущему, n для следующего).
Другой способ вспомнить команду из списка истории путем набора на клавиатуре !number, где number является номером записи в списке истории (восклицательный знак часто называют взрывом). Например, !2 выполнит "ls / home" команду из рисунка 9.2. Чтобы повторить последнюю команду, введите !! (ПИФ-паф). Вы также можете использовать! Указав !chars, где chars являются первыми символами команды, которую вы

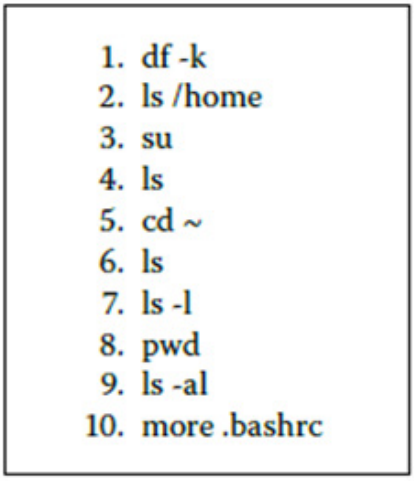


Рисунок 9.2
Пример списка истории в Linux.

1 Это означает, что нажать кнопку управления и, удерживая ее, нажмите клавишу 'P'. Это обозначение будет использоваться в этой главе наряду с escape + key, а это значит, нажмите клавишу Escape (не удерживайте ее нажатой) и нажмите клавишу "key". Различие между ctrl и escape является то, что вы держите нажатой клавишу ctrl, но вы только нажимаете и отпускаете клавишу escape.

хотите повторить.
Будем считать, что команды 11 по списку истории из рис 9.2 являются cp * / home / zap-paf. Если вы затем введете !c, она будет выполнять последнюю команду, которая начинается на 'C', команды cp. Если вы вводите !cd, она будет выполнять последнюю команду, которая начинается на 'CD', которая будет инструкция 5, CD ~. Точно так же, !l (маленькая буква 'L') повторяет инструкцию 9 (ls -al), как это делает !ls и !ls- (T.e. с последующим пространством после дефиса Ls). Если мы введем !ls/, она будет повторять вторую команду в списке (ls / home).
Использование ctrl + p и ctrl + n являются двумя из многих доступных форм редактирования командной строки. Концепция редактирования командной строки впервые был представлен в Tcsh, где команды были похожи на команды редактирования, изобретенные в Emacs. Редактирование командной строки осуществляется на текст, который в настоящее время находится в командной строке. Используя комбинацию ctrl или escape и другие клавиши, вы можете перемещать курсор по командной строке, вырезать символы, и / или вставить символы. Ниже приведен список более полезных операций редактирования. Обозначения здесь c + char или m + char, где c + char означает удерживать клавишу ctrl и нажмите символ (например, c + n означает "нажмите кнопку ctrl и, удерживая её нажатой, нажмите N"), и m + char означает нажать клавишу escape за которым следует символ (например, m + f означает "нажать клавишу escape, а затем нажмите кнопку F"). Причина, почему мы используем 'm' для escape является то, что некоторые люди относятся к клавише escape в качестве клавиши "meta".

• c + n - отобразить следующую команду из списка истории в командной строке
• c + p - дисплей предыдущую команду из списка истории в командной строке
• c + f - переместить курсор вперед на один символ
• c + b - перемещение курсора в обратном направлении на один символ
• m + f - перемещение курсора на одно слово вперед
• m + b - перемещение курсора на одно слово назад
• c + a - переместить курсор в начало строки
• c + e - переместить курсор в конец строки
• c + d - удалить символ, где находится курсор
• c + k - убить (удалить) содержимое от курсора до конца строки
• c + w - удалить все символы перед курсором в начале слова

• c + y - взять все, что совсем недавно было удалено (с помощью C + W или C + k) и вернуть его обратно, помещая в текущую позицию курсора (это операция вставки)
• c + _- отменить последнюю операцию редактирования (удаления или вставки)

И, конечно же, вы можете использовать Backspace, чтобы удалить предыдущий символ,del удалить текущий символ, клавиши со стрелками влево и вправо для перемещения вперед и назад, а также клавиши home и end, чтобы перейти к началу и конец строки.
После того, как вы запомните эти сочетания клавиш редактирования, ввода команд из командной строки становится, по крайней мере, немного легче. Например, представьте, что вы хотите скопировать пять файлов из текущего каталога в каталог /home/zappaf, изменяя каждое имя файла. Пять файлов называются file1.txt, file2.txt и т.д., и вы хотите переименовать их zappa1.txt, zappa2.txt и т.д. Первая команда будет введена без использования редактирования:

cp file1.txt /home/zappaf/zappa1.txt <enter>

Остальные четыре команды могут быть легко введены путем редактирования этой строки. Нажмите c + p, чтобы получить команду. Нажмите c + a, чтобы перейти к началу строки. Нажмите m + f дважды, чтобы двигаться вперед на два слова. Курсор теперь в промежутке между file1 и TXT. Нажмите backspace для удаления 1 и нажмите 2, чтобы вставить 2. Теперь, нажмите c + e, чтобы добраться до конца строки. Нажмите m + b, чтобы переместить вас к началу “TXT” в zappa1.txt. Нажмите c + b, чтобы перейти к точке, Backspace, чтобы удалить 1 и 2 для вставки 2. Команда теперь:

cp file2.txt /home/zappaf/zappa2.txt

Теперь, нажмите <enter> для ввода этой измененной команды. Затем вы повторите описанные выше шаги, чтобы повторить эту инструкцию для других файлов.
Еще одна очень полезная функция редактирования Bash называется автодополнение. По мере ввода названия каталога или имени файла, если вы дошли до части этого имени, которое является уникальным, нажатие клавиши табуляции заставит интерпретатор завершить имя

для вас. Рассмотрим, например, что вы находитесь в рабочем каталоге, который имеет следующие файлы:

forgotten.txt frank.txt fresh.txt functions.txt funny.txt other_stuff.txt

Вы хотите, просмотреть один из этих файлов, который меньше используется.[2] Если вы наберете less fo <Tab>, интерпретатор Bash обнаружит, что только forgotten.txt начинается с "fo", и поэтому завершает его за вас. Если вместо этого вы набираете less fr <Tab>, Bash не может выполнить это, потому что неясно, хотите ли вы frank.txt или fresh.txt. В таком случае, интерпретатор подает звуковой сигнал, чтобы вы знали , что имя файла не является уникальным. Если вы должны были набрать less fra <Tab>, то она может быть завершена, чтобы быть frank.txt, или если вы набрали less fre <Tab>, она может быть завершена, как fresh.txt. Нажатие tab дважды вызовет интерпретатора Bash, чтобы перечислить все соответствующие имена файлов (или, если есть больше, чем несколько файлов, то вы получите сообщение, или все файлы должны быть перечислены). Таким образом, ввод less fr <Tab> <Tab> приводит к тому, что Bash откроет список frank.txt и fresh.txt.
В Linux, alias является командой быстрого доступа. Вы определяете псевдоним, так что, когда вы вводите ваш alias, он заменяется полной командой. alias можно вводить в любое время в командной строке, или они могут быть определены в других файлах (мы рассмотрим использование определений этих файлов в Персонализации оболочки Bash). Форма определения alias

alias name = команда

где alias name это имя ярлыка и это команда Linux, которая заменит имя. Простой пример:

alias add = / USR / SBIN / useradd

2 Чем меньше команда используется для отображения содержимого файла в окне терминала, один экран за раз. В конце каждого экрана, вы можете двигаться вперед на один, или вперед или назад на одну строку, или меньше и вернуться к командной строке. Команда less, наряду с more и cat, удобные способы просмотра содержимого файла, но less удобнее, чем more или cat.

alias add будет использоваться для замены более длинной команды. Если команда Linux содержит пробелы, то команда должна быть помещена внутри одиночных кавычек (' '). Именем может быть любая строка символов (включая цифры и знаки препинания) до тех пор, как выбранные символы не зарезервированы для других целей Linux (например, %m может быть использовано в качестве псевдонима, но -m не может).
В большинстве случаев вы определяете alias для упрощения ввода. Приведенный выше пример позволяет сократить команду к чему-то проще набрать. Другим примером может быть, чтобы уменьшить длину команды, которая содержит опции. Например, если вы предпочитаете, чтобы ваши команды ls всегда появляются в качестве длинного списка, вы могли бы определить следующий alias:

alias ls = 'ls -l'

Обратите внимание на использование кавычек, потому что пространства между ls -l. Также обратите внимание, что имя alias может быть имя уже существующей команды.
alias может быть также полезным, если вы не можете вспомнить, как использовать команду. Например, в случае useradd, вы можете забыть, где она хранится. Или, в случае команда, которая имеет несколько вариантов, вы можете забыть, какие из них использовать. В случае rm, вы можете забыть использовать опцию -i, но для безопасности, полезно всегда использовать rm-i. Таким образом, вы можете определить следующий alias.

Alias rm = ‘rm –i’

Сама команда ls не требует параметров, так что левая шкала раскрывается ls -l. Но и rm и useradd требуют параметров. Как параметры взаимодействуют с псевдонимом? Например, если мы хотим удалить foo.txt, мы должны ввести rm foo.txt. Но теперь rm является псевдонимом для 'rm -i'. На самом деле, замена псевдонима работает, как можно было бы ожидать. Псевдоним развертывается от rm rm-i с foo.txt завершения инструкции. Таким образом, rm foo.txt становится rm -i foo.txt. Аналогичным образом добавьте -m -G cit130 bearb будет разворачиваться в / USR / SBIN / useradd -m -G cit130 bearb. Псевдоним просто заменяется перед выполнением команды. С псевдонима rm можно указать даже дополнительные опции. Например, если вы хотите сделать

рекурсивную (rm -r), команда rm -r / Foo разворачивается, чтобы стать rm -i -r / Foo.
Псевдоним также может быть настроен для обработки общих опечаток, которые может сделать пользователь. Представьте себе, что, будучи быстрой, но не очень осторожной машинисткой, вы часто переставляете команды из двух букв. Таким образом, mv может выйти, как vm, rm как mr и cp как pc. Двухбуквенные комбинации vm, mr и pc не команды Linux, и введение любой из них приведет к сообщению об ошибке в командной строке (команда не найдена). Можно определить псевдоним для каждого из них:

Alias mr = rm
Alias pc = cp
Alias vm = mv

Обратите внимание, что в этом случае кавычки не требуются, поскольку команда не включает в себя пространство. Комбинируя эту идею с предыдущим примером rm, мы могли бы определить эти псевдонимы:

alias rm = 'rm -i'
alias mr = 'rm -i'

Ниже приведены несколько других интересных идей для alias.

```
alias..='cd ..'                          A shortcut to move up a level
alias...='cd ../..'                      A shortcut to move up two levels
alias md=mkdir                           A shortcut to create a directory
alias egrep='grep -E'                    Enforcing a common option in grep
alias h='history 10'                     A shortcut to display a partial history
alias xt='xterm -bg black -fg white &'   A shortcut to generate a terminal window
```

Если вы определяете псевдоним в командной строке, псевдоним существует только в этой сессии оболочки. Он не будет существовать в других терминальных окнах, ни если вы начали новый сеанс оболочки (набрав Bash, например). По этой причине, он является общим для сохранения псевдонимов. Мы обсудим, как сделать это в Персонализации оболочки Bash.
Помимо коротких команд, предлагаемых псевдонимами, доступны два других типа коротких команд через тильды и выражений в фигурных скобках. Вы уже видели, как тильды (~) используется для обозначения

домашнего каталога пользователя. Переводчик Bash заменяет ~ с домашней директории пользователя, поэтому, например, less ~ / foo. txt становится less /home/foxr/foo.txt. Расширение Тильда может также использоваться для обозначения домашних каталогов других пользователей с обозначениями ~ Имя пользователя. Например, вы можете использовать less ~ zappaf / foo.txt, чтобы просмотреть файл в домашнем каталоге zappaf.

Расширение скобок используется, когда у вас есть список элементов, которые команда должна применять. Фигурные скобки,{}, будут содержать список, где каждый пункт в списке идет через запятую. В качестве простого примера, представьте, что вы хотите создать несколько ls подкаталогов в вашем домашнем каталоге. В вашем домашнем каталоге есть foo1, foo2 и foo3, где foo2 содержит два подкаталога, foo2a и foo2b. Можно выпустить пять отдельных ls команд (по одноq для Foo1, foo2, foo2a, foo2b, foo3). Кроме того, можно перечислить каждый из них в одной команде ls: ls foo1 foo2 foo2 / foo2a foo2 / foo2b foo3. Тем не менее, обратите внимание, что для foo2a и foo2b, вы должны перечислить их как foo2 / foo2a и foo2 / foo2b. Вы могли бы вместо этого использовать расширение скобок, где вы перечисляете все каталоги внутри скобок. Поддиректории могут быть аналогичным образом перечислены во внутреннем (или вложенном) наборе скобок. Команда становится: ls {Foo1, foo2, foo2 / {foo2a, foo2b}, foo3}, который интерпретатор Bash разворачивает в пять отдельных команд, ls ls Foo1, ls foo2, ls foo2 / foo2a, ls foo2 / foo2b и ls foo3.

Другой формой расширения называется расширением имени файла. Это происходит, когда вы используете символы подстановки среди ваших имен файлов. Подстановочная, так же, как в покере, может действовать, как угодно. Наиболее распространенный символ подстановки является *, который по своей сути означает "соответствовать чему-нибудь". Вы бы использовали * при выполнении системных команд файлов, такие как ls. Например, ls * .txt будет список всех файлов, которые заканчиваются с расширением .txt, независимо от того, каким было их название. Если текущий каталог содержит файлы foo1.txt, foo2.txt, foo3.text и bar.doc, * .txt обозначение будет соответствовать foo1.txt и foo2.txt. Оно не будет соответствовать foo3.text, потому что txt не то же самое, что и текст, и он не будет соответствовать bar.doc, потому что документ не соответствует txt. Тем не менее, команде ls *.* будут соответствовать все четыре из них. С другой стороны, если каталог содержит также файл stuff, ls *.* не будет соответствовать ему, потому что нет ни одной точки stuff. Команда

ls * будет соответствовать всем пяти файлам (четыре с точкой и stuff).
Использование группового символа является обыкновенным для пользователей Linux. Есть и другие подстановочные знаки, такие как ?, Что означает "любой одиночный символ". Мы будем исследовать подстановочные знаки более подробно в следующей главе, когда мы рассмотрим регулярные выражения.
Все различные методы редактирования командной строки предоставляют пользователю мощный набор коротких команд, так что работа с командной строкой проще. Тем не менее, за немногими исключениями, все они требуют практики, ни по какой другой причине, чем запомнить, что они существуют. Начальные пользователи Linux часто отказываются использовать эти методы, поскольку они не относятся к интуитивным студентам, которые только учат Linux. С другой стороны, изучая рано, это позволяет пользователям быть более знающими и, следовательно, более эффективными в Linux.

ИЗУЧЕНИЕ ИНТЕРПРЕТАТОРА BASH

Теперь, когда мы узнали некоторые из коротких команд и функций редактирования, доступных в оболочке Bash, мы переходим к тому, как работает интерпретатор Bash. Введем сначала три дополнительных инструментов, доступных для пользователя: переменные, перенаправления и получения справки.
Переменные доступны в оболочке Bash. Переменная это имя, данное в хранилище местоположениях. Используя переменную, вы можете ссылаться на эту переменную из различных команд, вместо того, чтобы использовать значение напрямую. Простым примером может быть переменной, которая хранит количество лицензий, доступных для названия программного обеспечения. По мере того как пользователи запускают программное обеспечение, эта переменная уменьшается, и, как пользователи прекращают работу программного обеспечения, переменная увеличивается. Если пользователь желает запустить программное обеспечение и значение равно 0, то они говорят, что нет свободных лицензий в настоящее время и что они должны ждать. В этом случае переменная является общим для всех пользователей. В сессии Bash переменная доступна для всего программного обеспечения, работающего в этой оболочке. Существуют два типа переменных Bash. Пользовательские переменные, определенные пользователем и часто используется только с помощью операций командной строки. Есть также

переменные окружения, установленные либо ОС, интерпретатора Bash, или каким-либо другим запущенным ПО. Такие переменные, как правило, доступны для всех программных продуктов. Оболочка Bash определяет свои собственные переменные окружения, многие из которых могли бы быть полезными для пользователя. Чтобы выяснить, какие переменные окружения были установлены, введите команду env. Среди переменных, определенных, вы, вероятно, увидеть такие вещи, как HOSTNAME (имя компьютера), SHELL (текущей оболочки, который для нас be / bin / bash) USER (своё имя пользователя), HOME (домашний каталог пользователя) и PWD (текущий рабочий каталог). Обратите внимание на то, что все эти переменные окружения полностью прописными буквами. Это соглашение, так что пользователь может легко определить, если что-то это переменная окружения, или что-то другое (такая определяемая пользователем переменная или псевдоним).

Для того, чтобы установить свою собственную переменную, просто введите оператор присваивания в командной строке. Формат оператора присваивания variable = value, где variable является именем переменной и value является значение. Значения по умолчанию оболочки Bash для хранения строк, так что значение должно быть строками, такие как имя человека. Если строка имеет пустое пространство, заключите всю строку в кавычки ("" или '', мы будем различать их позже). Значение может быть целым числом; Тем не менее, Bash будет рассматривать любое число в виде строки, если не указано, что оно должно быть числом. Это делается путем ограждающих значений внутри скобок, так как в возрасте = (29).

Чтобы получить значение, хранящееся в переменной, вы должны предшествовать переменной имя с $. Например, если first = Frank и last = Zappa, и вы хотите создать переменную, полное имя, вы могли бы выдать оператор присваивания FULLNAME = "$ first $ last". Переменная FullName затем устанавливается как значение, хранимое в переменной первой, за которым следует пробел, а затем значение, хранимое в переменной Last. Если вы должны были сделать FullName = "first last", то FullName буквально название "first last", а не значения хранятся в этих переменных. Обратите внимание на то, что кавычки необходимы из-за пустого пространства. Если вы не заботитесь о пустом месте, вы можете также сделать FullName = $ first $ last, а также FULLNAME = "$ first $ last".

Если значение в переменной является числом, и вы хотите выполнить некоторый тип арифметической операции на нем, вы должны вложить всю операцию в $ ((...)). Рассмотрим, если, после того, как сохраним 29 в

возрасте, как показано выше, вы делаете NewAge = $ age + 1. Вы получаете 30 в новом возрасте? Нет; на самом деле, вы получаете 29 + 1. Почему? В буквальном смысле слова, команда присоединяет элементы, значение, хранящееся в переменной возраста (29) + и значение 1. Концентрация сочетает в себе строки вместе. Но вот, мы хотим добавить 1 к возрасту. Для этого потребуется использование данной ((...)), так что правильная инструкция NewAge = $ ((age + 1)). Обратите внимание на то, что $ предшествует ((...)) и не появляется непосредственно перед переменной возраста. Для тех из вас, кто программировали раньше, вы, вероятно, знакомы с переназначением заявления, принимая значение переменной, изменяя ее, и переназначение переменной на новое значение. Таким образом, вместо NewAge = $ ((age + 1)), вы можете также сделать age = $ ((age + 1)).
Если у вас есть два числовых значений, скажем, X и Y, вы могли бы сделать Z = $ ((X + Y)). Арифметические операции Доступные +, -, * (умножение), / (деление) и % (остаток, также известный как мода или по модулю).
Помимо назначения переменных и их использование в операторах присваивания, вы можете также просмотреть содержимое переменных с помощью команды echo. Команда echo ожидает список элементов. Что бы ни было в этом списке, выводится. Например,

echo Привет, как ты?

буквальный выход "Привет, как ты?" на следующей строке окна терминала. Обратите внимание на то, что вам не нужно размещать "" по всему списку. Вы можете также вывести значения, которые сохраняются в переменных, как в эхо $ FULLNAME или echo $age. Вы можете комбинировать необработанный текст с варьируемыми значениями как в echo Hello $FULLNAME. Эхо допускает утверждение, но не требует, чтобы вы помещали список элементов в кавычки. Вы можете использовать "" или ", хотя есть одна большая разница. Если вы используете " и помещаете имя переменной внутри одиночных кавычек, вы получите имя переменной, а не значение. Так, например, echo "Hello $ FullName" выводит Привет Фрэнк Заппа (предполагая, что FullName хранит Frank Zappa), в то время как эхо 'Hello $ FullName' выводит Hello $ FULLNAME. И, конечно же, эхо "FULLNAME Hello" будет просто выходной Привет FullName, потому что не было $ перед переменной.
Последнее замечание по поводу echo заявления. Представьте, что вы

хотели выводить приветствие пользователя и текущей даты и времени. Linux имеет команду date. Если вы должны были использовать

echo "Привет $ FullName, сегодня date"

вы буквально получить Hello Frank Zappa, сегодня дата. Разместив команду Linux внутри `` (обратная кавычки), то команда будет выполняться до echo-заявления. Так,

echo "Привет $ FullName, сегодня` date` "

будет выводить что-то вроде Hello Frank Zappa, сегодня чт 9 февраля 17:54:03 EST 2012 (естественуj, в зависимости от времени и даты, когда команда была введена). Команда, помещается в `` может также появиться в следующем формате: $ (команда). Предыдущая команда эхо может вместо того, чтобы быть

усрщ "Привет $ FullName, сегодня $ (date)"

Это не особо принято, что пользователю нужно будет определять свои собственные переменные для таких целей, как хранения вашего имени или возраста. Пользователи могут определять переменные, которые могут быть использованы в программном обеспечении, что они пишут. Чаще всего, хотя, переменные определяются с помощью программного обеспечения и используются программным обеспечением. Есть несколько установленных переменных среды в Linux. Мы видели некоторые из них раньше, когда мы говорили о команде env. Вот другие примечания.

• DISPLAY-имя Linux для стандартного вывода (по умолчанию: 0.0).
• HISTSIZE-количество команд, которые могут быть сохранены в списке истории.
• PATH-список каталогов, которые проверяются каждый раз, когда пользователь вводит команду; Это позволяет пользователю выполнять некоторые программы или найти некоторые файлы без необходимости указывать весь путь.
• PS1-определяет запрос пользователя, который может включать в себя специальные символы, представляющие имя пользователя или текущий каталог, или номер команды, так как он будет сохранен в списке истории (таблица 9.1)

• PS2, PS3, PS4, и могут быть использованы для определения других подсказок, которые используются в различных программах.

Из переменных, перечисленных выше, вы, скорее всего, только измените PATH. Переменная PATH хранит список каталогов. Всякий раз, когда вы вводите команду или имя файла Linux, если элемент не найден в текущем рабочем каталоге, то интерпретатор проверяет Linux для элемента в каждой директории, указанной в переменной PATH. Только если он не будет найден в текущем рабочем каталоге или любом из каталогов PATH не выводит сообщение об ошибке.

Таблица 9.1 Символы для переменной PS1 для специализации подсказок пользователя

Символ	Значение
\ d	Текущая дата
\d {format}	Текущая дата, используя данный формат, где формат соответствует формату ожидаемой командой STRFTIME
\t	Текущее время в военном формате
\@	Текущее время в 12-часовом формате (AM / PM)
\H, \ h	Полное имя хоста компьютера или первая часть имени хоста
\s	Название оболочки
\u	имя пользователя
\w	Текущий рабочий каталог
\!	Текущий номер команды (как он будет отображаться в списке истории)
$	По умолчанию пользователь спецификатор (быстрый)
@	По умолчанию разделитель, например, \ d @ \ u $
\$?	Статус последней команды
\e [... \e [m	Изменить цвет строки, то ... это код цвета в виде #; ##, где первый # либо 0 (светлый) или 1 (темный), а второй ## является цвет (30 для черного, 31 для красного, 32 для зеленого, 33 для коричневого цвета, 34 для синего, 35 фиолетовый, 36 для голубого цвета)

В оболочке Bash, переменная PATH устанавливается изначально

системным администратором, типичное хранение / USR / kerberos / bin: / USR / local / bin: / USR / bin: / bin: / USR / X11R6 / bin. ":" Используется для разделения каждого пути. Вы должны признать, что многие из этих путей (например, / USR / local / BIN, / USR / bin, / bin). Kerberos используется для сетевой аутентификации и программы, находящиеся в каталоге bin общие сетевые программы (например, FTP, Telnet, Rlogin). X11R6 хранит каталог X11 (оконные) файлы.
Если вы хотите добавить в переменную PATH, вы могли бы выдать переназначение заявление следующим образом:

PATH = $ PATH: newdirectory и

Здесь вы устанавливаете PATH равным тому, что в настоящее время в переменной PATH, за которым следует: и новый каталог. Так, PATH = $ PATH: / home / foxr будет add / Home / домашний каталог foxr в на вашем пути. Вы можете сделать это с вашей собственной домашней директорией, так что, независимо от того, где вы работаете в файловой системе Linux, ваш домашний каталог будет протестирован вместе с остальными PATH каталогов для файла. Вы бы, конечно, заменить собственное имя пользователя для foxr-x годов.
В PS1 переменная хранит, как выглядит ваш запрос. Изменяя PS1, вы измените внешний вид вашей подсказки. Если вы не знаете, что вы делаете, то лучше оставить в покое PS1. Кроме того, существует мало, нужно изменить другие переменные окружения.
Интерпретатор Bash, как правило, принимает ввод с клавиатуры и посылает вывод в окне терминала. Тем не менее, бывают случаи, когда вы хотите перенаправить либо ввод или вывод. Например, вы можете захотеть изменить входные данные для программы исходить из файла, а не в командной строке. Или, вы можете отправить вывод в файл вместо дисплея. Третья возможность заключается в том, что вы можете взять вывод одной команды и использовать его в качестве входных данных для другой команды. Все эти формы перенаправления.
Перенаправление производится с использованием одного из следующих пяти наборов символов:

• <перенаправить ввод приходить из входного файла
•> перенаправить вывод пойти в выходной файл, перезаписать файл, если он уже существует
• >> перенаправить вывод, который будет добавлен к уже существующему

файлу, или создать файл, если он не существует
• << перенаправить ввод на клавиатуре, где ввод прекращается со специальным ключевым словом , указанный после <<
• | перенаправить вывод одной команды, чтобы быть входом другой - это известно как труба

Первые четыре экземпляра перенаправляют файл чтобы просто переопределить стандартный ввод и / или выходной. Например, стандартный выход для кошки является экран. Набирая cat foo1.txt выход будет содержимым foo1.txt к монитору. Это похоже на команды more и less, но на выходе появляется все сразу, прокрутка вниз и за пределы экрана, так что вы только прокручиваете экран, чтобы увидеть последнее значение. В более и менее, выходной сигнал паузы в конце каждого экрана. При использовании cat, вы можете решить, взять выходной и послал его в другой файл. Вы бы сделать это с помощью перенаправления, как

cat foo1.txt> foo2.txt

Выше инструкция дает тот же результат, как если бы вы сделали Ср foo1.txt foo2.txt. Тем не менее, представьте, что вы хотите взять вывод нескольких файлов и скопировать их в другой файл.
Команда ср не делает этого. Таким образом, вы можете использовать cat и переназначение, как в

cat file1.txt file2.txt file3.txt> file4.txt

Файл file4.txt затем будет конкатенация file1.txt, file2.txt и file3.txt. Если file4. txt уже существует, то эта команда перезаписывает. Если вы должны были использовать >> вместо>, то либо три файла будут скопированы в новый файл file4.txt если file4.txt ранее не существовало, или они будут добавлены к концу file4.txt если file4. TXT действительно существуют.
Перенаправления ввода < просты в использовании. По сути, вы хотите заменить ввод с клавиатуры, чтобы прийти из файла вместо этого. Это полезно только, если ожидаемый ввод приходит с клавиатуры, а не файл. Так, например, программа wc (количество слов) не требует, < так как типичная форма обучения является туалетом файла. Наиболее распространенное использование перенаправления ввода будет, когда вы пишете свои собственные сценарии оболочки. Этот вопрос будет

обсуждаться в главе 14. Сейчас мы будем игнорировать, использование <.
Другая форма перенаправления ввода, <<, в некотором смысле является противоположностью <. Где <говорится: "принимать входные данные из этого файла, а не стандартного ввода", << говорится: "принимать входные данные от стандартного ввода, а не файла ". << Используйте , когда вы хотите заставить программу, которая принимает входные данные из файла, чтобы принять ввод с клавиатуры вместо этого. При использовании <<, то программа сделает паузу, чтобы позволить вам ввести элементы. Каким образом программа знает, когда ваш вклад будет сделано? Это не просто нажав клавишу <Enter>, потому что вы можете иметь разрывы строк в вашем входе. Вместо этого, после << символов, вы поместите ключевое слово. Именно это ключевое слово, которое завершит ввод. Общий пример использования <<, чтобы создать файл с помощью cat. Обозначение для cat, как правило, cat имя файла (ов). Тем не менее, если вы выполните cat << ключевое слово, а затем ввод поступает с клавиатуры, пока вы не ввели ключевое слово. Примите во внимание следующее:

cat << done

Это приведет к появлению подсказки, >. В командной строке, вы начинаете печатать. Вы можете ввести все, что вы хотите, в том числе нажав клавишу <Enter>. Программа cat продолжает принимать введенные данные, пока вы не вводите сделать. В этот момент cat получил весь его вход, и теперь он выполняет, что приводит к отображения содержимого ввода на стандартный вывод, то есть то, что вы только что ввели в систему, на экране появляется сообщение. Это не очень полезно. Тем не менее, вы можете перенаправить вывод. Если вы должны были сделать:

cat << done> shoppinglist.txt

и вы вводите в список покупок, заканчивая словом "Done", то все, что вы набрали, теперь хранится в файле shoppinglist.txt. Таким образом, от cat, и с помощью перенаправления, вы можете создавать свои собственные TextFiles. Попробуйте следующее, где> знаки отображаются в окне терминала:

cat << done> shoppinglist.txt

> бананы
> яблоки
> хлеб
> молоко
> аспирин
> done

Вы должны найти файл, shoppinglist.txt, который содержит текст бананы, яблоки, хлеб, молоко, аспирин, каждый на отдельной строке, но не слово done.
Наиболее распространенное использование перенаправления будет посылать данные в файл. Например, если вы хотите сохранить информацию о файле, вы можете сделать одно из следующих действий:

ls -l> directory_listing.txt Ls -l >> directory_listing.txt

В последнем случае, если файл уже существует, команда добавит к файлу, а не перезапишет его.
Последняя форма перенаправления называется труба. Труба перенаправляет вывод одной программы, чтобы выполнить вход в другую программу. В качестве простого примера, можно использовать ls для просмотра содержимого текущего каталога. Тем не менее, есть слишком много файлов, поэтому многие из них уехало за пределы экрана слишком быстро для вас видеть . Вы хотите заставить ls сделать паузу между экранами. Вы можете сделать это с помощью команд more или less. Таким образом, вы используете канал, как в

ls | more
или ls | less.

Команда сначала выполняет ls, но вместо того, чтобы отображать выходные данные, он перенаправляет вывод в следующей программе, more (или less).
Кроме того, можно решить эту проблему гораздо более окольным способом, введя

ls> directory_listing.txt

а затем ввод

less directory_listing.txt

Труба избавит вас от необходимости сделать две отдельные команды и избавляет вас от создания файла, который вам вы действительно не нужен .

В Linux, труба может использоваться так часто, как требуется. Это потенциально может привести к длительным и очень сложным инструкциям. Рассмотрим следующую операцию:

ls -l | -fr sort | less

В этом случае вы получите длинный список текущего каталога, сортировки списка элементов в обратном алфавитном порядке и без учета регистра (он не будет отличаться между верхними и нижними буквами), а затем отображение результата, используя less. Мы увидим, другие виды использования трубы в двух следующих главах.
Мы заканчиваем этот раздел с учетом особенностей справки, которые доступны в Bash. На самом деле, эти функции можно найти в любой оболочке Linux. Помощь, описанная здесь, относится к тому, как использовать различные команды Linux. Наиболее распространенной формой помощи является страницы руководства (man) для обучения. Для просмотра страницы вручную, используйте команду человека, который берет на команде форму человека, как в человеке рода или человеку найти . Есть справочные страницы для большинства, но не для каждой команды Linux. Страницы руководства содержат ряд различных типов информации, которые помогают объяснить, как использовать команду. Они включают:

Краткое описание его использования (синтаксис)
Описание команд
Варианты (параметры), доступные для команд
Авторы программы
Файлы, связанные с командой и другие связанные команды, или подобные этому

Давайте посмотрим на некоторые примеры. Во-первых, мы смотрим на

инструкции rm. Когда мы вызываем man rm, мы сначала видим экран, показанный на рисунке 9.3.
Первый экран страницы rm man перечисляет имя, инструкцию по краткому обзору его использования, описание и первые два варианта . В нижней части окна показывает ":" и курсор после него. Для того, чтобы управлять движением на странице человека, мы будем использовать пробел, чтобы двигаться вперед, экран на экране. Но, как и меньшинство команды, мы также можем двигаться назад и вперед по одной строке за один раз с помощью клавиш со стрелками вверх и вниз. Буква "Q" завершает работу (завершает работу) справочную страницу. Варианты (только часть, которой показана на рисунке 9.3) и далее на следующих двух экранах. Рисунок 9.4 показывает, что остаток выглядит.
При рассмотрении справочной страницы rm, мы можем видеть, что опции -f, -i, -r или -R, и -v. Мы также можем использовать --interactive вместо -i, или --force вместо -f. Есть также варианты для --help и --version, и так далее.
Команда rm проста. Более сложная страница man с командой find. На самом деле, страница man find длиной 50 экранов. Одним из заметных различий в страницах руководства двух "команд в том, что находка включает в себя список примеров, чтобы проиллюстрировать, как использовать команду. Хотя примеры не являются общими в man страницах, вы поймете, что команда достаточно сложная.
Еще одна вариация, которую можно найти на страницах руководства

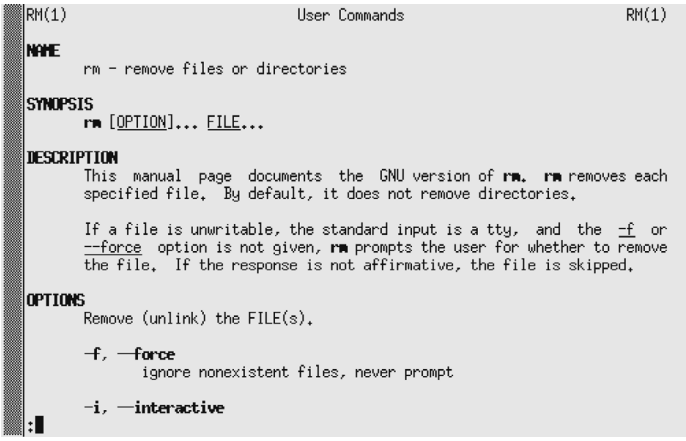


Рис.9.3
Первый экран страницы rm man.

с помощью справочной страницы для mount. В то время как rm и find имеют только одно использование по синопсису, mount имеет четыре. Они заключаются в следующем:

mount [-lhV]
mount -a [-fFnrsvw] [-t vfstype] [-o optlist]
mount [-fnrsvw] [-o optlist[, ...]] device | dir
mount [-fnrsvw] [-t vfstype] [-o опции] device dir

Эти четыре строки указывают, что mount может быть использован с четырьмя различными типами параметров. Во-первых, mount может следовать одному или более l, h, или V в качестве опции. Во-вторых, mount -a может сопровождаться списком опций, типа виртуальной файловой системы, а также любыми дополнительными опциями для этой команды. В-третьих, mount может сопровождаться списком опций, дополнительных опций, а также устройств или каталогов (символ | означает, что, либо устройство, или каталог должны быть указанными, представленными). Наконец, mount может сопровождаться списком опций, типом виртуальной файловой системы, списком опций, устройства, и каталога . Многие инструкции имеют несколько записей под синопсис. Другой, но более простой, пример можно найти с командой useradd.

Вторая форма помощи доступна команде help. Синтаксис команды справки является help имя_ команды . Тем не менее, очень немногие

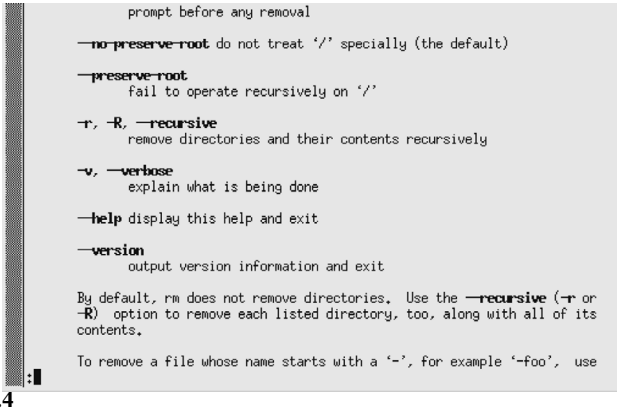


Рисунок 9.4
Второй экран страницы man rm

из команд Linux имеют справочные страницы, так что лучше всего использовать человека.
Окончательный вид помощи не обязательно поможет в том, как использовать команду. Вместо этого, он может помочь вам найти команду. Одна из проблем, с Linux вспоминать все-команды по имени. Если бы вы знали имя команды, то страница man могла бы помочь вам узнать, как использовать ее. Но что, если вы знаете, что вы хотите достичь, но не знаете название команды? Тогда появляется apropos. Синтаксис команды является apropos string, где string является строкой, что вы ищете. В команде apropos выходит список всех команд, которые имеют соответствующую строку, найденную в описании команды.
Например, есть команда, которая сообщает о текущем использовании виртуальной памяти.
Вы не можете вспомнить название команды. Вы вводите

apropos "virtual memory"

команда apropos выводит следующее:

mremap (2) - повторно отобразить виртуальный адрес памяти
vfork (3P) - создать новый процесс; обмениваться виртуальной памяти
vmstat (8) - Отчет статистики виртуальной памяти

Бинго, последний пункт является то, что мы искали! Хотя кавычки вокруг виртуальной памяти не нужны, без них, кстати, будет возвращено что-либо, что соответствует либо виртуальной или памяти. Это дало бы нам для исследования список длиннее. Тем не менее, с кавычками, кстати, будут возвращены только точные совпадения. Если мы хотим найти команды, чьи описания были оба слова виртуальной и память, но не обязательно в таком порядк е, мы не хотели бы использовать кавычки.
В качестве другого примера, вы хотите найти команду, чтобы удалить файл. Вы вводите apropos delete file, который выводит длинный список команд. Так что вы пробуете apropos "delete file". На этот раз, apropos выводит:

Delete file: ничего подходящего

указывая, что apropos не смогла найти соответствие "delete file". Использование apropos может быть сложно, но это является ценным

ресурсом.
Теперь, когда мы видели различные функции, доступные в оболочке Bash, мы приходим к выводу что этот раздел с краткого обзора, так как работает переводчик. Вы ввели команду в командной строке. Что происходит сейчас? Предпринимаются следующие шаги:

• интерпретатор читает входной файл.
• Переводчик разбивает ввод в слова и операторы
- Операторы - символы, такие как <,>, |, ~, *

Теперь, когда мы видели различные функции, доступные в оболочке Bash, мы кратко излагаем этот раздел в процедуре работы переводчика. Вы ввели команду в командной строке. Что происходит далее? Предпринимаются следующие шаги:
• Интерпретатор читает входной файл.
• Интерпретатор разбивает входной файл на слова и операторы
– операторы – это такие символы, как <,>, |, ~, *
• Если команда имеет какие-либо кавычки, эта обработка кавычек.
• Запуск оператора для определения псевдонимов, и если их находят, замена или расширение псевдонима.
• Разбивка различных слов и операторов на отдельные команды (например, если есть операторы перенаправления, то команды отделяются от этих операторов).
• Разложение интерпретатора, если требуется что-либо из
– Раскрытие скобок
– Раскрытие тильды
– Присваивание различных значений
• Если любая из команд появляется в виде `` или $ (), выполнение команды.
• Выполнение любых требуемых арифметических команд.
• Выполнение любых переназначений (включая конвееризацию).
• Выполнение расширения имени файла, то есть подбирание символов подстановки.
• Команда выполнена и, после завершения, код завершения команды (при необходимости) отображается в окне терминала (если не требуется исходное перенаправление).
Звучит так, будто нужно сделать очень много операций, но все происходит очень быстро. Теперь вы понимаете, почему использование командной строки Linux может дать большую гибкость и мощность,

нежели использование графического интерфейса.

Персонализация оболочки Bash

Как указано в разделах Особенности редактирования оболочки Bash и Изучения интерпретатора Bash, можно вводить команды псевдонимов и определять переменные из командной строки. К сожалению, любые такие псевдонимы или переменные будут известны только в пределах этой конкретной сессии оболочки. Что делать, если нужно иметь заданный псевдоним или переменную для всех сессий оболочки? Есть целый ряд файлов, доступных пользователю, которые загружаются при каждом запуске новой сессии оболочки. Если вы поместите свой псевдоним или определения переменных в один из этих файлов, то псевдоним или переменная будут известны для каждой сессии оболочки. Так, мы рассмотрим использование этих файлов. Прежде чем мы к ним перейдем, давайте рассмотрим, что происходит в начале сессии оболочки.
При запуске, оболочка проходит через процесс инициализации. Интерпретатор Bash выполняет несколько скриптов оболочки. К ним относятся в таком порядке /etc/profile, /etc/bashrc, .bash_profile, .bash_login, .profile, and .bashrc. Первые два скрипта устанавливаются системным администратором. По умолчанию, ими обеспечиваются все пользователи, например, путем создания псевдонимов, которые должны иметь все пользователи и путевые имена, с которых должны начинать все пользователи. Скрипт учетной записи выполняется каждый раз, когда запускается любая из оболочек, в то время как скрипт .bashrc выполняется только тогда, когда запускается оболочка Bash. Последние четыре скрипта определяются пользователем в своем домашнем каталоге. Скрипт .bash_profile обычно используется для подавления чего-либо установленного в системных скриптах, а также для персонализации элементов (псевдонимы, переменные) пользователя. Файл .bash_login выполняется только в том случае, если пользователь входит в систему (дистанционно) противоположно открытию новой оболочки. Скрипт .profile выполняется каждый раз, когда запускается любой тип оболочки , а не только Bash. Скрипт .bashrc выполняется, когда новая оболочка Bash запускается в новом окне. Для простоты, вероятно, лучше просто модифицировать скрипт .bash_profile, поскольку он используется при каждом запуске сессии Bash.
Если вы проанализируете эти файлы, то найдете множество входных данных. Существуют заранее заданные псевдонимы и переменные,

но существуют и скрипты операций оболочки. Так как мы будем рассматривать скрипт оболочки в главе 14, сейчас мы на этом останавливаться не будем. Скрипт .bash_profile содержит две следующие операции:
source ~./bashrc
source ~./bash_login
Для выполнения скрипта используется команда информации. Знак ~ указывает на то, что скрипт находится в вашем домашнем каталоге. Знак ./ означает "выполнить этот сценарий". Таким образом, эти две инструкции будут выполнять два скрипта, .bashrc и .bash_login. Если вы найдете эти две команды в скрипте .bash_profile, то на самом деле вы сможете редактировать любой из этих трех файлов, и все они будут выполняться при запуске новой оболочки Bash.
Давайте посмотрим, что вы могли бы поместить в один из этих файлов, скажем, в файл .bashrc . Команды будут выглядеть так:
Экспорт PS1 = "…" // любой нужный вам запрос вписывается в кавычки PATH экспорт данных = "$PATH:~/:~/bin:~/scripts:/usr/sbin"
Слово "экспорт" означает, что определенная переменная должна быть известна для всей среды оболочки, а не только внутри этого скрипта. Переменная PS1, уже заданная , в настоящее время изменяется на нужный вам запрос. Например, вы можете использовать \d, чтобы обозначить данные или \w, чтобы указать текущий рабочий каталог. Последняя инструкция присваивания принимается на основе переменной PATH, которая определяется системным администратором в файле /etc/profile file, и присоединяет дополнительные каталоги к нему. Чтобы увидеть, что хранит ваша переменная PATH, вы можете ввести $PATH. Если вы хотите, вы можете определить дальнейшие переменные, но вы должны убедиться, что перед всеми переменными стоит слово "экспорт", благодаря чему они станут известными для среды оболочки.
Вы также можете определить некоторые заранее заданные псевдонимы. Вы можете изменить или добавить к этим псевдонимам, то, что вы хотите. Например, можно указать псевдоним исправления опечаток или вы можете указать псевдонимы, которые помогут вам уменьшить количество нажатий клавиш, что мы описывали ранее в этой главе.
Одно последнее замечание. Если вы когда-нибудь измените один или несколько из этих файлов , и позже обнаружите, что вы сделали ошибку или вам не понравится то, что вы сделали, вы можете скопировать новые версии каждого из этих файлов. Их можно найти в каталоге /etc/skel.

Текстовые редакторы

Как пользователь Linux, вы часто будете использовать текстовый редактор вместо текстового процессора. Это, конечно, больше касается системного администратора, которому часто приходится редактировать конфигурационные файлы или писать файлы скриптов оболочки, все из которых должны храниться в виде обычных текстовых файлов. Как мы говорили в предыдущем разделе, вы также можете отредактировать один из ваших скриптов инициализации Bash, такие как .bashrc. Для редактирования текстового файла, вам нужно использовать текстовый редактор. Поэтому стоит узнать о текстовых редакторах, доступных в Linux.
Проблема в использовании текстовых процессоров является то, что текстовые процессоры, по умолчанию, сохраняют документы в отформатированном виде. Например, текст , написанный в Word, будет сохранен в формате фала Word doc (.doc или.docx), а текстовой процессор OpenOffice сохраняет файл в формате .odt. Вы можете переопределить тип файла с помощью функции "Сохранить как" и изменить тип файла текста, но это может надоесть. Кроме того, запуск такого текстового процессора, как Word занимает гораздо больше времени, чем запуск простого текстового редактора. Когда вам нужно быстро отредактировать небольшой текст, вы лучше используете текстовый редактор вместо полноценного текстового процессора.

В Linux есть три общих текстовых редактора. Во-первых, есть редактор графического интерфейса, который часто называют просто Текстовый редактор или Gedit. Также есть две программы редактирования текста: vi и Emacs. Редактор vi является самым ранним из двух редакторов и поставляется со всеми установками Linux. Emacs может и не быть частью вашей Linux, хотя он достаточно прост в установке. Хотя Gedit и имеет GUI-интерфейс, так что вы можете перемещать курсор с помощью мыши и использовать мышь для выделения текста и выбора операций в меню, он вам может не понравиться или быть недоступным. Например, если вы вошли в компьютер Linux с помощью удаленного входа в систему при помощи протоколов Telnet или SSH, у вас не будет доступа к Gedit. Кроме того, vi и Emacs загружаются быстрее, и поэтому, если вы захотите сделать быстрое редактирование, то Gedit вам не подойдет. Итак, здесь мы рассмотрим редакторы vi и Emacs. Интересное попутное примечание: Linux (и Unix) пользователям, которые сначала изучают

vi, он нравится гораздо больше, а Emacs они ненавидят, в то время как те, кто сначала изучает Emacs, имеет противоположные чувства. Тем не менее, все пользователи Linux должны изучить работу обоих редакторов и использовать затем более предпочтительный для быстрого редактирования текста.

Редактор vi

Редактор vi является текстовым редактором Linux по умолчанию. Более новая версия vi называется vim. Следующее описание относится как к vi, так и к vim.

Поскольку это текстовый редактор, команды, которые вы найдете в меню с графическим интерфейсом, выполняются как нажатия клавиш. Редактор использует три различных режима - командный, режим вставки и режим замещения - нажатия клавиш отличаются в разных режимах, это мы обсудим позже. Важно помнить, что нажатия клавиш, которые используются в качестве команд, чувствительны к регистру. Например, "o" и "O" - клавиши вставки для разных положений.

Командный режим - это режим по умолчанию. Нажатия клавиш интерпретируются как команды. Множественное нажатие клавиш можно изменить на то число, сколько раз нужно выполнить команду. Например, команда dd удаляет текущую строку. 5dd удаляет пять линий подряд. Режим вставки - это режим, который наиболее похож на текстовый процессор; когда вы печатаете, символы вводятся в документ в точке курсора. С каждым нажатием клавиши, курсор переходит на следующую позицию. Единственная команда нажатием клавиши в этом режиме клавиша Esc, которая выходит из этого режима, возвращаясь в командный. Режим замещения аналогичен режиму вставки; тем не менее, как вы вводите клавиши, символы замещают уже существующие символы. Как и в режиме вставки, клавиша ESC возвращает вас в командный режим.

Для того, чтобы узнать, в каком режиме вы находитесь, посмотрите на нижнюю часть редактора, где будет написано "--ВСТАВКА--", "--ЗАМЕЩЕНИЕ--", или ничего (командный режим). Есть множество команд для входа в режим вставки или замещения. Они приведены в Таблице 9.2. Их так много из-за того, что каждая из них помещает курсор в разные места. Например, 'i' оставляет курсор вставки на текущей позиции, 'o' вставляет пустую строку в строку после курсора и вставляет курсор в начале пустой строки и 'O' вставляет пустую строку в строку, предшествующую курсор. Для перехода в режим вставки или замещения из командного режима, используйте одну из букв в Таблице 9.2, а для

возвращения в командный режим нажмите клавишу ESC.
Помимо изменения положения курсора при переключении в режим вставки, есть множество других способов. Они перечислены в Таблице 9.3. Обратите внимание, что нажатия клавиш перемещение курсора на одну позицию (вниз, вверх, влево, вправо) нужно делать через клавиши j, k, h, l, а не клавиши со стрелками, которые могут не отображаться vi/vim. Клавиши G или #G (где # - это номер) быстро переместят вас к любой точке в файле; тем не менее, для #G нужно знать номер строки. Команда поиска может быть гораздо полезней.

ТАБЛИЦА 9.2 Команды изменения параметров

Клавиша	Описание
i	Вход в режим вставки непосредственно слева от курсора
a	Вход в режим вставки непосредственно справа от курсора
o	Вставка пустой строки после текущей строки и переход в режим вставки
O	Вставка пустой строки перед текущей строки и переход в режим вставки
I	Вход в режим вставки в начале текущей строки
A	Вход в режим вставки в конце текущей строки
r	Заменить символ следующим введенным символом
R	Вход в режим замещение и продолжить замещать (переписывать) символы (до нажатия ESC)

ТАБЛИЦА 9.3 Команды перемещения курсора и поиска

j, k, h, l	Перемещение курсора вниз / вверх / влево / вправо на одну позицию (клавиши со стрелками могут не работать)
w, b	Перемещение курсора вперед / назад на одно слово
G	Переход к последней строке файла
1G	Переход к первой строке файла
*G	Переход к строке * (где * - число, например, 5G или 50G)
H, L	Переход к верхней / нижней строке экрана
0, $	Переход к первому / последнему символу строки
/текст <enter>	Поиск вперед для текста (текст может включать в себя обычные выражения)
/<enter>	Поиск для следующего появления последнего поиска

?текст <enter>	Поиск назад для текста
? <enter>	Поиск для предыдущего появления последнего поиска
<pg вверх> или cntrl+f, <pg вниз> или ctrl+b	Переместить файл вверх или вниз в замене окна на экране

Вы можете отметить текст, а затем перейти к нему. Это полезно для редактирования больших документов. Например, если файл содержит книгу, вы можете отметить начало каждой главы. В vi есть 36 доступных знаков. Они обозначаются использованием либо букв от A до Z, либо цифр от 0 до 9 (26 букв и 10 цифр дают 36 знаков). Вставить знак можно набрав mchar в командном режиме, где символ покажется либо буквой от A до Z , либо цифрой от 0 до 9. Переместиться к метке можно с помощью ‘char’. Вы также можете перейти к предыдущей метке, введя ‘char’. Например, если вы поместите метку 'Z' в начале Главы 2, а вы находитесь в середине Главы 5, нажатие ’z‘ переместит вас к началу Главы 2 и ‘z’ возвращает вас в середину Главы 5, где вы находились перед вводом ’z’.

Если курсор находится на разделителе, который также используется в C/ C++ или Java программировании, такие как {}, (,), [или], то нажатие % переместит вас к другому разделителю. Например, если вы находитесь на {, что запускает функцию, нажав % вы переместитесь к }, которая закрывает эту функцию. Это полезно при редактировании программы.

Таблица 9.4 отображает различные опции повтора и отмены. Многие команды можно выполнить несколько раз, если перед нажатием клавиши зажать цифру. 5J, например, будет повторять ‘j’ 5 раз, то есть, он будет перемещать курсор на пять строк вниз. Исключение: когда число предшествует букву G, в этом случае курсор переместится на начало данной строки; например, 5G перемещает курсор в начало строки 5.

Очевидно, что в любом текстовом редакторе / текстовом процессоре должна быть функция "вырезать и вставлять" или "копировать и вставлять". Как с графическим интерфейсом на основе текстового процессора, идея заключается в том, чтобы выбрать последовательность символов, а затем вырезать или скопировать их в буфер. Затем их можно извлечь из буфера. В текстовых процессорах, выражение для вставки часто называют рывком (вырвать текст из буфера обратно). Таблица 9.5 содержит команды для вырезания, копирования и вставки.

При запуске vi, вы можете либо указать файл, либо открыть новый буфер. После того, как вы отредактировали сессию vi, вам понадобятся

команды файлов. Все команды файлов начинаются с ':', как указано в Таблице 9.6. Обратите внимание на то, что некоторые из них требуют имя файла, которое нужно ввести после команды, как в названии :w, для того, чтобы сохранить содержимое буфера в имени файла с именем. Между командой и именем должен быть пробел.
Таблица 9.7 содержит несколько различных оставшихся команд vi.
Конечно, единственный способ действительно ознакомиться с vi - это практика. Учитывая столько причудливых комбинаций клавиш, необходимых для управления vi, вы можете задуматься - нужно вам им вообще пользоваться или нет? Тем не менее, как системный администратор или писатель скриптов оболочки, вам, вероятнее всего, придется часто пользоваться vi (или Emacs). Как уже говорилось ранее, те пользователи Linux, которые сперва изучают vi, как правило, сначала пользуются именно им. Вы увидите в редакторе Emacs, что многие команды, возможно, немного легче запомнить, поскольку многие команды начинаются с первой буквы названия команды.

ТАБЛИЦА 9.4 Команды повтора и отмены

u	Отменить предыдущую команду (можно использовать несколько раз, чтобы отменить несколько команд)
.	Повторить последнюю команду редактирования
n.	Повторить последнюю команду редактирования n раз, где n означает число, например, 5.
nk	Повторить команду k n раз, где k нажатие клавиши, а n - число, например, 5j или 5dd

ТАБЛИЦА 9.5 Команды удаления, копирования, ставки и изменения

x	Удалить следующий символ
nx	Удалите следующие n символы , где n – число
dw	Удалить следующее слово
ndw	Удалить следующее n слово, где n – число
dd	Удалить эту строку
ndd	Удалить следующие n строки , начиная с текущей строки, где n - число (также dnd)
D	Удалить от курсора до конца этой строки
yy	Копировать текущую строку в буфер
nyy	Скопировать следующие n строки , начиная с текущей строки, где n - число (также yny)

p	Вставить все строки, хранящиеся в буфере, ниже текущей строки
J	Присоединить следующую строку к концу текущей строки (удалить конец строки)
cw	Изменить текущее слово, заменив текст при вводе, до нажатия клавиши ESC
C	Изменить все слова в текущей строке до нажатия клавиши ESC

ТАБЛИЦА 9.6 Команды файлов

:w <enter>	Сохранить файл
:w *имя* <enter>	Сохранить файл как
:w! имя <enter>	Переписать имя файла с содержимым текущей сессии
:n,m *имя* <enter>	Сохранить n строки через m текущей сессии к имени файла
:q <enter>	Завершить vi
:q! <enter>	Завершить vi без сохранения
:wq <enter>или :x <enter>	Сохранить файл и завершить vi
:r *имя* <enter>	Вставить имя файла в текущую позицию текущего файла (Открыть файл, если сессия vi пуста)

ТАБЛИЦА 9.7 Различные команды

:h	Открыть справку vi
Ctrl+T	Вернуться в файл из справки
:!*команда* <enter>	Выполнить команду (это полезно, если вы знаете название команды, но не помните комбинацию клавиш (s))
:. =	Отображение номера текущей строки в нижней части экрана
: =	Отображение общего номера строк в нижней части экрана
^g	Отображение номера текущей строки и общего числа строк в нижней части экрана

Редактор Emacs

Emacs (обозначает Editor MACroS - редактор макросов) представляет собой текстовый редактор, как и vi. Тем не менее, в то время как у vi есть три режима (вставка, замещение и команда), Emacs имеет только один режим (вставка). Команды выполняются с использованием либо CTRL, или ESC в сочетании с другими клавишами. Например, для перемещения

одной строки вперед, нужно ввести CTRL + n ('n' для следующего). Многие из команд используют букву, на которую называется команда и, таким образом, Emacs можно легче изучить, чем vi. Программисты чаще выбирают Emacs, так как правильно настроенная среда редактора может выполнять программные команды. Например, язык программирования Липс часто устанавливается в Emacs и поэтому программист может отредактировать Липс-код, а затем выполнить все это внутри Emacs.

(a) (б)

При запуске Emacs сначала появляется окно-заставка (Рисунок 9.5A). Как только вы начнете редактирование (введя символы или переместив курсор), изображение в Emacs заменяет пустая страница (за исключением трех строк текста, смотрите Рисунок 9.5). Вы заметите, что на Рисунке 9.5 есть GUI-команды. Если вы запускаете Emacs внутри графического интерфейса операционной системы, вы можете контролировать Emacs через мышь (также через клавиши). Тем не менее, если открыть Emacs сессии на основе текста (например, если вы вошли в Linux через SSH-доступ), то вы можете использовать только клавиши.

В Emacs, вы редактируете буфер. Буфер представляет собой область памяти, которая содержит текст - это именно та сессия, что вы в настоящее время редактируете. В Emacs, много буферов могут быть открытыми одновременно. Это позволяет редактировать несколько документов в одно время и вырезать / копировать и вставлять между ними. Вообще есть два буфера, видимые в любой момент времени: основной буфер (большое окно в обеих частях Рисунка 9.5) и мини-буфер (в нижней части окна обеих частей Рисунка 9.5). Мини-буфер - это место, где вы будете вводить команды, имена файлов и ответы на вопросы (такие, как "Сохранить файл?"). Вы действительно можете открыть два буфера одновременно (в таком случае большое окно нужно разделить на две части) и каждый отдельный буфер можно разбить еще на два буфера. Мы обсудим это позже.

Emacs заворачивает символы на следующую строку, когда вы достигаете правой стороны буфера, как и любой текстовый процессор; тем не менее, вы можете увидеть, что строка расширяется, поскольку ее конец обозначается символом \. Если бы вы находились на этой строке и использовали бы команду "перейти к концу строки", курсор переместился бы к концу расширенной строки.

Например, если у вас есть:

```
Desperate nerds in high offices all over the world have been known to enact the m\
ost disgusting pieces of legislation in order to win votes (or, in places where
```

they d\
on’t get to vote, to control unwanted forms of mass behavior).
При вводе каких-либо комбинаций клавиш, они будут добавлены в текущий буфер Emacs на положение курсора, только если вы не вводите команду. Для ввода команды, вам нужно либо удерживать клавишу CTRL и вводить один или несколько символов (это будет обозначаться как c+клавиша), либо нажать и отпустить клавишу ESC и следом вводить один или несколько символов (это будет обозначаться как m+клавиша, m означает мета, термин, который некоторые пользователи используют для клавиши ESC). Начнем с рассмотрения команд перемещения курсора, аналогично тому, что мы видели в vi. В Emacs, большинство из этих команд обозначаются первой буквой названия, так что, например, c + б перемещает курсор назад (backward) на один пробел. Вы можете узнать некоторые из этих команд из дискуссии по редактированию командной строки в Особенностях редактирования оболочки Bash. Таблица 9.8 содержит команды перемещения курсора.
Обратите внимание на то, что также вы можете отмечать позицию, как и в vi. Но здесь нельзя ставить отметку (в vi можно было поставить до 36 отметок). Здесь, Emacs может записать до 16 знаков. Вы можете либо вернуться к самой последней отмеченной позиции (c + x c + x) или вы можете пересмотреть все отмеченные позиции, зажав c+u c + пробел, один раз на отметку. Если вы начнете отмечать позиции после 16-ой, самые ранние предыдущие отметки будут забыты.

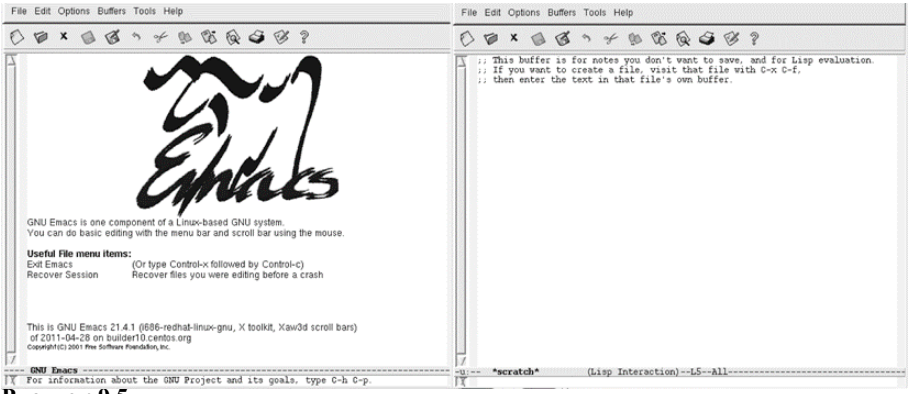


Рисунок 9.5
Emacs во время запуска (а) и во время редактирования (б)
и курсор находится в первой строке, то переход к концу строки на самом деле переместит курсор к концу behavior)., а не к “m\”.

Помимо использования c+u для перемещения сквозь метки, эта команда также используется для обозначения "повторить следующее движение 4 раза". Так, например, c+u c + n продвинет вас вперед четыре раза. Интересно, что c+u c+u повторяет следующую команду 16 раз. Вы можете также использовать c+u n, где n - число, которое означает "повторить следующую команду n раз". Можно также указать многозначное количество (например, 15, 35, 200) с использованием m+digit m+digit. Например, m+1 m+0 c+f перемещает курсор вперед на 10 пробелов, а m+3 m+2 c+n переместить курсор на 32 строки вперед. Когда вы достигнете нижней части файла, курсор дальше не продвинется и не вернется к началу.

ТАБЛИЦА 9.8 Команды движения курсора

c+n	Переместить курсор на одну строку вниз (следующую)
c+p	Переместить курсор на одну строку вверх (предыдущую)
c+f	Переместить курсор на одну позицию вперед
c+b	Переместить курсор на одну позицию назад
c+a	Переместить курсор в начало строки
c+e	Переместить курсор в конец строки
c+v	Переместить курсор вниз на один экран
m+v	Переместить курсор вверх на один экран
m+f	Переместить курсор на одно слово вперед (к следующему пустому)
m+b	Переместить курсор назад к началу слова (или предыдущему слову)
m+a	Переместить курсор в начало предыдущего предложения
m+e	Переместить курсор в начало следующего предложения
m+<	Переместить курсор в начало документа
m+>	Переместить курсор в конец документа
m+g *n*	Переход к номеру строки n, где n - число
c+пробел	Отметить текущую позицию
c+x c+x	Переход к отмеченной позиции (также перемещает отметку в положение, где вы только что были)
c+s	Открывает мини-буфер для поиска по ключевому слову, поиск вперед (каждый c + s) будет проводить поиск до ввода новой команды
c+r	Поиск назад

При вводе текста символы вставляются в текущую позицию курсора, так

что Emacs всегда находится в режиме вставки. Команды редактирования для удаления / вырезания и вставки представлены в Таблице 9.9. Обратите внимание на то, что команда отмены отменяет последнюю операцию редактирования, но не операцию перемещения. Так, например, если вы удалите символ, отмена восстановит этот символ. Если удалить символ, переместить курсор и выбрать отмену, восстановится удаленный символ, а курсор не переместится к предыдущей позиции.

Обратите внимание, что текст добавляется в буфер до тех пор, пока вы не введете другую команду. Например, если вы удалили 5 строк подряд, то все 5 строк скопируются в буфер. Если вам нужно удалить строку, введите c+n для перехода на следующую строку, затем удалите эту строку и только последняя строка сохранится в буфере, поскольку между этими двумя удалениями была выполнена еще одна команда. Последующие удаления сохранятся в буфере.

Например, рассмотрим следующую строку:

The brown cow jumped over the silver spoon.

Предположим, что курсор находится в настоящее время на ‘v’ в слове “over”. Если вы вводите m+d, удаляется “ver”. Если вы повторите m+d, удалится “the”. Если вы сделаете c + y, “ver the” восстановится. Если курсор находится в том же месте, и вы дважды введете m+<пробел>, удалится “jumped ov”, но не “er”. Если бы вы сейчас сделали c+f, а затем m+d, это удалило бы “r” в “er”. Если затем вы введете c+y, восстановится только “r”, так как “jumped ov” теперь не в буфере.

Операции фала приведены в Таблице 9.10. Мини-буфер используется для запроса имени у пользователя при "Сохранить как", и открывается, когда вы хотите выйти из Emacs без сохранения документа. Обратите внимание, что, если вы никогда не сохраняли содержимое буфера, Emacs завершится без сохранения. Если вы сохранили содержимое буфера в файл, но после этого внесли изменения и не сохранили их, тогда у вас спросят, хотите вы сохранить файл под существующим именем, под новым именем или выйти без сохранения.

ТАБЛИЦА 9.9 Команды редактирования

c+x u	Отмена (можно также использовать c+_ и c+/)
c+x u	Удалить следующий символ (также клавиша удаления)
m+d	Удаление следующего слова (от курсора)
c+k	Удалить всю строку (от курсора) (также m+k)
<пробел>	Удалить предыдущий символ

m+<пробел>	Удалить в обратном направлении от курсора до начала слова
m+w	Сохранить всю область от последней отметки до курсора (копировать)
c+w	Удалить всю область от последней отметки до курсора (вырезать)
c+y	Вставить из буфера копирования (все вышеперечисленные вставки символов в буфер кроме отмены и c+d)
m+y	То же самое, что и c + y, за исключением того, что он заменяет текст с тем, что находится в буфере, а вставляет текст в позиции курсора
c+t	Перестановка текущего и предыдущего символов
m+u, m+l	Верхний / нижний регистр всех букв от курсора до конца слова
m+c	Делает заглавной текущую букву
m+%	Поиск и замена - вам будут предложены строки для поиска и замены, а затем для каждого совпадения, вам будет предложено либо заменить (y), пропустить (n), выйти (q), заменить все оставшиеся экземпляры (!) или заменить один раз, и выйти (.).

ТАБЛИЦА 9.10 Команды файлов

c+x c+s	Сохраните файл под текущим именем
c+x c+w	Сохранить как (вам будет предложено ввести новое имя)
c+x c+f	Открыть новый файл в новом буфере
c+x c+c	Выход из Emacs (если содержимое не было сохранено , но имя файла существует, то вам будет предложено сохранить содержимое перед выходом)

Команда m+x выводит подсказку в мини-буфере. Здесь вы можете ввести команду и нажать клавишу ввода. Существует очень много команд. Если вы не уверены в названии команды, вы можете использовать версию Emacs заполнения нажатием клавиши TAB. Начните набирать название команды и нажмите пробел. В нижней половине окна буфера появятся все соответствующие команды. В Таблице 9.11 наведены команды для открытия, закрытия и переключения между буферами. Обратите внимание, что пробел для выхода может быть также использован для завершения ввода имени файла. Например, если вы хотите открыть файл, введите c+x c+f. Затем вам предложат ввести имя файла в мини-

буфере. Вы вводите путь к файлу path/home/foxr/cit130/ , но не помните его название. При нажатии на пробел сгенерируется список всех подходящих элементов, в данном случае, все файлы в этом каталоге. Вот лишь некоторые команды, которые можно ввести в мини-буфере:

- добавление-в-файл
- режим-автоматического-сохранения
- калькулятор
- календарь
- сделать-заглавной-область, сделать-заглавным-слово
- проверить-круглые-скобки
- копировать-файл
- посчитать-строки-страница, посчитать-строки-область
- перейти-строка
- вставить -файл
- сохранить-буфер
- оболочка (открывает оболочку Linux)
- отмена

ТАБЛИЦА 9.11 Команды буфера

c+x k	Закрыть текущий буфер (если есть несохраненные данные, то будет предложено сначала их сохранить)
c+x b	Переключение в самый последний буфер, кроме текущего буфера
c+x 2	Открыть второе окно (в рамках текущего окна), которое занимает половину окна для того, чтобы увидеть второй буфер (примечание: делать это внутри половины окна дает два окна размером в четверть и так далее)
c+x 1	Закрыть второе окно к одному окну
c+x o	Переключение из одного буфера в другой, в том числе в мини-буфер

Есть целый ряд программ, которые могут запускаться внутри Emacs. К ним относятся доктор (программа искусственного интеллекта), ханой (ханнойская башня), фазы-луны (показывает предстоящие фазы луны) и тетрис. Программу можно запустить, используя m+x название, как в m+x доктор <enter> для запуска программы доктор. Внутри Emacs можно запускать команды Linux.

Есть несколько режимов встроенной проверки орфографии при помощи программ aspell или ispell. Flyspell (использует aspell) выдвигает на первый

план все слова в буфере с ошибками. Вы также можете использовать режим программирования flyspell, который только подчеркивает слова с ошибками в комментариях (например, находит все орфографические ошибки в комментариях программы Java). Ispell используется для оставшихся режимов. Ispell может проверить выделенное слово или может завершить текущее слово, может проверить текущий буфер или текущую область. В ispell, окно разделяется на два буфера со списками слов для замены ошибок, которые возникают в нижнем буфере. Также можно добавить слова в словарь ispell. Для запуска или проверки орфографии нужно ввести M+x <command>, где <command> это одно из:

• режим-flyspell
• режим-flyspell-прог
• ispell
• буфер-ispell
• область-ispell
• сообщение-ispell
• изменение-словаря-ispell (позволяет изменять ваш словарь ispell).

Вы можете также использовать M + $ для проверки орфографии текущего слова и M+x<tab> <esc> <tab> для завершения проверки текущего слова. При использовании ispell, если слово выбрано неправильным, вы можете сделать любое из этих действий:

• <пробел> - пропустить слово (оно по-прежнему считается орфографически неверным, просто игнорируется в данный момент)
• r слово <return> - выделенное слово будет заменено на слово, которое вы ввели
• цифра - заменить слово словом под определенным номером, которое выбрано с предоставленного списка
• a - оставить слово как есть
• i - вставить слово в словарь
•? - показать список всех опций, доступных в Ispell.

Одна последняя особенность Emacs - это возможность определения и повторного вызова макросов (это доступно и в vi, но мы это не обсуждали ранее). Макрос - это программа, наделенная определенной комбинацией клавиш, которую можно выполнять более одного раза. Вы определяете макрос, а затем используете его, когда он вам нужен. Для запуска макроса, введите c + x (с последующим нажатием клавиш макроса и снова c + x). Для выполнения введенного макроса введите c+x e.

Рассмотрим пример. Представьте себе, что у вас есть группа текста, и вы хотите поместить этот текст в HTML-таблицу. Текст в настоящее

время определяется как # Ch. # p. #, где # - число. Вы хотите поместить этот текст в формат HTML-таблицы, где каждая строка HTML таблицы соответствует строке в текстовой таблице. Первый # будет помещен в одну колонку, Ch. # в другую, и p. # в третью. HTML- таблица начинается с <table> и заканчивается </ table> и каждая строка состоит из: </td></ tr>.столбец 1 </td><td> столбец 2 </td><td>столбец 3 </td></tr>. Так что нам нужно добавить к каждой строке </td></tr>.перед первым #, </td></ tr>перед Ch. #, </td><td> перед p. #, A </ TD> </ TR> до конца строки. Для того, чтобы выполнить поставленную выше задачу, необходимо сделать следующее:

• Переместить курсор к первой строке таблицы, ввести <table> <enter > и ввести c + x (

• Ввести c + a <tr><td> - это переместит курсор в начало строки и добавит текст <tr><td> в начале

• Ввести m+f - это переместит курсор за первое слово (в данном случае, первое число)

• Ввести </td><td> - закончить первый столбец и начать второй

• Ввести c+s . c+s . c+b c+b - это переместит курсор непосредственно перед p.

• Ввести </td><td> - закончить второй столбец и начать третий

• Ввести c+e </td></tr> - перейти к концу ряда, закончить столбец и ряд

• Ввести c+n - для перехода к следующей строке

• Ввести c + x) - закончить макрос.

Если бы было 15 рядов для таблицы, теперь мы сделаем c+u 14 c+x e (выполним макрос 14 раз) и, наконец, введем </table>. Вы можете сохранять макросы и давать им имена, так что вы можете создать много макросов и использовать их, когда они нужны. Позже вы можете отредактировать макрос с названием добавить к нему что-то или внести изменения.

Наконец, Emacs может сбивать с толку. Если вы потерялись, вы можете вызвать файл справки в новом буфере. Чтобы вызвать справку, введите c+h c+h или, если вы хотите получить справку о конкретной команде, введите типа c+h команда. Если вы не уверены в правильности команды, введите первую букву или буквы команды. Как и в случае vi, лучший способ изучить Emacs - практика!

ГЛАВА 10.
РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Глава 9 концентрировалась на Linux. Глава 10 также фокусируется на теме, связанной с Linux , на регулярных выражениях. Регулярные выражения - это мощный инструмент для пользователей и администраторов Linux. С помощью регулярных выражений, пользователь может производить поиск через текстовые файлы не для конкретных строк, но для строк, которые соответствуют определенному интересующему шаблону. Linux предлагает утилиту grep ,которая выполняет такую задачу поиска по заданному регулярному выражению. В этой главе, регулярные выражения вводятся вместе с многочисленными примерами и рассмотрением утилиты grep. Из-за их сложной природы, при изучении регулярных выражений, читатель должен помнить, что мастерство в них приходит только с большим опытом работы, это может быть трудная глава для чтения и понимания. Рекомендуется, чтобы читатель попробовал многие из этих примеров. В этой главе также рассматривается использование символов-джокеров Bash.

Целями обучения данной главы являются:

• Описать регулярные выражения и почему они полезны.

• Проиллюстрировать использование каждого метасимвола регулярного выражения.

• Обеспечить многочисленные примеры регулярных выражений.

• Проверить утилиту grep.

• Описать использование символов-джокеров Bash и показать, как они отличаются от регулярных выражений.

• Объединить ls и grep через перенаправление.

Рассмотрим строку символов, содержащую только 1s с последующим 0s, например, 111000,100 и 10000. Для указания такого шаблона можно использовать регулярные выражения. Написанное регулярное выражение можно сравнить с группой строк и оставить те, которые соответствуют шаблону. Регулярное выражение - строка, которая сочетает в себе буквенные символы (например,0 или 1) и метасимвола, символы, которые представляют варианты. С помощью метасимволов, вы можете указать, что, например, данный символ или набор символов может

соответствовать "любому количеству" или "по крайней мере, один" раз, либо указать список символов, чтобы подходил "любой символ".
Регулярные выражения могут быть очень полезны для любого пользователя или системного администратора, если речь идет о поиске файлов или элементов, хранящихся в файлах. В этой главе мы рассмотрим, как определить регулярные выражения и как использовать их в Linux. Регулярное выражение считается настолько полезным, что Linux имеет встроенную утилиту, которая называется grep, которая является важным инструментом для пользователей Linux. Символы-джокеры, форма регулярных выражений, доступны в Linux, хотя они по-разному интерпретируются в зависимости от их использования в регулярных выражениях. Регулярные выражения были построены в некоторых языках программирования, которые широко используются в Linux, таких как Perl.
Рассмотрим два простых примера для того, чтобы объяснить , почему мы хотим изучить регулярные выражения. Во-первых, вы, как пользователь, имеете доступ к каталогу изображений. Среди изображений есть такие форматы, как jpg, gif, png и tiff. Вы хотите перечислить все изображения в формате TIFF. Тем не менее, вы не уверены, как другие пользователи их назвали - ..tiff, .tif, .TIFF, .TIF, .Tiff или .Tif рассширение. Вместо того, чтобы писать шесть различных Ls инструкций или даже одну Ls инструкцию для того, чтобы перечислить все возможные расширения, вы можете использовать регулярные выражения. Во-вторых, как системный администратор, вы должны искать каталог (say /etc) для всех файлов, которые содержат IP-адреса, как вы хотите изменить некоторые жёстко запрограммированные IP-адреса, но вы не помните , какие файлы нужно проверить. Регулярное выражение может быть построено, чтобы соответствовать строке вида #. #. #. #, где каждый # - это значение в диапазоне от 0 до 255. При создании такого, регулярное выражение и использовании grep, вы сможете увидеть все совпадения с помощью одной команды вместо того, чтобы исследовать десятки или сотни файлов.
В этой главе, мы сначала рассмотрим метасимволы для регулярных выражений. Мы рассмотрим десятки примеров регулярных выражений, и с чем они могут совпадать. Затем мы рассмотрим, как некоторые символы используются в качестве символов-джокеров интерпретатором Bash. Это может привести к путанице, поскольку * имеет различное значение, когда используется в качестве регулярного выражения в такой утилите как grep и как интерпретатор Bash использует его в такой

инструкции как ls. И, наконец, мы рассмотрим утилиту grep и как её использовать. Правильное применение регулярных выраженний может стать проблемой. Несмотря на то, что во многих случаях их значение может быть очевидным, часто они могут запутать пользователей, которые не знакомы с ними. Имейте терпение при их использовании и, в конце концов, вы даже можете наслаждаться ими.

МЕТАСИМВОЛЫ

Существует множество символов, которые используются для описания параметров в шаблоне. Они известны как метасимволы. Любое регулярное выражение будет содержать буквенные символы и метасимволы (хотя регулярное выражение не требует метасимволов). Метасимвол * означает , что "соответствует предыдущему символу 0 или более раз"; так, например, a* означает "ноль или более A" . Регулярное выражение 1010 соответствует только 1010, поскольку здесь нет метасимволов. Так как мы обычно хотим, чтобы регулярные выражения соответствовали более чем одной конкретной строке, мы почти всегда будем использовать метасимволы. Таблица 10.1 содержит список метасимволов. Далее в главе, мы рассмотрим каждый из них.

Мы начнем с самых основных символов: * и +. Чтобы использовать любой из них, сначала нужно указать символ для сравнения, затем поместить метасимвол * или + после символа, чтобы показать, что мы ожидаем увидеть этот символ 0 или более раз (*) или 1 или более раз (+).

ТАБЛИЦА 10.1 Метасимволы регулярных выражений

Метасимвол	Объяснение
*	Совпадает с предыдущим символом, если он появляется 0 или более раз
+	Совпадает с предыдущим символом, если он появляется 1 или более раз
?	Совпадает с предыдущим символом, если он появляется 0 или 1 раз
.	Совпадает с любым символом
^	Совпадает, если это выражение начинает строку
$	Совпадает, если это выражение заканчивает строку
[chars]	Совпадает, если выражение содержит любой из символов в []
[chari-charj]	Совпадает, если выражение содержит любые символы в диапазоне от $char_j$ до $char_j$ (например, A-Z, 0-9)

[[:class:]]	Альтернативная форма [], где: class: может быть одним из нескольких категорий, таких как альфа (алфавитный), цифра, специальное выражение (буквенное или цифровое), пунктуатор, пространство, верхнее, нижнее
[^chars]	Совпадает, если выражение не содержит какой-либо из символов в []
\	Следующий символ должен буквально интерпретироваться и использоваться для избегания смысла метасимвол, например, \ $ означает "соответствовать $"
{n}	Совпадает, если строка содержит n последовательных наличий предыдущего символа
{n,m}	Совпадает, если строка содержит последовательные наличия предшествующего символа между n и m
{n,}	Совпадает, если строка содержит, по крайней мере, n последовательных наличий предыдущего символа
{,m}	Совпадает, если строка содержит не более m последовательных наличий предшествующего символа
|	Совпадает с любой из этих строк ("ИЛИ")
(…)	Элементы ... рассматриваются как группа, которая соответствует всей последовательности

Например, 0 * 1 * соответствует любой строке из нуля или более 0s, за которой следует ноль или более 1s. Это регулярное выражение будет соответствовать этим строкам: 01, 000111111, 1, 00000, 0000000001 и пустая строка. Пустая строка - строка без символов. Это выражение соответствует пустой строке, потому что * может быть использован для 0 совпадений, так что 0 * 1 * соответствует строке нет 0s и нет 1s. Этот пример регулярного выражения не соответствует ни одному из следующих значений: 10,00001110, 0001112, 00a000 или алфавит. В первом и втором случаях 0 следует за 1. В следующих трех случаях, в строке есть символы, кроме 0 и 1.

Регулярное выражение 0 + 1 + указывает, что должно быть, по крайней мере, один 0 и один 1. Таким образом, это регулярное выражение не совпадает с пустой строкой; также оно не соответствует ни одной строке, которая не содержит одну из двух цифр (например, оно не соответствует 0000 или 1). Это выражение соответствовало бы 01, 00011111 и 000000001. Как и 0 * 1 *, оно не будет соответствовать строке, в которой есть другие символы, кроме 0 или 1 (например, 0001112), также не будет соответствовать и строке, в которой 0 следует за 1 (например, 00001110). Конечно, мы можем сочетать использование * и + в регулярном

выражении, как и в 0 * 1 + или 0 + 1 *. Мы также можем указать буквенные символы без * или +. Например, 01 * будет соответствовать 0, за которым следует ноль или более 1s - так, например, 0, 01, 01111, но не равно 1, 111, 1110 или 01a. Несмотря на то, * и + понять проще всего, их полезность ограничена, когда можно указать только последующий символ. Мы увидим, что * и + являются более полезными, когда мы можем объединить их с [], чтобы указать комбинацию повторяющихся символов.

? - это вариант, как и * или +, но он будет соответствовать предыдущему символу только 0 или 1 раз. Это позволяет определить ситуацию, когда символ можно или нельзя ожидать. Однако, это не означает, что совпадение может быть неоднократным (как для * или +). Вспомним .tiff / TIF пример. Мы могли бы указать регулярное выражение, которое соответствовало бы tiff или tif следующим образом: tiff?. В этом случае мы буквально ожидаем увидеть три первых символа, tif. Однако, последний символ, f, может появиться 0 или 1 раз. Хотя это регулярное выражение не удовлетворяет такие строки, как TIFF (то есть все заглавные буквы), с этого можно начать. Теперь, с ?, * и +, мы можем контролировать, как часто мы ожидаем увидеть символ, 0 или 1 раз, 0 или более раз, 1 или более раз.

В отличие от * и +, ? накладывает ограничение на количество раз, которое мы ожидаем увидеть символ. Поэтому, с ?, мы могли бы на самом деле перечислить все комбинации, которые могут совпасть. Например, 0? 1? будет соответствовать только четырем возможным строкам: 0, 1, 01 и пустая строка. В случае 0 * 1 *, существует бесконечное число строк, которые могли бы совпадать , так как "0 или более" не имеет верхнего предела.

Обратите внимание, что * и ? используются в командах Linux, как символы-джокеры ls. В Bash и Символы-джокеры, мы узнаем, что в таких командах, их значение отличается от значений, представленных здесь.

. (период) может быть использован для того, чтобы соответствовать любому отдельному символу. Например, b.t может соответствовать любой из этих строк bat, bet, bit, but, bot, bbt, b2t. Оно не будет соответствовать bt, boot, or b123t. Метасимволом . можно скомбинировать с *, + и ?. Например, b.*t будет соответствовать любой строке, которая начинается с b, за ней следует любое количество других символов (в том числе не символов) и заканчивается t. Таким образом, b.*t совпадает с bat, bet, bit, but, bot, bbt, b2t, bt, boot, b123t и так далее. Выражение b.+t такое же, за

исключением того, что между b и t должен быть хотя бы один символ, так что ему будут соответствовать все те же строки за исключением bt. Регулярное выражение b.?t будет соответствовать bt или всему, что соответствует b.t. Знак вопроса соответствует . (период). Следовательно, . применяется 0 или 1 раз; так что это дает нам регулярное выражение, которое соответствует bt или b.t. Оно не соответствует ни одной строке, которая содержит более одного символа между b и t.

Следующий метасимвол используется для задания набора или списка. Она начинается с [, содержит список, и заканчивается]. Например, [aeiou] или [123]. Идея заключается в том, что такой шаблон будет соответствовать любой строке, которая содержит какой-либо один из элементов в списке. Мы могли бы, например, указать b[aeiou]t, которые соответствовали бы bat, bet, bit, bot, и but. Или мы могли бы использовать [], чтобы указать на верхний или нижний регистр. Например, [Tt] будет соответствовать либо нижнему регистру t, или верхнему T.

Теперь у нас есть инструменты, необходимые для соответствия любой форме tif/tiff. Следующее регулярное выражение будет соответствовать любой форме tif или tiff с использованием любой комбинации строчных и прописных букв: [tT][iI][fF][fF]?. ? относится только к четвертому []. Таким образом, оно будет соответствовать любым строчкам с прописными или строчными буквам t, за которыми следуют прописные или строчные i, с последующими прописными или строчными f, и заканчивающимися на ноль или одну f в верхнем регистре.

Количество элементов в скобках, не обязательно должны быть полностью нумерованным списком. Вместо этого, может быть диапазон букв от а до г. Ряд представляется первым символом в диапазоне, дефисом (-) и последним символом в диапазоне. Допустимыми символами для диапазонов являются цифры, строчные и заглавные буквы. Например, [0-9] будет означать любую цифру, в то время как [а-г] будет означать любой символ нижнего регистра от а до г. То есть, [а-г] эквивалентно [абвг]. Вы можете комбинировать перечисленный список символов и диапазон, например, [б-вг-тв-хг] будет списком строчных согласных звуков.

В качестве альтернативы перечисленному списку или диапазону, можно также использовать двойные скобки и класс . Например, вместо [a-zA-Z], вы можете использовать [[:alpha:]], который представляет класс алфавитных символов. Есть 12 стандартных классов, доступных на регулярных выражениях Linux, как показано в Таблице 10.2. Нестандартный класс [[:слово:]], который состоит из всех буквенно-

цифровых символов плюс символ подчеркивания .
В списке, указанный с помощью [] или [[]], будет соответствовать любому одному символу, найденному в строке. Если вы хотите, чтобы он соответствовал какой-то комбинации символов в диапазоне, вы можете добавить *,+,., или ? после скобок. Например, [a-z]+ означает одну или несколько букв нижнего регистра.
Представьте себе, что вы хотите найти соответствие чьему-то имени. Мы не знаем, будет ли первая буква имени заглавной, но мы ожидаем, что все остальные буквы будут в нижнем регистре. Для соответствия первой букве мы будем использовать [A-Za-z]. То есть, мы ожидаем букву либо и верхнего, и нижнего регистра. Затем мы ожидаем некоторого количества строчных букв, что обозначим [a-z]+. Наше регулярное выражение [A-Za-z][a-z]+. Должны ли мы использовать * вместо + для нижнего регистра букв? Это зависит от того, ожидаем ли мы, что имя может состоять из одной буквы. Так как мы ожидаем имя, а не инициал, то мы думаем, что оно состоит из нескольких букв. Тем не менее, мы могли бы также использовать [A-Za-z][a-z]*, если мы думаем, что именем могло бы быть, например, J.
С чем может совпадать [A-Za-z0-9]*? Это выражение будет соответствовать нулю или большему количеству букв или цифр. Оно также включает пустую строку (как * включает в себя нулевые вхождения), любую одну букву (верхний или нижний регистр) или цифру, или любое сочетание букв и цифр. Таким образом, будут соответствовать:
abc ABC aBc a12 1B2 12c 123456789 aaaaaa 1a2b3C4D5e
Так что не будет совпадать? Любая строка, которая содержала бы другие символы, кроме букв и цифр. Например, a_b_c, 123!, a b c (то есть буквы и пустые пространства), а также a1#2B%3c*fg45.

ТАБЛИЦА 10.2 Метасимволы регулярных выражений

Класс	Значение
[[:alnum:]]	Буквенно-цифровой символ - буквенный символ (буква) или цифра
[[:alpha:]]	Алфавитный - буква (верхний или нижний регистр)
[[:blank:]]	Пробел или tab
[[:cntrl:]]	Любой управляющий символ
[[:digit:]]	Цифра
[[:graph:]]	Любой видимый символ
[[:lower:]]	Строчная буква

[[:print:]]	Любой видимый символ плюс пробел
[[:punct:]]	Любой знак препинания
[[:space:]]	Любой пробельный символ (tab, клавиша return, пробел, backspace)
[[:upper:]]	Прописная буква
[[:xdigit:]]	Шестнадцатеричная цифра

Обратите внимание, с помощью [] мы можем контролировать, какие символы могут совпадать, но не их появление. Если мы, например, требуем, чтобы ''b' находилась за 'a', мы записываем их в последовательности, используя два набора скобок, например, [aA][bB], для указания верхнего или нижнего регистра 'a' , за которым следует прописная или строчная буква ‘b’. Мы могли бы также позволить любое их количество с помощью [aA]+[bB]+ (или использовать * вместо +). Это может стать сложнее, если мы хотим, чтобы комбинация букв следовала за комбинацией. Подумайте, мы хотим создать регулярное выражение, которое соответствовало бы любой строке символов, где существует некоторая согласная (e), за которой следует некоторая гласная (e) с последующим согласным (и). Мы могли бы использовать следующее регулярное выражение:

```
[b-df-hj-np-tv-z]+[aeiou]+[b-df-hj-np-tv-z]+
```

Можно ли применить больший контроль к "1 или более"? * и + подходят, когда мы не заботимся о том, как много совпадений может произойти, но мы могли бы хотеть иметь ограничение. Например, должно быть не более пяти, или должно быть не менее двух совпадений. Можем ли мы выполнить это с помощью некоторой комбинации? Метасимвола? Например, чтобы указать "не более пяти ", мы могли бы использовать" ????? "? К сожалению, мы не можем совместить знаки вопроса в этом случае. Первый знак вопроса относится к символу, но остальные знаки вопроса применяются к предыдущему знаку вопроса.

Однако, мы могли бы поместить символ (период для "любого символа"), за которым следует знак вопроса и повторить это пять раз , как в:

```
.?.?.?.?
```

Это регулярное выражение применяет каждый знак вопроса к периоду. И так как знак вопроса означает "0 или 1", это то же самое, как говорят любой один, два, три, четыре, или пять символов. Но как же мы заставим символы стать именно этими символами? А что если использовать [[:visible:]]?[[:visible:]]?[[:visible:]]?[[:visible:]]?[[:visible:]]? К сожалению, как и с периодом, символ в каждом [[:visible:]] может быть любым видимым

символом, но не обязательно нужным.
Наше решение - использовать другой метасимвол, в данном случае, {n,m}. Здесь, n и m являются положительными целыми числами с n меньше m. Эта запись гласит, что предыдущий символ будет соответствовать вхождениям между n и m. То есть, " совпадение с, по крайней мере, n, но не больше , чем предыдущий символ". Вы можете опустить либо "по крайней мере", либо "не более", или вы можете указать одно значение для обеспечения "точно".
Например, 0 {1,5} 1 * будет означать "от одного до пяти 0s" с последующим любым количеством 1s, в то время как [01] {1,5} означает "от одной до пяти комбинаций 0 и 1". В этом последнем случае, нам все равно в какой последовательности появятся 0s и 1s. Следовательно, это последнее выражение будет соответствовать 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, вплоть до пяти полных символов.
Мы будем использовать {2}, чтобы указать "по крайней мере, два" и {5}, чтобы указать "не более пяти". С использованием {n,m}, мы можем ограничить число совпадений с конечным числом. Рассматривать 0 {5} 1 {5}. Это будет соответствовать только 0000011111. Тем не менее, [01] {5} будет соответствовать любой комбинации из пяти 0s и 1s.
Следует отметить, что использование {} доступно только при использовании расширенного регулярного выражения. По умолчанию, утилита grep не использует расширенный набор метасимволов. Для того, чтобы использовать {} в grep, вам придется использовать расширенную grep. Это egrep или grep-E. Мы увидим это более подробно в Утилите grep.
Давайте объединим все идеи, которые мы видели до этого момента, и напишем регулярное выражение, которое будет соответствовать номеру социального страхования. Номер социального обеспечения имеет вид ### - ## -####, где каждый # это цифра. Регулярное выражение для соответствия такому номеру приводится ниже:

```
[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
```

Это регулярное выражение состоит из цифры, цифры, цифры, дефиса, цифры, цифры, дефиса, цифры, цифры, цифры и цифры. Используя {n}, мы можем сократить выражение:

```
[0-9]{3}-[0-9]{2}-[0-9]{4}
```

Как бы выглядел номер мобильного телефона? Это зависит от того, хотим ли мы, чтобы выражение соответствовало номеру с кодом города или без него, или тот, который может совпадать с номером телефона с или без кода города. Мы пока не будем отвечать на этот вопрос и вернемся к

нему в конце главы.
Давайте попробуем что-нибудь другое. Как бы мы сопоставили IP- адрес? IP-адрес имеет форму 0-255.0-255.0-255.0-255. Следующее регулярное выражение не является правильным. Можете выяснить, почему?

[0-255].[0-255].[0-255].[0-255]

Регулярное выражение IP-адреса имеет два недостатка; первый очевидный, а второй немного непонятный. Что означает [0-255]? В регулярном выражении, вы используете [], чтобы указать список вариантов. Вариантами могут быть либо нумерованный список, например, [abc], либо диапазон, как [a-c]. В квадратных скобках этого регулярного выражения содержится как диапазон, так и нумерованный список. Во-первых, есть диапазон 0-2, который будет соответствовать 0, 1 или 2. Во-вторых, есть нумерованный список 5, 5. Таким образом, каждый из заключенных в скобки элементов будет соответствовать любому из 0, 1, 2, 5 или 5.Таким образом, это регулярное выражение будет соответствовать 0.1.2.5 или 5.5.5.5 или 0.5.1.2. Оно не будет соответствовать ни 0.1.2.3 (нет 3 в скобках), ни 10.11.12.13 (все скобки указывают цифру, не многобуквенное значение, как 13).
Так как же мы укажем правильный нумерованный список? Можно ли перечислить все номера от 0 до 255? Нелегко , и мы не хотели бы, чтобы этот список содержал 256 цифр! Как насчет следующего: [0-9]{1,3}. Это выражение может соответствовать любой отдельной цифре от 0 до 9, любым двум цифрам от 00 до 99, а также любым трехзначным числам от 000 до 999. К сожалению, у нас нет простого способа для ограничения трехзначных чисел как 255 или меньше, поскольку это будет соответствовать строке 299.388.477.566. Но на данный момент, мы будем использовать эту функцию обозначения. Итак, давайте перепишем выражение:

[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}

Вторым недостатком в нашем оригинальном регулярном выражении является использование . (период). Напомним, что . означает "соответствовать любому одному символу". Если мы используем период в нашем регулярном выражении, он может соответствовать чему угодно. Таким образом, наше новое регулярное выражение может соответствовать 1.2.3.4 или 10.20.30.40 или 100.101.201.225, но она может также соответствовать 1a2b3c4 или 1122334 или 1-2-3-4, а также многим другим последовательностям, которые не являются IP-адресом. Наша проблема состоит в том, что мы не хотим, чтобы период воспринимался как метасимвол; мы хотим, чтобы период воспринимался буквально.

Как мы указываем буквенный символ? Большинство символов нужно просто перечислить, чтобы воспринимать их буквально. Например, abc буквально считается строкой “abc”. Но если сам символ является метасимволом, мы должны сделать что-то особенное. Есть два варианта, первый - поместить его в []. Это подходит, хотя не используется для размещения одного символа в [], так [0-9] {1,3} [.] [0-9] {1,3} [.] [0-9] {1,3} [.] [0-9] {1,3} сработает. Когда мы хотим указать символ, который считается метасимволом, мы должны "уйти" от его смысла. Это делается с предшествующим символом \, в \. или \ + или \ {или \ $. Таким образом, наш окончательный ответ (на данный момент):

[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}

Мы постараемся исправить недостаток, который не позволяет использовать трехзначные числа больше 255.

Есть и другие виды использования символов выхода, они называются управляющие последовательности. Таблица 10.3 предоставляет список общих управляющих последовательностей. Например, если вы хотите найти четыре последовательных пробельных пространств, вы можете использовать \s{4}. Или, если вы хотите найти соответствие к любой последовательности от 1 до 10 без цифр, вы можете указать \D{1,10}.

[] имеют другое использование, хотя в этом случае их труднее правильно применить. Если вы поставите ^ перед нумерованным списком в [], список теперь интерпретируется в "совпадает, если ни один из этих символов не присутствует ". Вы можете использовать [^0-9]*, чтобы найти строку, которая не содержит никаких цифр или [^ A-Z] *, чтобы найти строку, которая не содержит букв верхнего регистра. Выражение [A-Z][a-z]+[^A-Z]* найдет строку, которая должна иметь прописную букву, некоторое количество букв нижнего регистра, но без каких-либо дополнительных букв верхнего регистра. Это может получиться, если мы ожидаем, что имя человека, например, может быть записано как ZaP-Pa.

Почему трудно использовать [^ ...]? Чтобы объяснить это, мы должны сначала пересмотреть, с чем совпадают регулярные выражения. Помните, что регулярное выражение - это строка, которая используется для сопоставления с другой строкой. В самом деле, то, чему будет соответствовать регулярное выражение, это подстрока большей строки. Подстрока это всего лишь часть строки. Например, если строка “Frank Zappa”, одно из следующих действий будет считаться подстрокой: “Frank”, “ank”, “Zappa”, “kZ”, “ppa”, “Frank Zappa”, и даже “” (пустая строка).

ТАБЛИЦА 10.3 Общие управляющие последовательности

\d	Совпадает с любой цифрой
\D	Совпадает с любой не-цифрой
\s	Совпадает с любым пробельным символом
\S	Совпадает с любым не-пробельным символом
\w	Совпадает с любой буквой или цифрой
\W	Совпадает с любой не-буквой, не-цифрой
\b	Совпадает с любой границей слова
\B	Совпадает с любой не-границей слова

Рассмотрим выражение 0{1,2}[a-zA-Z0-9]+. Это регулярное выражение будет соответствовать любой строке, которая состоит из одного или двух 0s с последующей любой комбинацией букв и цифр. Теперь рассмотрим следующую строку:

0000abcd0000

Так как мы определили регулярные выражения ранее, эта строка не должна соответствовать выражению, потому что он имеет не "один или два 0s", оно в буквальном смысле имеет четыре 0s. Тем не менее, выражение только ищет любую подстроку строки, которой оно могло бы соответствовать, которая имеет "один или два 0s" с последующими буквами и цифрами ". Поскольку строка содержит "0a", она соответствует. Регулярное выражение не должно соответствовать каждому символу в строке; оно только должно найти соответствующую подстроку.

Возвращаясь к использованию [^ ...], давайте рассмотрим пример. Рассмотрим [^A-Z]+. Значение этого выражения кажется ясным: соответствует чему-то, что не содержит заглавных букв . Теперь рассмотрим строку abCDefg. Казалось бы, что регулярное выражение не должно соответствовать этой строке. Но регулярное выражение на самом деле говорит "не совпадают заглавные буквы", но строка также содержит строчные буквы. Таким образом, регулярное выражение будет соответствовать ab и efg из строки, поэтому строка найдена, как соответствующая. Как использовать [^AZ]+ затем, если оно соответствует строке, содержащей заглавные буквы? Единственная строка, которой это регулярное выражение не будет соответствовать, это любая строка, которая состоит только из букв верхнего регистра. Так, хотя оно соответствует abCDefg, оно не будет соответствовать ABC-DEFG. Нужно быть очень острожными, для того, чтобы в полной мере использовать [^ ...].

Имея это в виду, рассмотрим 0 + 1 +. Оно будет соответствовать 0001,

01111, 01, но он также будет соответствовать 0101 и 000111aaa, потому что эти две строки содержат последовательность, которая соответствует 0 + 1 +. Так как мы можем обеспечить соответствие точно по выражению? Для этого нам нужны два дополнительных метасимвола ^ и $. ^, как показано выше, может использоваться внутри [] и означает "совпадает, если эти не найдены". Но за пределами [], ^ означает "совпадает в начале строки" и $ означает "совпадает в конце строки". Если наше регулярное выражение имеет вид ^выражение $, то это означает, соответствующая строка может иметь только такой формат.

Мы можем захотеть, чтобы любые строки, которые начинаются с цифр, соответствовали. Этого можно достигнуть через регулярное выражение ^ [0-9] +. Мы используем + вместо *, поскольку * означает "совпадает ни с одним из них" , так что это будет соответствовать строке, которая не начинается с цифр. Мы можем захотеть, чтобы любые строки, которые заканчиваются на аббревиатуры штатов, соответствовали. Все государственные сокращения - это две заглавные буквы, так что это будет выглядеть как [[A-Z][A-Z]$, или [A-Z]{2}$. Если мы захотим аббревиатуру с окончанием в виде периода, мы могли бы использовать [A-Z] [A-Z] \. $, и если мы хотим сделать период по желанию, мы могли бы использовать [A-Z] [A-Z] \.? $. Обратите внимание, что с помощью [A-Z][A-Z], мы также находим соответствие в любых двух буквах в верхнем регистре, так, например, AB и ZZ, которые не являются юридическими государственными сокращениями.

В общем, мы не хотим использовать оба метасимвола ^ и $ в одном выражении, потому что это было бы слишком ограничивающе. Для демонстрации понятий, рассматриваемых до сих пор, давайте рассмотрим файл, содержащий информацию о сотрудниках. Информация подана строка за строкой, фамилия, имя каждого сотрудника, должность, год принятия, номер офиса и домашний адрес.

Напишем регулярное выражение, чтобы найти всех сотрудников, нанятых в конкретный год, скажем 2010. Наше регулярное выражение может иметь вид просто 2010. Однако, использование только 2010 может привести к ошибочным совпадениям, потому что строка 2010 может появиться как номер офиса человека или как часть адреса (адрес, почтовый индекс).

Что делать, если мы хотим найти сотрудников, нанятых с 2000 года? Регулярное выражение для этого может быть таким 20[01] [0-9]. Опять же, это может соответствовать части служебного номера или адреса. Если мы используем ^20 [01] [0-9]$, как наше решение, мы ограничиваем

совпадения строками, которые состоят только из четырех цифр, число между 2000 и 2019. Никакая строка не будет соответствовать, так как ни одна не имеет четырехзначного номера. Мы должны потупить еще умнее.

Обратите внимание на то, что год следует за фамилией, именем и должностью. Мы могли бы представить фамилию, как [A-Z] [A-Z] +. Мы могли бы представить имя, как [A-Z] [A-Z] * (при условии, что мы допускаем написание как 'J'). Мы могли бы представить должность, как [A-Za-Z] +, то есть любой комбинацией букв. И, наконец, мы ожидаем получить год принятия, который в этом случае должен быть 20[01][0-9]. Если каждая из частей информации о работнике написана через запятую, мы можем затем построить наше выражение таким образом, чтобы каждая часть заканчивалась запятой. Смотрите Рисунок 10.1. Заметьте, что год принятия, как показано здесь, включает года в будущем

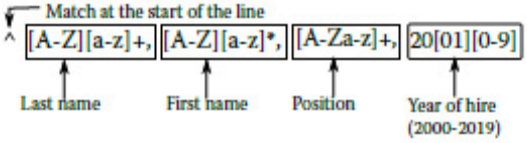


Рис. 10.1.
Обозначает «Год найма с 2000.»
Match at the start of the line – Совпадение в начале линии; last name – Последнее наименование; first name - Первое наименование; positions – Позиция; year of hire (2000-2019) - Год найма (2000-2019);

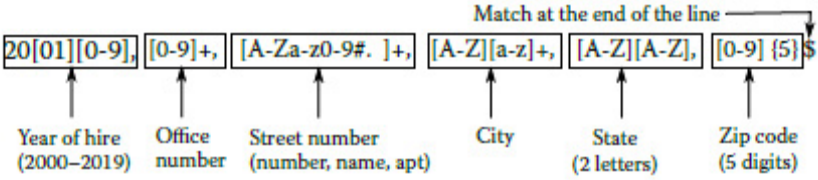


Рис. 10.2.
Возможное регулярное выражение, чтобы соответствовать адресу
Match at the start of the line – Совпадение в начале линии; year of hire - Год найма (2000-2019); office number - Номер офиса; sreet number (number, name, apt) - Номер улицы (номер, название, кв.); city – Город; state (2 letters) - Штат (2 буквы) ; zip code (5 digits) - Индекс (5 цифр)

(до 2019 года через).
Мы также могли бы решить эту проблему поиска с другого конца ряда. Год принятия стоит перед номером офиса и адресом. Если мы предположим, номер офиса состоит только из цифр, и что адрес состоит из адреса, города, штата и почтового индекса, а затем, чтобы завершить строку, мы ожидаем увидеть 20[01][0-9],[0-9]+, и адрес. Адрес более сложный, потому что он состоит из нескольких различных компонентов, уличного адреса, города, аббревиатуры (две буквы) и почтового индекса. Сам адрес улицы может иметь структуру цифр с буквами и пробелами (например, 901 Pine Street). Также могут быть периоды, появляющиеся в адресе (например, 4315 E, Magnolia Road), или он может включать в себя цифры в названии улицы (например, 50 N. 10th Street). Она также может иметь номер квартиры, который использует символ # (например, 242 Olive Blvd, # 6). Мы могли бы попытаться решить все это путем размещения каждой потенциальной буквы, цифры, символ и пробела в одном из перечисленных списке, как и в [A-Za-z0-9.#]+. Название города должно состоять просто из букв (хотя также может включать пробелы и периоды), но мы будем принимать во внимание только буквы. Рисунок 10.2 представляет полное регулярное выражение, которое можно было бы использовать для завершения строки, которая начинается с года принятия, по крайней мере, 2000, но только с пятизначным почтовым индексом (мы увидим, как обращаться с девятизначным почтовым индексом ниже).
Последние метасимволы - | и ( ). Использование этих метасимволов обеспечивает последовательность символов, где какая-либо одна последовательность может совпадать. То есть, мы можем перечислить список предметов. В отличие от нумерованного списка в [], здесь мы перечисляем последовательности. Подумайте, мы хотим найти соответствие любому из этих трех 2-буквенных сокращений: IN, KY, OH. Если бы мы использовали [IKO] [NYH], то оно будет соответствовать IN, KY, и OH, но и IY, IH, KN, KH, ON, и OY. Перечислим последовательности внутри () и поставим | между вариантами. Так, (IN | KY | OH) в буквальном смысле будет соответствовать только одной из этих трех последовательностей.
Используя () и |, мы можем обеспечить выражение, которое соответствовало бы как пятизначному, так и девятизначному почтовому индексу. Пятизначное выражение - [0-9] {5}. Девятизначное выражение - [0-9] {5} -[0-9] {4}. Мы сочетаем их с помощью | и помещаем их в (). Это дает нам ([0-9] {5} | [0-9]{5} - [0-9] {4}).

Точно так же мы можем использовать () и |, чтобы решить проблему IP-адреса. Напомним, что наше решение с использованием [0-9] {1,3} будет соответствовать трехзначному числу не больше 255, и, таким образом, будет соответствовать числам, которых не было в IP-адресе. Мы не хотели бы перечислять все 256 возможных значений, как в (0 | 1 | 2 | 3 | ... | 254 | 255), но мы могли бы использовать () и | другим способом. Подумайте, что [0-9] будет соответствовать любой из цифр, и IP-адреса могут включать в себя любую однозначную цифру. Мы могли бы также использовать [0-9] [0-9], потому что любая последовательность 00-99 является законным IP-адресом, хотя мы обычно не используем ведущие нули. Таким образом, мы могли бы упростить это до [0-9] {1,2}. Тем не менее, это не включает в себя последовательность 100-255. Мы могли бы выразить 100-255 несколькими различными возможностями:

1[0-9][0-9]—this covers 100–199
2[0-4][0-9]—this covers 200–249
25[0-5]—this covers 250–255

Давайте закончим этот раздел целым рядом примеров. Во-первых, мы опишем некоторые строки, которые мы хотим соответствовать и придумаем регулярные выражения, которые соответствовали бы им. Во-вторых, у нас будут некоторые регулярные выражения, и мы попытаемся объяснить, чему они соответствуют. Предположим, что у нас есть текстовый файл, который содержит список информации о студенте следующего вида:
Student ID (a 16 digit number), last name, first name, major, minor, address.
Основные и второстепенные специальности будут иметь трехбуквенный

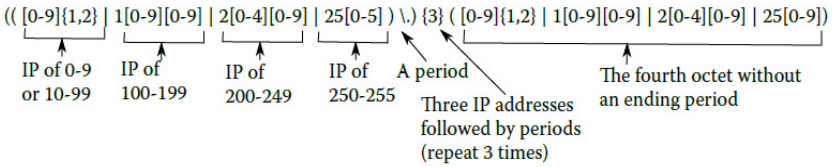


Рисунок 10.3
помещает эти варианты вместе в одно длинное регулярное выражение (пробелы вставляются вокруг "|" символов, чтобы сделать их немного более удобными для чтения).
Перевод: three IP addresses followed by periods (repeat 3 times) - Три IP адреса сопровождаемые периодами (повторить 3 раза); the fourth octet without an ending period - период Четвертый октет без оканчивающегося периода

код (например, CSC, CIT, BIS, MIN), все заглавными буквами. Второстепенная специальность необходима, но она может включать в себя три пробела для указания "специальность не выбрана ". Адрес - адрес, город, штат, почтовый индекс.

Мы хотим найти всех студентов, которые имеют специальность в информатике (CSC) или компьютерных информационных технологиях (CIT). Очевидный ответ просто искать (CSC | CIT).Однако, это не делает различие между основной и второстепенный специальностью, мы хотим найти только основную специальность. Заметьте, что за основной специальностью следует 16-значный номер и имя. Так, например, мы могли бы ожидать увидеть 0123456789012345, Zappa, Frank, CSC, ... Тогда мы могли бы написать регулярное выражение, начиная с начала строки:

^[0-9]{16}, [A-Z][a-z]+, [A-Z][a-z]+, (CSC|CIT)

Мы можем его сократить. Поскольку перед второстепенной специальностью всегда стоит основная, которая состоит из трех заглавных букв, что мы ожидаем увидеть перед основной, но не второстепенной специальностью, это имя, которое не полностью капитализируется. Таким образом, мы могли бы уменьшить выражение следующим образом:

[A-Z][a-z]+, (CSC|CIT)

Здесь нам необходима последовательность прописной буквой с последующими строчными буквами (имя), а затем через запятую следует один из CSC или CIT. Если CSC или CIT соответствуют второстепенной специальности, строка, которая ей предшествует, не будет включать в себя строчные буквы, а если после прописных букв следуют строчные буквы, которые соответствуют фамилии или почтовому адресу, за ним не будет ни CSC, ни CIT.

Мы хотим найти всех студентов, которые живут в квартирах. Мы делаем предположение, что квартиры указаны в адресной части, используя либо apt, apt., # или apartment. С помощью [.]? (Или \.?), мы можем указать, что период после того apt не является обязательным. Мы можем перечислить их следующим образом:

([Aa]pt[.]?|[Aa]partment|#)

Мы не ожидали, что сокращение KY или OH появится где-нибудь в адресной строке. Тем не менее, возможно использовать KY или OH, как часть из трех букв аббревиатуры. Если мы хотим проводить поиск в безопасности, мы будем считать, что сокращение сразу появляется перед почтовым индексом, который заканчивает строку. Для этого, мы могли

бы указать:
(KY|OH), [0-9]{5}$
или если предположить, что почтовый индекс будет пятизначным и расширение четырехзначным, мы могли бы использовать:
(KY|OH), ([0-9]{5}|[0-9]{5}-[0-9]{4})$
В качестве альтернативы, так как аббревиатура появится после того, как мы поставим пробел, и если она заканчивает период, мы можем добавить пробел и период, так что трехбуквенная аббревиатура не будет соответствовать:
(KY\.| OH\.)
Если за главное брать SKY или OHM, это не будет соответствовать аббревиатурам этих штатов из-за пространства и периода, которые окружают две буквы.

Спам-фильтры и регулярные выражения

Если вы хотите создать спам-фильтр, чтобы отфильтровать нежелательные сообщения электронной почты , то это простая задача. Нужно написать программу, которая будет искать через электронную почту (текстовый файл) некоторые слова: "наличные деньги", "время действовать", "похудеть", "виагра", "работа на дому", "вы были выбраны" и так далее. Но некоторые спамы проходят, пытаясь скрыть ключевые слова.
Рассмотреть спам по электронной почте, рекламу, дешевые и доступные таблетки виагры. Адрес электронной почты можно попытаться замаскировать, например, слово «Виагра» можно замаскировать под множеством различных форм: V1agra, V.i.a.g.r.a, Vi@gra, V!agra, Via-SEXYgra и так далее.
Если мы беспокоимся только о буквенных заменах , мы могли бы попытаться перечислить все возможные замены в нашем регулярном выражении, такие как:
[Vv][iI!][Aa@][gG9][Rr][aA@]
Что насчёт замен, характер которых нам неизвестен, например, вместо 3 - ‘e’ или вместо 1 или ! - ‘i’? Например, V##gra или Viag^^? Здесь нам нужно быть осторожней с нашими регулярными выражениями. Мы можем использовать, например, [Vv].{2} gra, [Vv]iag.{2}, и другие варианты, но нужно быть внимательным, чтобы не заблокировать легитимные слова. Например , [Vv]ia.{3} может не быть спамом.
Как насчет версии этого слова, в которое вставлены дополнительные символы, как V.i.a.g.r.a., ViaSEXYgra, или Via##gra? Для решения

этой проблемы, мы можем вставить обозначения. * между буквами. Напомним, что . обозначает любой символ и * обозначает 0. Таким образом, V.*i.*a.*g.*r.*a.* нужно искать совпадения в буквах ‘V’, ‘i’, ‘a’, ‘g’, ‘r’, ‘a’ независимо от того, что может появиться между ними. Из-за использования *, мы можем также сравнивать строки, где нет ничего между этими буквами.
Без регулярных выражений, спам-фильтры были бы гораздо менее успешными. Но, как вы можете видеть, определение регулярных выражений может быть сложной задачей для фильтра!

Мы хотим найти всех студентов, чьи почтовые индексы начинаются на 41. Проще простого:
41[0-9]{3}
Хотя может совпадать только часть номера индекса (например, 0123412345678901 2), или поисковик может перепутать индекс с адресом, улицей или домом. Поэтому нужно перенести код на конец строки и исправить это с помощью:
41[0-9]{3}$
Или может показать четырехзначное расширение:
(41[0-9]{3}|41[0-9]{3}-[0-9]{4})$
Мы хотим найти любого студента, чей ID заканчивается четным числом. Мы не можем просто использовать [02468], потому что это будет соответствовать любой строке, которая содержит любую из этих цифр, в любом месте (в идентификационном номере, в адресе, в почтовом индексе). Но число студентов всегда состоит из 16 цифр, поэтому мы хотим найти только 16-значный ID:
[0-9]{15}[02468]
Мы могли бы в начале поставить ^, чтобы убедиться, что мы ищем только ID (чтобы предотвратить такое, что поисковик может выдать нам, к примеру, адрес, что состоит из 16 цифр или больше).

Последний пример. Мы хотим найти учеников, чьи фамилия Заппа. Это просто:
Zappa
Хотя, что такое Zappa? Это слово отображает имя человека или его адрес? Вряд ли, конечно, но может быть. Фамилия студента всегда пишется после его ID. Поэтому мы можем убедиться в соответствии студента с его ID : ^[0-9]{16}, Zappa.
Давайте посмотрим на это с другой стороны. Вот некоторые регулярные

выражения. Будут ли здесь ошибки в соответствии?
([^O][^H], [0-9]{5}$|[^O][^H], [0-9]{5}-[0-9]{4}$)
Мы нашли всё, что не имеет аббревиатуры OH, что пишется после пяти или девятизначного номера в конце строки. Таким образом, мы определили студентов, которые не проживают в Огайо.
[A-Z]{3},[ ]{4},
Эта строка будет находить три заглавные буквы, что следуют за запятой и четырьмя пробелами (пробел после запятой), за которыми следует запятая.

Мы ожидаем увидеть только три заглавные буквы для главного или второстепенного результата поиска.
^0{15}[0-9]
Здесь мы ищем строку, которая начинается 15 нулями, и после идут любые другие цифры. Если номер ID студента был оформлен правильно, то можно идентифицировать первых 10 студентов.

Командный интерпретатор и подстановочные знаки

Вспомним из главы 9, что интерпретатор Bash выполняет несколько шагов при выполнении инструкции. Среди этих шагов множество расширений. Один из них – расширение имени файла. Если пользователь указывает групповой символ в имени файла (или имя пути), Bash разворачивает символ подстановки в список соответствующих файлов или каталогов. Затем этот список передается вместе с программой. Например, с Ls * .txt, то команда Ls не выполняет конвертирование , но вместо этого Bash конвертирует * .txt в список всех файлов, которые соответствуют, а затем предоставляет весь список в Ls. Команда Ls, как многие команды Linux, могут работать на одном элементе или списке одинаково.
К сожалению, символы, используемые для обозначения символов подстановки в Bash такие же, как некоторые из регулярных выражений метасимволов. Это может привести к путанице, особенно, так как большинство пользователей изучают групповые символы, прежде чем они узнают регулярные выражения. Так как вы изучили карты первыми, вы, вероятно, интерпретируете * в Ls * .txt как "соответствовать чему-нибудь".

Как уже говорилось о метасимволах, метасимвол * означает

"соответствовать предыдущему символу ноль или более раз". В случае Ls * .txt, предыдущий символ представляет собой пустое пространство. Это было бы неправильное толкование. Поэтому, как пользователь Linux или администратор, вы должны быть в состоянии отличить использование символов подстановки, используемых в Bash для выполнения глобирования от того, как они используются в качестве регулярных выражений в таких программах, как Grep.

Подстановочные знаки, используемые в Bash – *, ?, +, @, !, \, [ ], [^…], и [[…]].
Некоторые из них включены в обычный набор выражений метасимволом, а некоторые нет. Мы рассмотрим использование общих групповых символов в данном разделе. Таблица 10.4 обеспечивает объяснение для каждого из них. Обратите внимание, что отмеченные в таблице групповые символы, которые доступны только, если вы установили Bash, чтобы использовать расширенный набор шаблонов соответствия

Таблица 10.4 Значения метасимволов в Bash

*	Соответствует любой строке, включая пустую
**	Соответствует всем файлам и каталогам
**/	Соответствует каталогам
?	Соответствует любому одному символу (примечание: не соответствует 0 символов)
+	Соответствует одному или более вхождениям (подобно тому, как он используется в регулярных выражениях)
@	Соответствует любому примеру из списка
!	Не соответствует ничему, кроме одного примера из списка
\	Используется, чтобы избежать сходства смысла данного символа, с регулярными выражениями, например, * означает не совпадать с *
[…]	Соответствует любому из закрытых символов, диапазоны допустимы при использовании дефиса, если первого символа после, он соответствует любому символу, который не заключен в скобки
{…}	Как и в случае с расширением в Bash, списки могут быть помещены в {}, чтобы указать "собрать все", как и в Ls {C, H} *
[[:class:]]	Как и с регулярными выражениями, соответствует любому символу в заданном классе.

символов. Как мы предполагаем, это не было установлено в вашей среде Bash, мы не будем смотреть на те символы, хотя их значение должно быть понятно.
Мы закончим этот раздел некоторыми примерами, которые используют некоторые из подстановочных символов из таблицы 10.4. Мы опускаем групповые символы, которые из расширенного набора. Для этого примера предположим, что у нас есть следующие файлы и подкаталоги в текущем рабочем направлении. Подкаталоги указаны с / перед их именем.

foo foo.txt foo1.txt foo2.dat foo11.txt /foo3 /fox /foreign /foxr FOO FOO.txt FOO1.dat FOO11.txt foo5?.txt /FOO4

Смотрите таблицу 10.5, которая содержит каждый пример. В таблице приведены Linux Ls команды и элементы из каталога.

Программа Grep

Программа Grep ищет один или несколько текстовых файлов для строк, которые соответствуют заданному регулярному выражению. Он печатает строки, где найдены такие файлы. Таким образом, пользователь или администратор может быстро получить строки из текстовых файлов, которые соответствуют желаемой структуре.

Таблица 10.5 Примеры

ls команды	**Элементы**
ls *.txt	foo.txt, foo1.txt, foo11.txt, FOO.txt, FOO11.txt, foo5?.txt
ls *.*	foo.txt, foo1.txt, foo2.dat, foo11.txt, FOO.txt, FOO1.dat, FOO11.txt, foo5?.txt
ls *	Перечисляет весь список в каталоге
ls foo?.*	foo1.txt, foo2.dat
ls foo??.*	foo11.txt, foo5?.txt
ls *\?.*	foo5?.txt
ls *.{dat,txt}	Перечисляет весь список в каталоге с окончанием either.txt or.dat
ls foo[0-2].*	foo1.txt, foo2.dat
ls *[[:upper:]]*.txt	FOO11.txt, FOO.txt
ls *[[:upper:]]*	FOO, FOO11.txt, FOO1.dat, FOO.txt,/FOO4

ls *[[:digit:]]*	Перечисляет каждый элемент, что содержит цифру
ls foo[[:digit:]].*	foo1.txt, foo2.dat (не перечисляет foo11.txt, потому что мы ищем только 1 цифру, не перечисляет foo5?.

Это еще в конце раздела о метасимволах мы смотрели на регулярные выражения, используемые для определения конкретных студенческих записей в файле. Как Grep может работать с несколькими файлами одновременно, Grep возвращает две вещи для каждого совпадения, имя файла и строку, которая содержала совпадение. С помощью опции -n, вы также можете получить номер строки для каждого совпадения. При использовании Grep, в зависимости от вашего регулярного выражения и строки в файле, программа может вернуть весь файл, если соответствует каждая строка, несколько строк файла, или никакие строки не соответствуют вообще.
Программа Grep/ задает расширение следующим образом:
grep pattern filename(s)
Если вы хотите использовать несколько файлов, вы можете использовать либо * или ?, как уже отмечалось выше, в части 2, или вы можете перечислить несколько имен файлов, разделенных пробелами. Если ваш шаблон содержит пробелы, необходимо заключить шаблон в " или "". Это хорошая привычка всегда использовать " или "" в своих регулярных выражениях в качестве меры предосторожности.

На самом деле, использование " является наиболее предпочтительным. Это происходит потому, что интерпретатор Bash уже интерпретирует несколько символов таким образом, что GREP не может. Рассмотрим утверждение Grep !! имя файла. Это утверждение кажется простым, поиск имени файла для персонажей !!. К сожалению, хотя, !! сигналы для интерпретатора Bash, и последние команды должны быть отозваны. Представьте себе, что инструкция была cd ~. Так как интерпретатор Bash разворачивает такие элементы, как !! перед выполнением команды, команда меняется на grep cd ~ имя файла. Таким образом, Grep будет искать имя файла для последовательности символов cd ~.

Другой пример происходит с $. Вы можете вспомнить, что $ предшествует имени переменных в Bash. Как и в случае !!, интерпретатор Bash заменит переменные с их значениями перед выполнением команды. Таким образом, команда Grep $ HOME имя файла будет заменено Grep/home/ имя пользователя имя файла. Чтобы обойти эти проблемы, одиночные

кавычки вызывают интерпретатор Bash, чтобы избежать любого раскрытия или интерпретации. Дважды цитируя не предотвратить эту проблему, так как "$ HOME" по-прежнему преобразуется в значение, хранимое в $ HOME.

Grep пытается сопоставить регулярное выражение для каждой строки данного файла. Это, возможно, не так, как мы изначально предполагали об использовании регулярных выражений, так как мы описали регулярное выражение, как соответствие строкам, а не линиям. В сущности, Grep рассматривает каждую строку файла в виде строки, и сравнивает соответствия с регулярным выражением любой подстроки. Либо что-то в шаблоне совпадает в этой строке или нет. Программа Grep не будет искать каждую отдельную строку файла (предполагается, что строки разделяются пробелом).
Например, если файл имеет строку:
bat bait beat beet bet bit bite boot bout but
Мы использовали регулярное выражение b.t, с тех пор, как b.t соответствует как минимум одной строке, Grep возвращает всю строку.
Программа Grep использует регулярные метасимволы. Она не включает так называемые расширенные регулярные наборы выражений, которые включают в себя {}, |. Тем не менее, Grep имеет опцию, ГРЭП -E (или программы), задать расширение, которое использует расширенный набор.
Программа Grep| как задать расширение:
tokenizer filename | grep b.t
Заметьте, что в таком случае, Grep не получает имя файла (ов), но вместо того, информация о файле поступает непосредственно к нему.
Другое дело, иметь в виду необходимость использовать \, чтобы избежать значения символа. Мы рассмотрели это раньше (метасимволы), когда нам нужно было в буквальном смысле искать символы в строке, где символ был метасимволом.
Например, $ означает "конец строки", но если мы искали $, мы могли бы указать это как \ $. Это верно в Grep, как и при задании регулярных выражений в других наборах Tings. Тем не менее, есть исключения в Grep. Например, если дефис запрашивается в качестве одного из списка элементов, это может выглядеть следующим образом: [@ #% & - = <>!]. Но помните, что дефис внутри из [] используется для обозначения диапазона; так что вместо этого, мы должны указать этот список, как [@ #% & \ -! = <>]. \ - используется для указания того, что дефис ищется

в буквальном смысле, а не для использования в качестве диапазона. Но есть исключение из этого правила. Если дефис появится в конце списка, нет необходимости использовать \, так что список может быть [@ #% & = <> -!].

Есть и другие случаи, когда экранирующий символ (\) не нужен. Одним из них является список символов, которые включают в $, но $ не в конце списка символов. Если мы намерены использовать $, чтобы отметить "конец строки", Grep не ожидает каких-либо знаков после него. Поэтому, если мы имеем $ [0-9] + (указать сумму в долларах), Grep принимает $ как истинный знак доллара, а не как "конец соответствия строки". То же самое относится и к ^. Наконец, большинство символов теряют смысл, если метасимвол находится внутри [], поэтому, например, мы не должны вводить [\ $] или [\?], если мы ищем в $ или ?; вместо этого, мы можем просто использовать [$] или [?].

Рассмотрим Grep (задать расширение) в настоящее время. На рисунке 10.4 показана часть результатов применения команды -n задать расширение [0-9] {1,3} \. [0-9] {1,3} \. [0-9] {1,3} \ . [0 -9] {1,3} / и т.д. / *. Регулярные выражения здесь те, которые мы разработали ранее, чтобы получить четыре числа, разделенных точками, то есть, наши четыре октета для получения IP-адреса. Это регулярное выражение потенциально

```
/etc/Muttrc:macro pager   <f1> "!less /usr/share/doc/mutt-1.4.2.2/manual.txt\n" "Show Mutt docume
ntation"
/etc/Muttrc:macro generic <f2> "!less /usr/share/doc/mutt-1.4.2.2/manual.txt\n" "Show Mutt docume
ntation"
/etc/Muttrc:macro index   <f2> "!less /usr/share/doc/mutt-1.4.2.2/manual.txt\n" "Show Mutt docume
ntation"
/etc/Muttrc:macro pager   <f2> "!less /usr/share/doc/mutt-1.4.2.2/manual.txt\n" "Show Mutt docume
ntation"
/etc/ntp.conf:restrict 127.0.0.1
/etc/ntp.conf:#restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap
/etc/ntp.conf:#broadcast 192.168.1.255 key 42           # broadcast server
/etc/ntp.conf:#broadcast 224.0.1.1 key 42               # multicast server
/etc/ntp.conf:#multicastclient 224.0.1.1                # multicast client
/etc/ntp.conf:#manycastserver 239.255.254.254           # manycast server
/etc/ntp.conf:#manycastclient 239.255.254.254 key 42    # manycast client
/etc/ntp.conf:server    127.127.1.0     # local clock
/etc/ntp.conf:fudge     127.127.1.0 stratum 10
Binary file /etc/prelink.cache matches
/etc/resolv.conf:nameserver 172.28.102.11
/etc/resolv.conf:nameserver 172.28.102.13
/etc/resolv.conf:nameserver 10.11.0.51
/etc/resolv.conf:nameserver 10.14.1.10
/etc/resolv.conf.predhclient:nameserver 172.28.102.11
/etc/resolv.conf.predhclient:nameserver 172.28.102.13
/etc/resolv.conf.predhclient:nameserver 10.11.0.51
/etc/resolv.conf.predhclient:nameserver 10.14.1.10
/etc/termcap:# GNOME Terminal 1.4.0.4 (Redhat 7.2)
```

Рисунок 10.4

Результат из grep.

может вернуть элементы, которые не являются IP-адресом (например, 999.999.999.999), но такие строки не будут найдены в каталоге / и так далее. Рисунок 10.4 показывает не весь результат, потому что команда Grep возвращает слишком много совпадений.

Обратите внимание на то, что на выходе мы видим имя файла для каждого совпадения, номер строки совпадения, и саму строку. Вы можете заметить, что файл / и так далее. / Muttrc фактически не содержит IP-адреса; вместо этого, номера версий программного обеспечения (-1.4.2.2). Даже если бы мы использовали наши "правильные" регулярные выражения для соответствия IP-адресов (из рисунка 10.3), мы бы до сих пор соответствовали записи в файле Muttrc. Мы могли бы избежать этого, добавив пробел перед регулярным выражением IP-адреса. Команда, результаты которой показаны на рисунке 10.4, должна

Таблица 10.6. Опции grep

-a	Обработка двоичного файла, если там есть текстовый файл (это позволяет осуществлять поиск бинарных файлов для конкретных строк двоичных чисел)
-c	Подсчет количества совпадений и вывод общего количества
-d read	Используется для обработки всех файлов данного каталога, используйте рекурсии вместо чтения, чтобы прочитать все файлы данного каталога и рекурсивно для всех подкаталогов
-E	Использование расширения (позволяет расширить ряд регулярных выражений)
-e *regex*	Регулярное выражение помещается после -e, а не там, где оно обычно расположено в инструкции; это используется для защиты регулярного выражения, если оно начинается необычно, например, дефисом
-h	Подавить имя файла на выходе
-i	Игнорировать регистр (например, [a-z] соответствует любой букве в верхнем или нижнем регистре)
-L	Выводить только те файлы, что не имеют совпадений, не выводить совпадения
-m NUM	Остановить чтение файла после NUM совпадения
-n	Выводить номер строки
-o	Выводить только часть строки, где есть совпадения с регулярными выражениями
-R, -r	Рекурсивный поиск
-v	Обратить совпадения

была быть представлена корнем . Это происходит потому, что многие из файлов и так далее / не могут быть прочитаны конечным пользователем. Если бы конечный пользователь подал команду в Grep, многие из тех же элементов были бы возвращены, но следующие сообщения об ошибках также были бы возвращены:
egrep: securetty: Permission denied egrep: shadow: Permission denied egrep: shadow-: Permission denied egrep: sudoers: Permission denied egrep: tcsd. conf: Permission denied
Можно также заметить своеобразную строку перед /etc/resolv.conf . Она читает "двоичный файл /etc/prelink.cache matches". Это информирует нас о том, что двоичный файл содержал совпадения. Тем не менее, мы не хотим увидеть какой-либо двоичный выход, нам не показаны фактические совпадения внутри этого файла. Мы можем заставить Grep выводить информацию из бинарных файлов (таблица 10.6). Программа Grep имеет ряд полезных опций; некоторые распространенные из них приведены в таблице 10.6. Особо следует отметить -c, -E, -i, -n и -v. Мы обсудим позже -v.
Давайте обработаем некоторые примеры. Мы будем использовать файл информации факультета офиса для поиска различных совпадений. Этот файл, offices.txt, хранит информацию для каждого факультета, их расположение офиса (аббревиатуру здания и номер офиса), их фамилию, а затем платформу компьютера (ов), что они используют. Каждый элемент отделяется запятой, и платформа компьютера может иметь несколько элементов. Параметры для платформы - PC, Mac, Linux, Unix. Адреса офисов имеют условное обозначение из двух- или трех букв, например, ST, MEP, или GH, а затем пробел. Написать команду Grep, чтобы найти все записи на третьем этаже своего здания (при условии, что трехзначный номер офиса будет 3xx).
egrep ‘3[0-9][0-9]’ offices.txt
Написать команду Grep, которая найдет все записи преподавателей, которые имеют компьютеры PC.
egrep ‘PC’ offices.txt
Что делать, если есть здание с ПК в рамках своей трёхбуквенной аббревиатуре ? Это будет соответствовать. Мы могли бы попробовать это вместо этого:

egrep ‘PC$’ offices.txt
Это выражение будет соответствовать только строкам, где PC появится в конце строки. Тем не менее, если преподаватель имеет несколько

компьютеров и ПК будет не последним в списке, этого человека будет не хватать. Учтите, что строка будет выглядеть следующим образом: 123 ABC, Fox, PC, Mac. Если мы хотим, чтобы он соответствовал ПК, он должен соответствовать после запятой и пробела. Это позволит избежать согласования здания, например, 456 или PCA 789 APC. Таким образом, наша новая команда Grep – это:

egrep ‘, PC’ offices.txt

Обратите внимание на то, что если член факультета имеет несколько компьютеров, каждый из них перечислен как компьютер, компьютер, компьютер, и так далее. Поэтому, если у нас есть компьютер в любом месте в списке, это произойдет после запятой и пробелом. Но если PC появится в здании, он будет либо появляться после того, как цифры и пробел (как в 456 PCA), или после того, как буквы в верхнем регистре (как в 789 APC). Написать команду Grep, которая найдет все записи преподавателей, которые имеют более одного компьютера. Есть ли способ, чтобы включить "счетчик" в команде задать расширенное? В некотором смысле, да. Напомним, что мы могли бы контролировать количество совпадений, ожидаемых при использовании {п, т}. Но то, что мы хотим на самом деле рассчитывать? Давайте посмотрим на два примера строки, факультет с одним компьютером и факультет с более чем одним.

123 MEP, Newman, Mac

444 GH, Fox, PC, Linux

С одного компьютера, первая запись имеет только две запятые. С помощью двух компьютеров, второй элемент имеет три запятые. Это говорит нам, что мы должны найти строки, которые имеют, по крайней мере, три запятые. Тем не менее, ', {3}' не является адекватным регулярным выражением, потому что будет соответствовать только строке, в которой было, по крайней мере, три последовательных запятых (например, 456 PCA, Заппа, PC, Mac). Мы должны разрешить любые символы, чтобы предстать перед каждой запятой. На самом деле, глядя на наш пример строк, все мы заботимся о том, есть ли пробелы и буквы перед запятой (не цифры, так как только цифры происходят в начале строки). Наше регулярное выражение, то есть:

egrep ‘([A-Za-z]+,){3,}’

Обратите внимание на использование (). Это необходимо для того, чтоб {3} применялось ко всем регулярным преобразованиям (а не только предыдущего символа, который является запятой). Мы могли бы на самом деле упростить это, используя . метасимвол. Напомним, что . может соответствовать любому одному символу. Мы можем добавить +, чтобы определить один или несколько из любого типа характера. Так как все мы заинтересованы в нахождении, по крайней мере, три запятые, мы можем использовать либо '. +. +. + ", Или мы могли бы использовать (. +) {3}.Написать команду Grep, которая найдет все записи преподавателей, которые имеют либо машину Linux или Unix. Здесь мы можем использовать опцию или, как в (Linux | Unix). Мы могли бы также попытать это заклинание с помощью [] вариантов. Это будет выглядеть следующим образом: [L] [IU] п [Ui] х. Две GREP команды являются:
egrep '(Linux|Unix)' offices.txt egrep '[L]?[iU]n[ui]x' offices.txt
Написать команду Grep, которая найдет все записи преподавателей, которые не имеют ПК в своем офисе. Очевидным решением было бы использовать [^ P] [^ C] в выражении. То есть, найти строку, которая не имеет в P с последующим C. К сожалению, это выражение возвратит каждую расширенную строку файла. Зачем? Поскольку регулярное выражение задает любую строку, которая не имеет компьютер. Тем не менее, способ, которым GREP работы является то, что он буквально сравнивает линию для ПК, и если линия не совсем ПК, то строка содержит совпадение. Если линия была 444 GH, Фокс, PC, Linux это будет соответствовать, потому что есть символы в строке, которые не являются ПК, например, '44', который запускает строку. То, что мы действительно хотим иметь это регулярное выражение, которое гласит: "некоторые вещи не следуют P с последующим не C, а затем некоторые другие вещи". Создание такого регулярного выражения является сложной задачей. Вместо этого, мы могли бы просто использовать регулярные выражения ПК и добавить параметр -v в нашу команду grep . То есть,
egrep –v 'PC' offices.txt
Эта команда видна для каждой строки, которая соответствует 'PC', а затем возвращает остальные строки файла. К сожалению, если у нас есть запись с офисом PCA или APC, эта строка не будет возвращена , имеют ли они компьютер или нет. Таким образом, мы регулируем регулярное выражение, чтобы быть "ПК", чтобы избежать согласования ПК во имя здания, так что команда расширение задана:

egrep –v ', PC' offices.txt

Другие методы использования регулярных выражений

С grep / задать расширение, нам разрешено использовать полный спектр регулярных выражений метасимволов . Но мы ограничены только символами подстановки. Что делать, если мы хотим найти каталог (или набор каталогов) для определенных файлов, которые соответствуют шаблону, который может быть создан с помощью регулярного выражения, но не групповых символов? Например, что если вы хотите перечислить все файлы, чьи права были доступны только для него самого? Вспомним из главы 6, что разрешения отображаются при использовании команды -l Ls. Файл только для чтения будет иметь права доступа, которые содержали r-- где-то в списке разрешений (на самом деле, не важно, если файл устанавливается как rwxrwxr-- или r -------- или какой-то другой вариант только до тех пор, пока r-- находится где-то в разрешениях).
Для Ls, * имеет значение "ничего". Так что мы хотим, чтобы в буквальном смысле "что-нибудь", за которой следует r-- "ничего". Будет ли левая шкала -l * r - * выполнять эту задачу для нас? Нет. Давайте посмотрим, как эта команда будет работать. Во-первых, интерпретатор Bash разворачивает обозначение * r-- *. Это означает, что интерпретатор Bash получает все имена в текущей директории, что имеет какое-либо следует r с последующими двумя черточками, после чего-либо. Например, foxr - текст будет соответствовать этому шаблону из-за r-- в названии. После того, как список файлов получается, что они будут переданы команде LS, и длинный список будет отображаться. К сожалению, мы сделали это в неправильном порядке, мы хотим длинный список, а затем мы хотим применить * r - *.
Наше решение довольно легко. Чтобы получить длинный список, во-первых, мы делаем Ls -l. Затем задаем расширенный результат. Наша инструкция становится Ls -l * | задать расширение 'r--' так, что мы получим длинный список всех элементов в текущем каталоге, а затем мы переходим, список (серия строк), чтобы задать расширение, которое ищет любые строки с r-- и возвращает только те. Эта команда будет возвращать любую строку, содержащую r-- в длинном списке. Если есть файл с именем foxr - текст, он возвращается, даже если его права не совпадают r--. Как мы можем избежать этого? Что ж, обратите внимание, что разрешения являются первым делом в длинном перечислении, и имя файла является последней вещью. Мы можем написать более точное регулярное выражение и включить ^, чтобы заставить его соответствовать в начале строки.

Права доступа начинаются с типа файла. Для этого нам не все равно, что за символ мы получим, но это должен быть только один символ. Мы можем получить любой одиночный символ, используя период. Затем мы ожидаем, еще девять символов, которые будут be r, w, x, или - из этих девяти полукокса и цифр, мы будем соответствовать любому r--. Таким образом, мы могли бы использовать регулярное выражение (r-- | [RWX -] {3}). Это будет соответствовать либо r-- , или любой комбинации R, W, X, и - более трех символов. К сожалению, это не будет работать для нас, потому что это может соответствовать RWX или rw-, или даже ---. Мы могли бы вместо того, чтобы написать это выражение как (r - [RWX -] {6} | [RWX -] {3} r - [RWX -] {3} | [RWX -] {6} r--). Здесь мы требуем r-- увидеть где-то в выражении. Теперь наша команда гораздо более сложная, но она предотвращает совпадения, где r-- находится в имени файла (или в имени пользователя или имя_группы). Рисунок 10.5 иллюстрирует решение; пробелы добавляются вокруг "|", чтобы помочь с дискретностью.

Вы можете комбинировать Ls -l и искать расширение для различных вещей, таких как файлы, размер которых больше 0, файлы, которые принадлежат конкретному пользователю, или файлы, созданные в этом году. Можете ли вы придумать способ, чтобы получить длинный список файлов, размер которых больше 0, используя Ls -l и задать расширение? Этот вопрос задается в задачах обзор в этой главе (смотрите вопросы 22 и 23).

Есть целый ряд других программ, которые используют регулярные выражения за пределами Grep. Программа sed является редактором потоков. Эта программа может быть использована для редактирования содержимого файла без необходимости непосредственного манипулирования файла через редактор. Например, представьте, что вы хотите извлечь первое слово в каждой строке текстового файла. Вы можете открыть файл в vi или Emacs и делать редактирование вручную. Или, вы можете использовать программу SED. В SED, необходимо указать регулярное выражение и строку замены. Регулярное выражение описывает строку, которую вы ищете. Строка замены используется вместо строки, найденной. Вы можете указать замену буквально, или вы можете применить такие команды, как "верхний регистр" или "нижний регистр". Вы можете удалить элемент.

SED инструмент очень полезен для создания больших и быстрых замен в файле. Тем не менее, он требует твердого понимания регулярных выражений. Один простой пример, чтобы удалить все HTML-теги из

HTML-файла. Можно было бы определить регулярное выражение как "<. *> '. И заменить его 0 (или пробелом). Так как ничего в <> знаке является HTML-тег, одна команда может найти и удалить все из них. Другое использование для SED - переформатировать файл. Рассмотрим файл, где информация хранится не строка за строкой, а просто через запятую для разделения каждого элемента. Вы можете заменить запятые символами табуляции (\t) и / или символами новой строки (\ n).

Еще одна программа для отметок - AWK. Название программы инициалы трех программистов, писавших AWK. В то время как SED ищет строки для замены, AWK выполняет поиск строк для обработки. С AWK, необходимо указать пары регулярных выражений и действий. Если строка соответствует регулярному выражению, то строка обрабатывается с помощью указанных действий. Один простой пример использования AWK - вывод отдельных элементов строки, которая соответствует

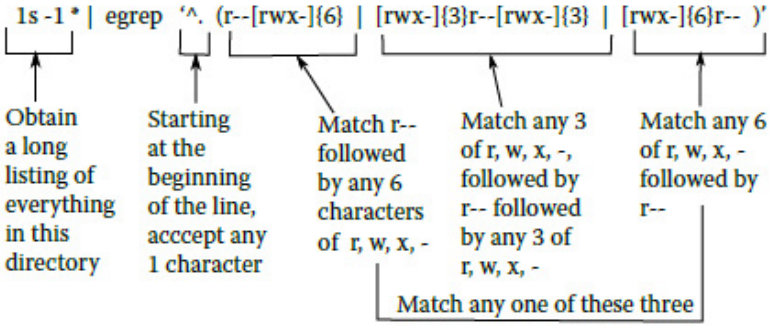


Рис. 10.5
Решение о поиске только для чтения файлов в каталоге с помощью Ls и задать расширенное.

Obtain a long listing of everything in this directory - Получить длинный список всего, что есть в этом каталоге
Starting at the beginning of the line, accept any 1 character - Начиная с начала строки, примите любой 1 символ
Match r—followed by any 6 characters of r, w, x - Подберите r-- следующий за любыми 6 символами r, w, h, -
Match any 3 of r, w, x,-, followed by r—followed by any 3 of r, w, x - Подберите любые 3 r, w, x, -, следующие за r--, следующие за 3 r, w, x, -
Match any 6 of r, w, x, - followed by r-- - Подберите любые 6 r,w,x – следующие за r --
Match any one of these three - Выберите любые из этих трех

регулярному выражению. Если задать расширение, когда совпадение найдено, то вся строка выводится, но AWK позволяет указать, что вы хотите вывести. Таким образом, AWK несколько походит на базу данных программы и все же она является гораздо более мощным, чем база данных, так как условие согласования основано на регулярном выражении.

Другой пример использования AWK - математические операции совпавших элементов. Например, представьте, что текстовый файл содержит информацию о заработной плате для сотрудников. Среди информации являются имена работников, количество отработанных часов на этой неделе, заработная плата и налоговая информация. С помощью AWK, мы можем сравнить всех сотрудников, которые работали сверхурочное время и вычислить сумму оплаты сверхурочных, что мы должны будем заплатить. Или мы могли бы сравнивать каждую запись и вычислить среднее количество отработанных часов.

Регулярные выражения были включены в оба VI и Emacs. При поиске строки, вы можете указать символьную строку, но вы также можете указать регулярное выражение. Как и с SED, это позволяет определить конкретные строки, представляющие интерес, так что вы можете отредактировать или отформатировать их.

И, наконец, регулярные выражения были включены в многочисленные языки программирования. Форма регулярного выражения была впервые введена в языке SNOBOL в начале 1960-х годов. Тем не менее, он был до 1987 года, потом регулярные выражения внесли в язык программирования Perl. Мощность Perl была главным образом сосредоточена вокруг определения регулярных выражений и хранения их в переменных файлах. Perl оказался настолько полезным, что он стал языком, который был использован для поддержки многочисленных интернет-приложений, в том числе веб-сервера сценариев. С тех пор, регулярные выражения были включены в новые языки, включая PHP, Java, JavaScript, платформы .Net, Python, Ruby и Visual Basic.

Мы закончим эту главу с двумя очень плохими шутками над регулярными выражениями.

"Если у вас есть проблемы, и вы думаете, что решите их с использованием регулярных выражений, то у вас есть две проблемы".

Вопрос: Что говорит сделать одно регулярное выражение другому?

ГЛАВА 11.
ПРОЦЕССЫ И УСЛУГИ

Управление процессом и процесс были введены в главе 4. В этой главе, контроль пользователя над процессом покрывается с точки зрения пользователя, Windows 7 и пользователь Red Hat Linux. Запуск процессов, процессов мониторинга, процессов планирования, и заканчивающихся процессов. Далее в главе рассматривается роль службы в качестве компонента операционной системы. Общие услуги рассматриваются с последующей настройкой служб в Linux. И, наконец, автоматизация запуска службы при инициализации системы рассматривается как для Windows 7, так и для Red Hat Linux.
Цели для изучения данной главы:

- Описание шаги запуска, контроль / управление и завершение процессов в Windows 7 и Red Hat Linux.

- Внедрение процесса планирования с помощью планировщика задач Windows и Linux. Команды at, batch, и crontab.

- Обсудить роль службы поддержки операционных систем.

- Описание процессов настройки служб Linux.

- Обсудить, как управлять запуском сервиса в Windows 7 и Linux.

Глава о процессе была введена в главе 4. В этой главе мы рассмотрим, как пользователи и администраторы системы могут контролировать свои процессы. Мы также посмотрим на операционную систему (ОС) услуг, которые являются фоновыми процессами, которыми также можно управлять с помощью системного администратора.

Запуск процесса

Чтобы запустить программу, программа должна быть исполняема, то

есть, программа должна быть в состоянии, на компьютерном языке. Поскольку большинство программ написаны на языке высокого уровня (например, Java, C ++), программист должен перевести программу из версии языка высокого уровня на компьютерный язык. Это, как правило, достигается за счет использования программ - компиляторов. Компилятор предоставляет исполняемую версию программы.

Вы должны составить программу для каждой платформы, чтоб программа была предназначена для работы (например, отдельный сборник необходим для Macintosh, Windows, Linux). *
Первым шагом в запуске любого процесса должно быть наличие исполняемой версии программы, которая доступна. Исполняемая программа хранится в файловой системе (как правило, на жестком диске, но также можно запускать программы, хранящиеся на оптическом диске, флэш-диске и так далее). Когда пользователь хочет запустить программу, операционная система должна найти исполняемую программу. В большинстве операционных систем, пользователь указывает на то, что программа должна проходить через один из нескольких возможных подходов.
Во-первых, это могут быть ярлыки. Ярлык представляет собой графическое представление ссылки на программы (напомним, ссылки в

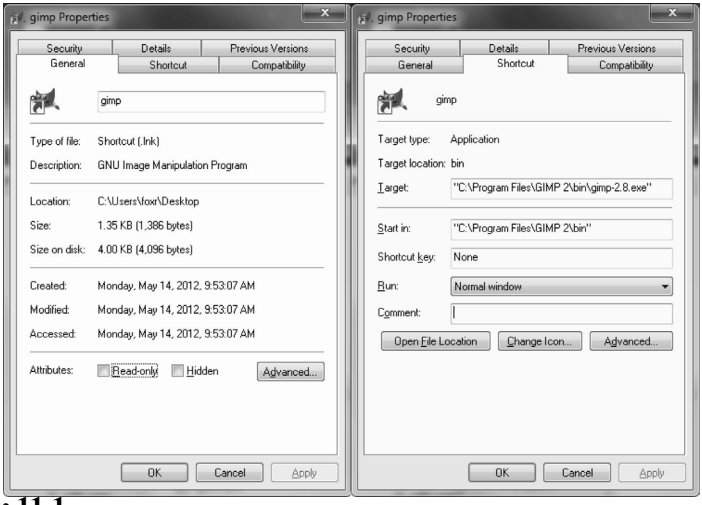


Рисунок 11.1
Две вкладки свойств двух ярлыков.

главе 5). Двойное нажатие на ярлык заставляет ОС запустить программу. Ссылка Свойства включают местоположение файла в файловой системе. В операционной системе Windows, вы можете просматривать свойства ярлыка, щелкнув правой кнопкой мыши на ярлыке и выбрав Свойства. Смотри рисунок 11.1 для примера быстрого доступа к программе Gimp (с открытым исходным кодом графического редактора). Есть несколько вкладок в окне свойств ярлыка.

Вкладка содержит местоположение самого ярлыка, в то время как вкладка Ярлык содержит местоположение исполняемого файла в поле под названием Target. Сам ярлык хранится в рабочем столе текущего пользователя, в то время как исполняемый файл находится в папке Program Files в подкаталоге Gimp \ Bin. Двойной щелчок на ярлыке заставляет операционную систему запустить программы, которые хранятся в целевом местоположении. Обратите внимание на то, что целевое окно редактирование-состояние, так что вы можете изменить местоположение исполняемого файла, если это необходимо. Вкладка Безопасность также содержит полезную информацию; она имеет права доступа к файлам для ярлыка (эти разрешения должны совпадать с разрешениями самого исполняемого файла).
Чтобы запустить программу, можно также дважды щелкнуть мышью на файле данных, созданных этой программой. Как правило, расширения имен (например, docx, .pdf, .mp3) отображаются в приложениях. Двойной щелчок на файл данных приводит к тому, ОС должна отобразить в соответствующих приложениях. Это программное приложение запускается, и файл открыт в прикладном программном обеспечении.
Большинство операционных систем также предоставляют меню программы. Многое, как ярлыки, пункты меню программы сохранения целевого расположения исполняемого файла. В операционной системе Windows, щелкнув правой кнопкой мыши на имени программы, в меню программы и выбора свойств вызывает такое же окно, как показано на рисунке 11.1.
Linux имеет меню программ и ярлыков, так же, как в Windows. Тем не менее, из Linux (а также DOS), вы можете запустить процесс из командной строки. На самом деле, будь то запуск программы из значка на рабочем столе, в меню программы, или в командной строке, все они выполняют ту же задачу таким же образом. Из командной строки, однако, вы можете предоставить дополнительные параметры. В большинстве случаев, целевое местоположение для ярлыка не включает в себя параметры.

Родительские записи, дочерние записи, висячие строки и зависшие процессы

Как описано в Главе 4, существуют процессы и потоки. Тем не менее, мы можем классифицировать данные элементы под другими именами. Процесс или поток может порождать другой процесс или поток. Дочерняя запись обычно является копией родительской записи, имеющей тот же процесс, но свои собственные данные и цели. Родительская и дочерняя записи могут разделять ресурсы, или родительская запись может ограничить ресурсы дочерней записи. Родительская запись может породить дочернюю запись для осуществления некоторой подзадачи, в то время как осуществляет контроль за ней. Фактически, родительская запись может породить множество дочерних записей и ожидать, когда они закончат свои задачи, объединяя их в конце.

В качестве примера, веб-сервер Apache запускается как единственный родительский процесс. Он порождает дочерние записи, каждая из которых имеет задачу обработать входящие запросы HTTP. В случае если родительская запись обнаруживает, что все дочерние записи заняты со своими запросами, то она может породить еще записи. И наоборот, если какие-либо из дочерних записей свободны, то родительская запись может их уничтожить.

В программах UNIX и Linux, осиротевший процесс – это дочерний процесс, который все еще существует даже после того, как его родитель был уничтожен. Это может быть, как и ошибкой, так и по специальной задумке. В любом случае, осиротевший процесс немедленно «удочеряется» специальным процессом под названием инициализация (системны процесс).

С другой стороны, дочерний процесс, завершивший свою задачу и не имеющий оснований более существовать, может стать зависшим процессом. В этом случае, процесс умирает – более не существует. Тем не менее, т.к. родительский процесс получил результаты, процесс остается в таблице процессов, пока родитель не прочитает статус дочернего процесса.

Например, в Linux, вы можете указать VI -l someprogram.lsp. Опция -l начинается VI в " lsp режиме" для поддержки редактирования программ на Лиспе. Кроме того, предоставляя имя файла в команде, VI открывает уже загруженный файл.
После того, как вы написали вашу команду, чтобы запустить программу, операционная система берет все на себя. Во-первых, она должна указать местонахождение исполняемого кода, хранящегося в файловой системе. Список каталога будет содержать указатель на исходное местоположение (дискового блока) программы. Далее, операционная система должна загрузить исполняемый код в файл подкачки, и скопировать часть исполняемого кода (обычно первые несколько страниц) в память. Если память не имеет свободных кадров, операционная система выбирает некоторые кадры для удаления. Операционная система создает таблицу страниц для процесса, и создает идентификатор процесса (PID) и структуру данных, которая описывает состояние процесса и вставляет его в очереди (готовые или ожидание, в зависимости от загрузки процессора и приоритет процесса). Процессор в конечном итоге получит этот процесс и начнет его выполнение. Это может выполниться немедленно или может занять несколько миллисекунд в зависимости от того, что другие процессы выполняют в то время.
В Linux (и Unix), процессы - либо родительские, или дочерние (или оба). Процесс, который начинает другой процесс, известен как родительский процесс. Процесс, запущенный другим процессом, известен как дочерний процесс. В качестве примера, если вы находитесь в терминале Windows и вы вводите команду, чтобы начать процесс, скажем, VI, то процесс является родителем, и этот процесс начался, VI в данном случае, является дочерним процессом. В Linux, только процесс 1, процесс создается при загрузке системы, не дочерний процесс. Все другие процессы являются дочерними процессами, но также могут быть родительскими процессами. Боковая панель на следующей странице и боковая панель в Process Execution предоставляет более подробную информацию о процессах, и как они соотносятся друг с другом.

Выполнение процесса

Во время выполнения процесса пользователь в значительной степени не вовлечен в управление процессом 'выполнение' . Пользователь может приостановить процесс, убить процесс, процесс переместить от переднего плана к заднему плану (или фону на переднем плане), и

изменить приоритет процесса. В противном случае процесс 'выполнение' находится в руках (или под контролем) ОС.

Вы можете вспомнить из главы 4, что на первом плане процесс, который может получить взаимодействие с пользователем. На первый план на самом деле представляет собой два разных понятия. Во-первых, на переднем плане процесс является процессом, который получает взаимодействие с пользователем. В обоих окнах и графический пользовательский интерфейс (ГПИ) от операционной системы Linux, на переднем плане процесс является процессом окна, который является "поверх" других процессов. Это окно иногда отмечено при наличии другого цвета заголовка. В Windows 7 вы можете заметить разницу, так как вкладка в панели задач для процесса несколько освещена, а «окно сам процесс» имеет красный" X "в верхнем правом углу. Рисунок 11.2 сравнивает передний план в сравнении с фоновыми процессами, показывая панель задач и окно (смотрите в правом верхнем углу документа Word).

Другое понятие переднего плана процессов является более точным. Приоритетными процессами являются те, которые находятся в состоянии готовности очереди. То есть, на переднем плане являются процессы, что процессор многозадачности. Фоновые процессы, то есть те, которые имеют достаточно низкий приоритет, что они ждут, пока процессор не освобождается от процессов переднего плана до ГПИ, совершающая им. Здесь мы в первую очередь сосредоточимся на прежнем понятии переднего плана.

Для перемещения фонового процесса на первый план в графическом интерфейсе, выберите его вкладку, или если части окна отображаются на

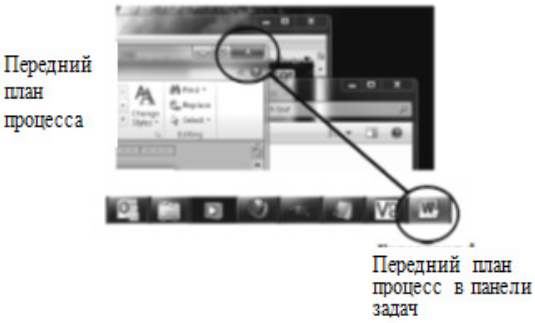


Рисунок 11.2
Передний план против фона в Windows 7.

рабочем столе, щелкните в окне. Чтобы переместить процесс переднего плана на задний план, нажмите на кнопку минимизации в окне, или выберите другой процесс, чтобы перейти на первый план.
Теперь рассмотрим запуск нескольких процессов в окне терминала в Linux, а не в графическом интерфейсе. Так как вы используете окно терминала, нет вкладки для выбора; нет никакого способа, чтобы свести к минимуму процесс. Это, казалось бы, указать, что в окне терминала, можно запустить только один процесс, в то время. Это, однако, к счастью, не так. Идея переднего плана и фона в окне терминала находится под контролем нескольких простых операций. Во-первых, вы можете запустить процесс на переднем плане, просто введя имя команды, определяя, как мы уже видели на протяжении всего учебника. Например, VI <войти> или Grep ... <войти> или сверху <войти> все запустить данную программу на первый план для этого окна терминала. Для того, чтобы начать процесс в фоновом режиме, выполните команду, за которой следует амперсанд (&). Например, если вы чувствуете, что ваша команда Grep может занять много системных ресурсов, и поэтому вы не хотите, чтобы нарушить работу системы, вы можете оформить его как Grep ... & <войти>. И заставлять его работать в фоновом режиме.
Вы также можете заставить процесс переключения с переднего плана на фон или фон на переднем плане. Команды являются fg и BG , соответственно. Однако, если процесс в настоящее время работает окно терминала, вы не можете иметь доступ к командной строке для того, чтобы выдать команду fg или BG. Чтобы получить доступ к командной строке, в то время как процесс запущен, в этом окне введите Control + Z. Это приостановит процесс, выполняется и возвращает управление оболочки к вам.
Например, вы выдаете верхнюю команду (вверху рассматривается в следующем разделе). Это интерактивная программа. Она заполняет окно с дисплеем и обновляет себя, так что вы не имеете доступа к командной строке. Теперь представьте, что вы нажимаете Control + Z. На данный момент, вы увидите:

[1] + Stopped верх

Это говорит вам, что задание 1, сверху, находится в остановленном состоянии. В командной строке, при вводе задания, вы увидите список всех запущенных и приостановленных заданий. Если есть только одна работа, набрав FG , возобновит эту работу на переднем плане и, набрав

BG, возобновит эту работу в фоновом режиме.
Представьте себе, что, когда вы набирали рабочие места , вместо этого вы видели следующее:

> рабочие места
[1] - Остановился сверху
[2] + Stopped VI

Здесь есть две подвесные рабочие места. Если вы наберете FG (или BG), задание выбирает то , с +, что указывает на совсем недавно исполняющую работу. Тем не менее, с двумя числами, вы можете также указать, какое задание перейти на передний план или фон с помощью fg p или BG, где p есть номер задания. Оба эти процесса являются интерактивными, так что это не имеет никакого смысла, чтобы переместить либо на задний план. Тем не менее, вы можете легко возобновить один, приостановить его, возобновить с другой стороны, приостановить его и возобновить первым, если, например, вы редактировали документ , но хотели бы, время от времени, видеть, как система выполняет.
Рассмотрим, в качестве другого примера, что у вас есть два времени интенсивной работы, и работа для редактирования. Вы даете команду - найти, которая будет искать всю файловую систему и процесс Grep , который должен будет искать тысячи файлов. Они могли занимать секунды или минуты. Таким образом, вы также хотите редактировать документ в VI. Вы выпускаете следующие три команды:

> Найти ... управление + g
> BG 1
> ... Управления задать расширенное + g
> BG 2
> VI

Теперь вы редактируете в VI, в то время как найти и запустить в задать расширенное фоне. Шаги здесь выполнить свою задачу, но вы могли бы также более просто ввести:

> Найти ... &
> ... & Задать расширенное
> VI

Fork, Exec, Process
В Linux, мы используем такие термины, как fork и exec. Что они означают? Они используются программами чтобы запустить дочерние процессы, то есть процессы, которые будут отсылать к одной из этих программ, когда необходимо создать дочерний процесс. Разница в том, как команды реагируют.Команда Fork дублирует материнский и порождает новый процесс, как дочерний. Дочерний процесс имеет уникальный PID, материнский PID и свои собственные ресурсы, в том числе и процессорное время. Он получает свою собственную память, копирование материнской памяти происходит изначально. В дочернем процессе есть открытый доступ к материнским файлам. После того, как команда Fork породила дочерний, материнский процесс может либо продолжить выполнение, или войти в состояние ожидания. В последнем случае, материнский ждет, пока дочерний (или дочерние) завершаться до возобновления. Команды Exec похожа на Fork заисключением того, что порожденный дочерний занимает место материнского. В этом случае дочерний берет материнский PID со всех ресурсов. Материнский прекращается, когда дочерний перенимает с двух процессов и не может поделить тотже PID.

Следует отметить, что далее в этой главе, мы рассмотрим еще один способ контролировать процесс просвечивания его PID. Номер задания, как здесь используется другое значение. Вы получаете номера заданий с помощью команды заданий. Вы получаете ИДП через PS или топ-команды (рассматривается в следующем разделе).
В Linux, вы можете управлять 'приоритет процесса' с помощью команды под названием 'хорошо' . Приятно команда относится к ' любезность процесса'. Любезность, в Linux, определяется как, насколько хороший процесс по отношению к другим процессам. Приятнее процесс, тем больше он готов отказаться от контроля над CPU для других процессов. Другими словами, лучше процесс имеет более низкий приоритет, чем менее приятный процесс. Приятное значение в диапазоне от -20 (хотя бы хорошо, или наивысший приоритет) до +19 (самый хороший, или самый низкий приоритет).
Хорошая команда выдается, когда вы запускаете процесс из командной строки, используя следующий формат:

Процесс-имя хорошо -n значение
Этот процесс выполняется имя-устанавливающее свое первоначальное значение славное значение. Если вы не выдаете команду, используя хорошо, это дается хорошее значение по умолчанию +10. После того, как процесс запущен, вы можете изменить его славное значение, используя ту же самую команду.
Например, если мы запускаем некоторый процесс Foo выдачи команды:
хорошо Foo -n 0
мы можем позже настроить Foo, выполнив команду:
хорошо Foo -n 19
Если вы задаете правильную команду без процесса или значения, то будет правильный ответ с правильным значением по умолчанию.Если вы выдать хорошую команду без процесса.

Статуст процесса

Состояние процесса является информация о процессе. Она будет включать в свое состояние ресурсы, предоставленные ему, и его PID. И Windows и Linux позволяют исследовать ваши процессы. Вы можете найти, например, сколько процессорного времени каждый процесс использует, насколько занят ваш процессор и сколько оперативной памяти и виртуальной памяти в данный момент используются. Вы также можете контролировать свой процесс "приоритет, регулируя время ЦП, основное внимание будет уделено процессуальному во время многозадачности. Здесь мы рассмотрим подходы, используемые для зондирования процессора для обработки информации.
В операционной системе Windows, производительность системы и информация о состоянии процесса, все это доступно с помощью диспетчера задач Windows. Типичная точка зрения смотреть на запущенные приложения (в отличие от самих процессов). На рисунке 11.3, мы видим, что существует семь пользовательских приложений, которые в настоящее время работают, но 74 из всего процесса (многие из процессов либо системные процессы, или были запущены в поддержку пользовательских приложений). Использование процессора составляет 0% в данный момент и объем физической памяти составляет 28% (то есть только 28% памяти используется).
На вкладке Приложения, мало что вы можете сделать, кроме как начать новый процесс, убить процесс (End Task), или заставить операционную систему, чтобы перейти от одного процесса к другому. На самом деле, с

помощью "Switch To", вы просто движете процесс от фона на передний план, в то время как текущий приоритетный процесс перемещается на задний план. При выборе New Task ..., появится всплывающее окно. В этом окне вы вводите имя составного спроса , вы хотите выполнить. Команда должна включать в себя полный путь к исполняемому файлу, например "C: \ Program Files \ Internet Explorer \ iexplore.exe". При выполнении процесса, таким образом, процесс выполняется под правами администратора.

Наиболее важное применение вкладке Приложения диспетчера задач убить задачу, которая перестала отвечать на запросы. Windows использует выражение "перестал отвечать", как вежливый способ сказать, что процесс зашел в тупик или иначе умер, но операционная система не может убить его сама. Для того, чтобы убить процесс, выделите его в списке процессов и выберите - Завершить задачу. Если приложение содержит несохраненные данные, вам будет предложено, если вы хотите, чтобы сохранить данные в первую очередь.

Вкладка Процессы обеспечивает гораздо большую информацию, чем вкладка Приложения . Во-первых, все процессы перечислены. Во-

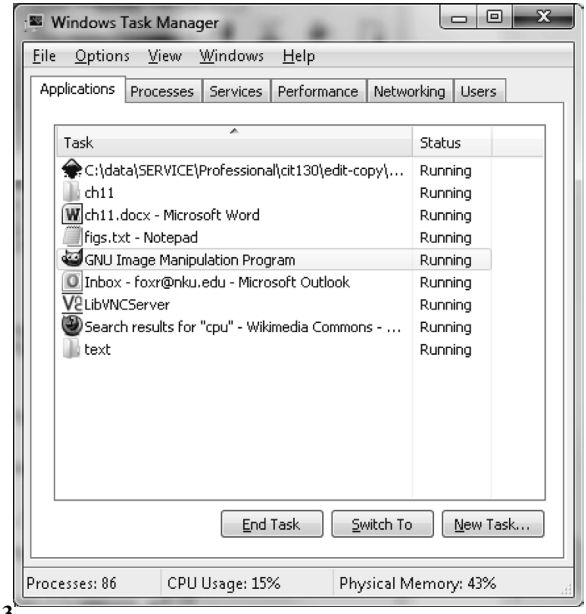
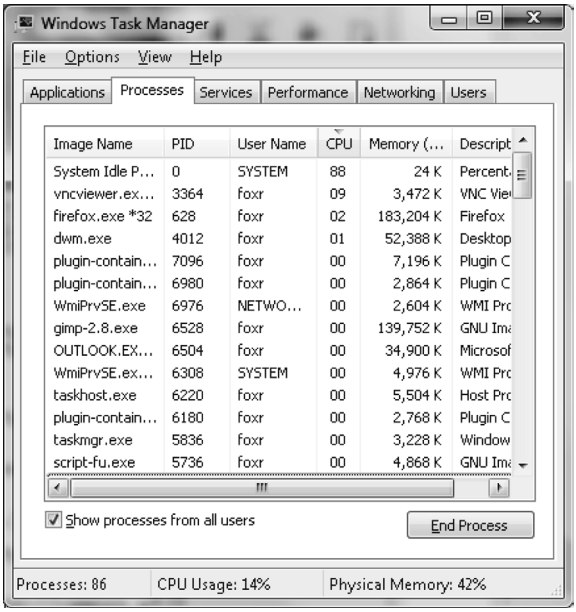


Рисунок 11.3
Менеджер задач Windows

вторых, вы видите каждый процесс "текущий процессор и использование памяти. Вы можете также узнать, кто владеет процессом (который начал его). В операционной системе Windows, вы будете обычно видеть один из четырех пользователей: система, местное обслуживание, обслуживание сети, или собственное пользовательское имя. Первые три показывают, что либо OS начал процесс, или что процесс породил в поддержку другого программного обеспечения (смотрите Рисунок 11.4).

При выборе любого из процессов в закладке Процессы, вы можете щелкнуть правой кнопкой на него и установите процесс "приоритет (повышать или понижать его) и сродство (какие процессоры или ядра разрешено запускать процесс). Приоритеты ограничены одной из шести значений: низкого, ниже нормального, нормального, выше нормального, высокого и реального времени . Значение по умолчанию для процесса является нормальным. В реальном масштабе времени означает, что процесс настолько важен , что он должен выполнить свою задачу, как она представлена, то есть, что оно никогда не должно быть отложено. Вы также можете завершить процесс, а также все процессы, которые он, возможно, породил (конец процесса дерева), отлаживать процесс, или



11. 4

Процессы в диспетчере задач Windows.

вывести на экран процесса "свойства. Окно свойства процесса, пример показан на рисунке 11.5. Здесь вы можете увидеть общую информацию (тип файла, описание, расположение в файловой системе, размер), изменить 'безопасности, получить подробную информацию о процессе, или выяснить процесс "процесса совместимости.

Вкладка Услуги диспетчера задач аналогична вкладке Процессы, за исключением того, что перечислены только услуги. Эти службы перечислены по PID, описание, состояние и группы. Если нажать на любую услугу, вы можете запустить или остановить его. Вы также можете получить список всех системных служб. Мы будем исследовать услуги подробно в службах, Настройка служб и создания служб во время загрузки.

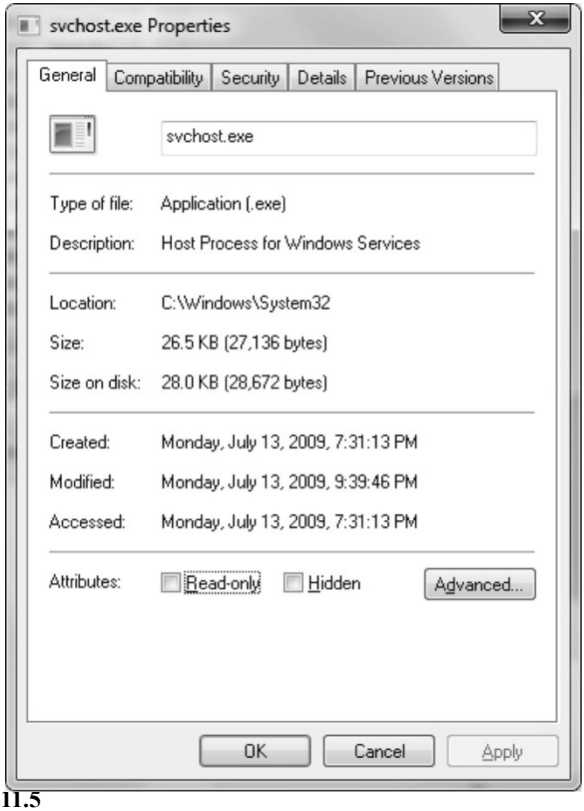


РИСУНОК 11.5
Свойства процесса для svchost.exe.

Следующие две вкладки диспетчера задач, производительности и работа в сети, обеспечивают систему конкретной информацией об использовании ресурсов. Производительность конкретно охватывает использование процессора, основное использование памяти, использование виртуальной памяти, и текущее число потоков, процессов и ручек, работающих под управлением. Ручка представляет собой тип ресурса и может включать в себя открытые файлы, переменные в памяти и операции труб. По существу, ручка является указателем в таблице к фактической сущности, который представляет дескриптор. На вкладке Сеть отображается текущая беспроводная локальная сеть и использование соединений (если таковые имеются).

На вкладке Производительность , есть кнопка, при помощи которой можно открыть функцию ресурсов монитора, которая обеспечивает более подробную информацию о процессоре, памяти, диске и об использовании сети. Рисунок 11.6 отображает как вкладка Производительность и окно «Монитор ресурсов», отображающий общее использование ресурсов (обратите внимание на то, что монитор ресурсов имеет другие вкладки, чтобы изолировать использование только CPU диска или сети). На вкладке «Users» отображаются все пользователи и запущенные процессы. Эта вкладка может показать вам другого пользователя, если кто-то удаленно подключится к вашему компьютеру, и будет управлять любыми процессами.

При использовании вкладки «Monitor», вы можете выбрать определенные процессы. На рисунке 11.6 видно, как firefox.exe (веб-браузер Mozilla Firefox) и WINWORD.EXE (Microsoft Word) выделены, и их использование отображается в памяти. Дополнительная информация представлена в этом окне через вкладку, что показывает объем оперативной памяти,

РИСУНОК 11.6

Вкладка «Производительность» и окно ресурсов монитора.

ТАБЛИЦА 11.1 Linux PS .Список команд и функций

a	Показать все процессы
c	Показать настоящее имя команды
e	Показать результат
f	Показать процессы «parent»/ «child relationship» (ASCII-art как иерархия)
l	Показать определенный пользователем формат
m	Показать все соединения
o	Отобразить неопределенный пользователем формат
p	Выбрать процессы ID
r	Выйти из запущенных процессов
S	Включить информацию «child process» (вместе с информацией «parent»)
t	Выбрать процессы окна (tty)
T	Выбрать процессы терминала
U	Выбрать процессы определенного пользователя
u	Отобразить формат, ориентированный на пользователя
x	Выбрать процессы, что не относятся к процессам терминала (tty)

объем используемой памяти, которую можно использовать совместно с другими процессами, и объем используемой памяти, которая является принадлежащей только этому процессу. Также в этом окне отображается фильтр, который показывает количество неисправностей (неисправности страницы), которые происходят в данную секунду. Так вы сможете выбрать вкладку «Disk» или «Network utilization» для выбранных процессов.

Существует целый ряд различных инструментов для получения информации о производительности в Linux. Более традиционные способы получения информации о процессе через программы PS (состояние процесса), выполненного в командной строке. Команда PS имеет ряд функций, которые могут сбить с толку, поэтому мы будем смотреть на более часто используемые функции.

Использование команды PS само по себе показывает активные процессы в текущем окне, которые используете вы . Этой информации недостаточно для пользователя, поскольку есть много процессов, которые не будут показаны, например, те, которые принадлежат другим пользователям (в том числе корень основы). Таблица 11.1 приводит список наиболее распространенных функций. Обратите внимание на то, что во многих случаях вы не используете - символ при предоставлении опции в команде PS. Это функция от Unix, где – востребован символ.

Пожалуй, наиболее распространенное использование PS – это использование самих команд и команд PS AUX. Версия PS AUX показывает все процессы всех пользователей. Эта версия может отображать слишком много информации. Поэтому можно передать результат в GREP, чтобы получить только те процессы, которые соответствуют заданной строке. Например, PS AUX Grep foxr будет отображать только те процессы, что принадлежат foxr, но также будет показывать все процессы, а не только те, что ограничиваются данным окном терминала. Разновидностью AUX опций является AXF, которая дает процессы в форме дерева, чтобы показать, какие процессы породили

```
  PID TTY          TIME CMD
 5031 pts/1    00:00:00 su
 5034 pts/1    00:00:00 bash
 5261 pts/1    00:00:00 ps
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1   2160   648 ?        Ss   May29   0:01 init [5]

root         2  0.0  0.0      0     0 ?        S<   May29   0:00 [migration/0]
root         3  0.0  0.0      0     0 ?        SN   May29   0:00 [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S<   May29   0:00 [events/0]
root         5  0.0  0.0      0     0 ?        S<   May29   0:00 [khelper]
root         6  0.0  0.0      0     0 ?        S<   May29   0:00 [kthread]
root         9  0.0  0.0      0     0 ?        S<   May29   0:00 [kblockd/0]
root        10  0.0  0.0      0     0 ?        S<   May29   0:00 [kacpid]
root       168  0.0  0.0      0     0 ?        S<   May29   0:00 [cqueue/0]
root       171  0.0  0.0      0     0 ?        S<   May29   0:00 [khubd]
root       173  0.0  0.0      0     0 ?        S<   May29   0:00 [kseriod]
root       237  0.0  0.0      0     0 ?        S    May29   0:00 [khungtaskd]
root       238  0.0  0.0      0     0 ?        S    May29   0:00 [pdflush]
root       239  0.0  0.0      0     0 ?        S    May29   0:00 [pdflush]
root       240  0.0  0.0      0     0 ?        S<   May29   0:00 [kswapd0]
root       241  0.0  0.0      0     0 ?        S<   May29   0:00 [aio/0]
root       459  0.0  0.0      0     0 ?        S<   May29   0:00 [kpsmoused]
root       488  0.0  0.0      0     0 ?        S<   May29   0:00 [mpt_poll_0]
root       489  0.0  0.0      0     0 ?        S<   May29   0:00 [mpt/0]
root       490  0.0  0.0      0     0 ?        S<   May29   0:00 [scsi_eh_0]
root       493  0.0  0.0      0     0 ?        S<   May29   0:00 [ata/0]
root       494  0.0  0.0      0     0 ?        S<   May29   0:00 [ata_aux]
root       501  0.0  0.0      0     0 ?        S<   May29   0:00 [kstriped]
root       510  0.0  0.0      0     0 ?        S<   May29   0:13 [kjournald]
root       541  0.0  0.0      0     0 ?        S<   May29   0:00 [kauditd]
root       574  0.0  0.3   3228  1664 ?        S<s  May29   0:00 /sbin/udevd -d
```

РИСУНОК11.7

Команда PS , версия PS AUX.

другие процессы. Например, с помощью Bush можно ввести Xterm, чтобы открыть окно терминала, и с этого окна терминала выдать другую команду (скажем, для компиляции текущего файла); Таким образом, Bush является родителем XTerm, который является родителем команды компиляции.Еще одна команда для получения информации о состоянии находится вверху. В то время как PS дает снимок того, что происходит, в то время как вы выполните команду, топ в интерактивном режиме, она обновляет себя и отображает самую последнюю информацию, как правило, через каждые 3 секунды. Поскольку эта команда является интерактивной, она остается в окне, и вы не имеете доступа к командной строке во время её выполнения. Команда PS не является интерактивной, она работает, показывая все активные процессы, а затем возвращает пользователя к командной строке. Еще одно различие между PS и top , что в top перечислены только те процессы, которые наиболее активны в данный момент. Есть целый ряд параметров, которые могут позволить вам изменить функции команды top. По большей части, если вы не пытаетесь работать с эффективностью вашей системы, вам работать с PS будет проще, и она будет более полезна в использовании. Для выхода из команды top, нужно нажать Ctrl + C. Рисунок 11.8 демонстрирует вывод из этой команды.

```
top - 05:08:54 up 6 days, 21:01,  3 users,  load average: 0.52, 0.48, 0.45
Tasks: 130 total,   2 running, 128 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.6%us,  3.6%sy,  0.0%ni, 83.4%id,  0.3%wa,  1.7%hi,  4.3%si,  0.0%st
Mem:    514908k total,   472176k used,    42732k free,   107128k buffers
Swap:  1020116k total,       80k used,  1020036k free,   205552k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3473 foxr      15   0 47856 9928 8208 S 10.0  1.9  14:33.72 vino-server
 3086 root      15   0 34288  21m  10m S  0.7  4.4   0:21.85 Xorg
 3624 foxr      15   0 41252  12m 8560 R  0.3  2.5   0:01.84 gnome-terminal
    1 root      15   0  2160  648  560 S  0.0  0.1   0:01.35 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      10  -5     0    0    0 S  0.0  0.0   0:00.44 events/0
    5 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 khelper
    6 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
    9 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
   10 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
  168 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 cqueue/0
  171 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
  173 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kseriod
  237 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khungtaskd
  238 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
  239 root      15   0     0    0    0 S  0.0  0.0   0:00.17 pdflush
  240 root      10  -5     0    0    0 S  0.0  0.0   0:00.14 kswapd0
  241 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 aio/0
  459 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 kpsmoused
  488 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 mpt_poll_0
  489 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 mpt/0
```

РИСУНОК 11.8

Команды вкладки Файловые системы.

Как и Windows, Linux также предоставляет графические инструменты для работы системы. Во-первых, инструмент System Monitor – это тот же Диспетчер Задач в Windows. System Monitor имеет три вкладки: для отображения процессов, использования ресурсов, а также файловую систему. На вкладке Processes, все процессы перечислены вместе с их статусом, загрузка процессора, любезность, UID и использование памяти. Отсюда, вы можете остановить или продолжить (режим сна) процессы, удалить процессы, или изменить их приоритет. Вкладка Resources отображает процессор, память и использование сети, как в текущее, так и в последнее время. Вкладка Файловые Системы отображает текущую установленную файловую систему, аналогичную команде df –k. На рисунке 11.9 показаны процессы и ресурсы вкладок.

В дополнение к Resource Monitor, Вы можете также просмотреть фон и услуги по требованию через графический интерфейс службы настроек. Существует также графический интерфейс System Log Viewer, который также доступен. Хотя это не дает прямую информацию о запущенных процессах, многие процессы обеспечивают информацию о времени выполнения в Log files. Файлы журнала обсуждаются далее в этой главе, наряду с функциями. Как Configuration Manager Service и инструменты системы просмотра журналов требуют, чтобы вы вошли в систему под корень, чтобы использовать их. Это не требуется для System Monitor.
Операционные системы

Операционная система выполняет планирование для пользователя. Есть различные формы планирования. В системах периодического действия,

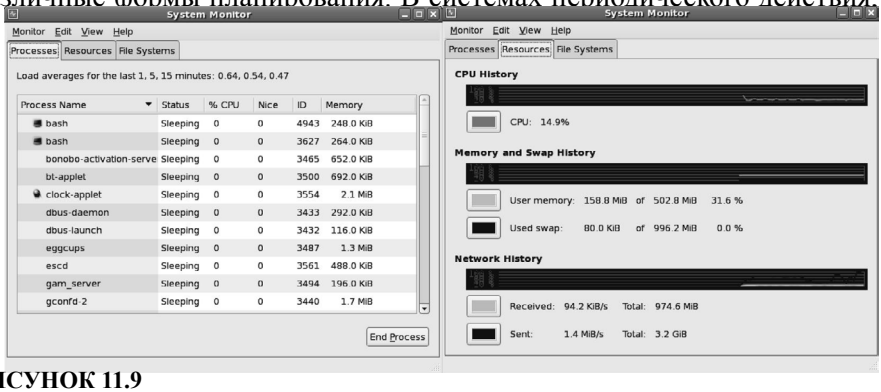


РИСУНОК 11.9
Два вида системного монитора в Linux.

планирование требуется, чтобы определить порядок, как процессор будет выполнять программы. Поскольку пользователи, как правило, не используют Пакетная обработка, эту форму планирования мы можем в значительной степени игнорировать. В многозадачности, циклического планирования является общий подход для выполнения этих процессов. Однако, как уже говорилось выше, пользователи могут изменять поведение циклического планирования путем явного изменения приоритетов процесса и / или переноса процессов на задний план. Еще одна форма планирования, что выполняет ОС, когда процессы по расписанию перемещаются из очереди ожидания на очередь готовности. В персональных компьютерах, это обычно не актуально, поскольку все процессы перемещаются в очередь готовых процессов. Тем не менее, пользователи могут запускать процессы, которые ставят их в очередь ожидания. Здесь мы сосредоточимся на том, как планировать процессы в Windows и Linux, которые указывают, когда процесс должен перейти в состояние готовности.

Windows 7 обеспечивает программу планировщика заданий. Смотрите рисунок 11.10. Вы можете задавать запланированные действия для выполнения. Задачи могут быть запланированы на ежедневной, еженедельной или ежемесячной активности, или разовых явлениях. В любом из этих случаев, необходимо указать начальное время. Можно также указать, что задача должна произойти при следующей загрузке компьютера, когда конкретный пользователь входит в систему, или когда типы события заносятся в журнал. В качестве примера, вы можете задать программе функцию для запуска вашего противовирусного программного обеспечения каждый раз, когда ваш компьютер загружается, или каждую ночь в 3 часа ночи. Вы можете так же написать программу для выполнения полного резервного копирования и планирования этой программы, которая будет выполняться один раз в неделю.

В Linux, существует три планирования программ записи. Все эти три процесса запускаются в фоновом режиме; таким образом, процессы требуют ввода из информации файлов. Команда (s) может быть востребована при надлежащем перенаправлении. Три процесса являются crontab. Команда пакетного файла позволяет планировать процессы, которые будут выполняться после загрузки ЦП ниже 80%. Команда позволяет планировать процессы запуска в определенное время. Эти команды выполняют единовременное действие запланированных

процессов. Команда crontab используется для повторяющихся планирований. Как связаны эти команды, мы обсудим позже, но имейте в виду, что команда batch будет вызывать процессы для запуска, и позволять загружать процессор. Таким образом, процессы будут выполняться только в запланированное время.

Оба файла могут принимать команды. Формат команды batch (пакета) – это почти batch или batch –f в поле имени. Каждый процесс должен быть выполнен через команду Linux, в том числе любые необходимые перенаправления, например, cat foo1.txt foo2.txt foo3.txt | sort >> foo4. txt. Если вы не используете –f функцию, то команда перенаправит вас в строку, что читается как at >. С этой строки вы вводите команду и нажимаете <enter> после каждой команды. Чтобы завершить процесс нужно ввести control+d. Когда всё сделано, на экране появится сообщение:

job 5 at 2012-03-01 12:38

которое определяет номер работы и время, когда была задана команда (не время, когда команда будет работать).

РИСУНОК11.10

Операционная система Windows .

Временной индикатор для команды at can может иметь несколько форматов. В первом способе, вы можете вводить время в формате HH:MM, например, 12:35 t, начать процесс в 12:35. Без определения поры дня (утро или вечер); таким образом, 12:35- это утро, а 00:35 – вечер. Вы можете указать минуты, а можете и не указывать. Там запрограммировано три времени: полночь, полдень и второй полдник(16:00).

В расписании командных процессов для запуска в следующий раз будет достигнуто заданное время. Например, если вы укажете в полдень, а сейчас 1:35 вечера, то процесс будет выполняться на следующий день в полдень. Можно также указать дату в течение 24 часов и после. Сроки обеспечиваются с помощью одного из трех нотаций MMDDYY, MM/DD/YY, или DD.MM.YY, где MM – это месяц в числовом формате (например, 1 или 01 для января и 12 для декабря), DD – это число месяца, и YY- это последние две цифры года. Также вместо даты можно указать день (завтра или послезавтра) и время в этот день.

В альтернативе, чтобы не указывать дату и время, можно использовать сегодняшнее число и добавлять к нему дни. Так же вы можете добавлять минуты и часы, дни и недели, или месяц. Но заметьте, что нельзя добавлять секунды.

Когда вы задали задание, вы должны проверить его при помощи atq. Инструкция atq покажет вам всю нужную информацию об ожидающих заданиях. Команда atrm позволяет удалять задания из очереди планирования. Если есть только одна работа, atrm удаляет его. В противном случае, вы должны указать номер задания, как в atrm 3, чтобы удалить третье плановое задание.

Это обе программы – программы планирования. Служба atd наблюдает за очередью планирования, чтобы выявить процессы, которые должны быть выполнены. Она работает в фоновом режиме, и в каждую минуту сравнивает текущее время и время заданий.

Другой Linux планировщик называется crontab. Эта программа будет использоваться для планирования повторяющихся задач, например, тех, которые происходят каждый понедельник в 1 час ночи или в первый день каждого месяца. В некотором смысле это похоже на первый планировщик, но это сложнее. Во-первых, необходимо создать файл

cron. Файл cron определяет две вещи: время / дату и процесс, который намечен. Время / дата состоит из пяти целых чисел или[1]. Пять значений, в порядке: минута, часы, день, месяц и день недели, чтобы выполнять команду по расписанию. Минута – это число от 0 до 59, час - от 0 до 23 с использованием военного времени, день месяца - от 1 до 31, месяц - от 1 до 12, и день недели - от 0 до 6 (где 0 означает воскресенье, 6 - субботу).

-Первый день каждого месяца, в 3:15 утра
-Каждое воскресенье в 2 часа дня
-Каждые полтора часа (например, 12:30, 1:30, 2:30)
-Каждое 31 декабря, в полночь
-Каждый час каждый день
Если одновременно указать дату (день и месяц) и день недели, например:
0 0 15 * 0
назначит для полуночи каждый 15-й день и каждое воскресенье.

Если вы хотите запланировать несколько других конкретных дат и дней недели, несколько периодов времени, вы можете перечислить несколько записей для каждой категории времени, разделяя записи запятыми. Например,
0,30 0,6,12,18 * * *
Можно планировать задания, что происходят ежедневно в 12:00, 12:30, 6:00, 6:30, 12:00, 12:30, 18:00, 18:30 вечера. Другой вариант, если вы хотите, чтобы всё происходило в течение определенного интервала времени, например, через каждые 15 минут. Это задается с помощью /15, например,
0/15 12 * * *
12:00 для каждого дня, время, которое не повторяется каждые 15 минут до 01:00 ночи.
Можно также указать такие диапазоны как:

1 Вот некоторые примеры с указанием времени (часть файла crontab):

15	3	1	*	*	—the first day at 3:15 am	of every month,
0	14	*	*	0	—every Sunday at 2 pm	
30	*	*	*	*	—every hour at (e.g., 12:30,	half past 1:30, 2:30)
0	0	12	31	*	—every December 31, at midnight	
0	*	*	*	*	—every hour of	every day

0 0 1-5 * *
Будут выполняться каждый день с 1-го по 5-е число каждого месяца в полночь.

Вторая половина файла crontab – это список команд, которые должны выполняться. По мере того, как команды должны помещаться на одной строке в файле, вы должны либо перечислить одну команду, либо написать скрипт. Например, запись:

0 0 1 * * ./somescript –a < foo.txt

Запланируйте выполнение –a < foo.txt на полночь, первого дня каждого месяца. script somescript находится в текущем каталоге и получает опционный и входной сигнал от foo.txt.

Причина, по которой ваши действия - ограничены одной строкой для каждой запланированной задачи в том, что crontab позволяет работать с файлами, которые содержат несколько записей, запланированные по одному в каждой строке. Таким образом, можно запланировать многочисленные процессы в разное время, используя один файл crontab. Рассмотрим следующие записи в файле crontab.

```
0     0     *     *     0     ./backup_script
0     0     *     *     2     ./cleanup_script
0/10  *     *     *     *     ./who_report
```

Это требует, чтобы программа резервного копирования запускалась каждое воскресенье в полночь, что зачистка скрипт будет работать каждый вторник в полночь и что сценарий, сообщения о том, кто вошел в систему, будет работать каждые 10 минут каждого часа каждого дня.

Запись файла crontab является первым шагом в задачах планирования с crontab. Во-вторых, выдача самого процесса crontab. Процесс crontab вызывается с помощью команды CRONTAB -f имя файла. Без этого, ни одна из задач, запланированных в файле, на самом деле, не будет запланирована. Дополнительная опция в crontab является -u – это имя пользователя. Это позволяет запускать процессы, перечисленные под другим именем пользователя; в противном случае, процессы по умолчанию работают под учетной записью пользователя. Это особенно

полезно, если вы перешли в коренной каталог перед выдачей crontab. Как правило, если вы не хотите, чтобы работа crontab проводилась как кореннной каталог, особенно, если вы не знаете всех процессов, необходимых для корневого доступа. Поэтому, в качестве коренного каталога использовать лучше всего crontab -u имя пользователя -f имя файла, где имя пользователя это ваша собственная учетная запись.

Как и в случае atq и atrm, вы можете проверить очередь заданий crontab и удалить их. Это можно сделать с помощью crontab -l и crontab -r , войти в список и удалить ожидающие задания. Вы можете также использовать crontab -e для редактирования самой последней работы crontab. По мере того, как используется atd для запуска и пакетных заданий, cron используется для запуска рабочих мест crontab.

Последнее замечание. Для защиты системы, вы можете указать, какие пользователи могут использовать crontab, и которые не могут. Существует несколько файлов, которые контролируют такой доступ: /etc/at.allow, /etc/at.deny, /etc/cron.allow, и /etc/cron.deny.

Процесс прекращения

Есть несколько способов прекращения процесс. Во-первых, он может закончить выполнение. Во-вторых, вы можете выбрать его остановку самостоятельно. В-третьих, программа может "умереть", то есть, прервать его выполнение. Очевидно, что если процесс завершается сам по себе, то, как только он завершает выполнение, нет необходимости, чтобы пользователь сам принимал решение. В третьем случае, как правило, будет своего рода обратная связь с пользователем, чтобы указать, что возникла ошибка прекращения. В операционной системе Windows, например, вы можете получить всплывающее окно об ошибке. Или выход, то есть окно программы может просто исчезнуть с экрана. В Linux, программы, которые неправильно заканчиваются, обычно оставляют файл core dump. Файл core dump представляет собой "свалку", которая представляет собой снимок процесса «рабочей памяти». Программист может изучить файл core dump для подсказок при решении проблемы.
Это вторая возможность для решения. Процесс остается «активным», но больше не выполняет никакого прогресса на пути к завершению. Это может быть вызвано тупиковой ситуацией (смотрите главу 4), или

дочерними процессами, которые перестали отвечать на запросы, или ситуацией, когда сама ОС потеряла указатели для этого процесса. В любом случае, пользователь должен обнаружить, что этот процесс не отвечает, и решить, что с этим делать.
В операционной системе Windows, чтобы остановить процесс, вы можете просто закрыть окно. Для процессов, которые не реагируют, это может не сработать, и поэтому вы вызываете диспетчер задач и убираете приложение или процесс с помощью этого графического пользовательского интерфейса. Если процесс запущен в окне терминала (Linux) или DOS окно командной строки (Windows), вы можете попробовать Ctrl + C, чтобы остановить его. В Linux, вы можете также использовать монитор ресурсов, чтобы выбрать процесс. Вы можете это также использовать, чтобы удалить команду из командной строки. Во-первых, вы должны получить процесс 'PID с помощью команды ps. Команда удалить - это уровень PID. Самый высокий уровень -9 должен использоваться всякий раз, когда это возможно. Чтобы удалить процесс, вы должны быть владельцем процесса или находиться в корневом каталоге. Команда killall всё может удалить все активные процессы. Чтобы использовать killall таким образом, вы должны быть в корневом каталоге.

Возможно, вам придется искать информацию через запущенные процессы, чтобы увидеть, какие процессы все еще работают, и закрыть процессы, которые вам больше не нужны. Это правильная тактика как для Windows, так и для Linux. Просто потому, что значок не привязан к панели задач, это не означает, что процесс не может быть запущен до сих пор. Часто в Linux процессы трудно отслеживать при запуске из командной строки. Интересная историческая справка, когда MIT закрыли свой IBM 7094 мэйнфреймов в 1973 году, они обнаружили процесс, ожидания с низким приоритетом, который был запущен в 1967 году! Неизвестно, является ли эта история правдивой.
Выключение всей системы является важным шагом. Вам нужно будет выключить компьютер после установки определенного программного обеспечения (особенно исправлений операционной системы). Кроме того, Windows 7 требует, чтобы ее закрыли и перезапустили, когда она выполняет обновление операционной системы. Linux может выполнять модернизацию своей ОС, без перезапуска.

Выключение системы также помогает очистить вашу виртуальную

память, если вы оставите систему работать слишком долго, то возможно, что виртуальная память станет фрагментированной и причиной низкой производительности. В операционной системе Windows, важно перезагрузить ОС иногда (скажем, по крайней мере, один раз в неделю) из-за коррупции, которая может произойти с системой, которые хранятся в виртуальной памяти и оперативной памяти. Поскольку Linux является более надёжней, чем Windows, то ей реже требуется перезагрузка.

Как вы выключаете компьютер? Ну, что вы никогда не хотите сделать это, просто нажав кнопку выкл. И Windows и Linux имеют последовательность отключения. Важно следовать этой последовательности, чтобы все процессы и услуги могли быть закрыты надлежащим образом, и все файлы также могли быть закрыты. Если вы не сделаете это правильно, файлы данных и системные файлы могут быть повреждены, что приведет к проблемам в вашей ОС с течением времени.

В операционной системе Windows, процесс завершения работы берет начало с кнопки Пуск. Варианты выключения системы: перезагрузка системы (перезагрузка), выход из системы, как текущего пользователя, режим ожидания или спящий режим. Спящий режим является режимом энергосбережения, что позволяет начать резервное копирование быстро. Hibernate является более длительным режимом. Процедура завершения работы проста в использовании, хотя процесс завершения работы будет осуществляться очень сложно с точки зрения шагов, участвующих в завершении работы, таких как ОС и аппаратное обеспечение. Выключение компьютера также может занять много времени.

В Linux, вы можете использовать графический интерфейс, или вы можете выключить систему через командную строку с помощью команды завершения работы. Из командной строки, завершение работы принимает параметр, -H, который заставляет систему остановиться. Система фактически останавливает всю работу компьютера, в то время как завершение работы просто помещает вас в специальный режим, который позволяет вам остановить систему. В дополнение к -H, необходимо указать время (в минутах), чтобы обозначить, когда должно произойти завершение работы. Это дает вам "льготный период", так что все пользователи могут выйти из системы по времени. Завершение работы команды -h 10 означает, что для завершения работы и остановки работы системы требуется 10 минут. Вы также можете отправить сообщение для пользователей и процессов, чтобы предупредить их о предстоящем завершении работы. Сообщение может быть что-то вроде "предупреждение, выключения системы в течение 10 минут - Закрыть

все процессы и выйти!" Опция -r не только отключает систему, но и перезагружает её потом.

Сервисы

Служба представляет собой часть программного обеспечения ОС, которая обеспечивает определенный тип функции. Это делает звук и любой другой компонент операционной системы. Есть несколько отличий. Во-первых, служба может быть запущена или остановлена (или приостановлена), тогда как программы, которые составляют ядро операционной системы, продолжают работать все время. Во-вторых, служба настроена на работу с любым количеством других агентов-приложений программного обеспечения, пользователей, сетевых коммуникаций и так далее. Таким образом, может рассматриваться как программное обеспечение, которое может быть призвано неизвестными клиентами. Ядро ОС обрабатывает запросы пользователя под управлением программного обеспечения, но не от неизвестных или удаленных клиентов. В-третьих, службы работают в фоновом режиме, без вмешательства пользователя. Поскольку службы работают в фоновом режиме, услуги не являются необходимыми для систем без многозадачности. Например, рассмотрим одну систему задач, где ОС является исполняющим или прикладным программным обеспечением. Там нет фона. Но в многозадачной системе, эта услуга может быть предложена время от времени.
Услуги могут быть запущены во время загрузки, они могут быть запущены на основе некоторой системы планирования, или они могут быть запущены (и остановлены) системными администраторами. В Linux, услуги часто называют *daemons* (произносится как демонс), и их имена часто заканчиваются на d, например, команда планирования находится под контролем *daemon*, известного как ATD. В операционной системе Windows, они известны как Windows Services, но в отличие от Linux, некоторые службы Windows могут работать как нормальный процесс, то есть выполнять задания переднего плана.

Одним из преимуществ, имеющих услуги в ОС является то, что системный администратор может адаптировать среду, контролируя работу сервисов и услуг, которые не работают. Например, может возникнуть необходимость в почтовой связи, если мы ожидаем, что этот компьютер получит сообщение по электронной почте.
Тем не менее, если это не так, то услуга сообщение электронной почты

может быть остановлена так, что не будет занимать место в памяти и требовать внимания от процессора.
Кроме того, системный администратор может контролировать то, как работает служба. При запуске службы, служба, скорее всего, прочитает файл конфигурации. Путем изменения файла конфигурации, вы можете контролировать то, как работает эта служба. Мы рассмотрим некоторые из файлов конфигурации Linux в Настройках служб.
Еще одно преимущество наличия услуг является то, что службы, если они выполняются, то помещаются в фоновом режиме. Таким образом, служба только требует системные ресурсы, когда она призвана выполнять свою службу. Так, например, услуга не ждет очереди и использует процессорное время. Только тогда, когда служба должна отвечать она будет использовать процессор.
В операционной системе Windows, службы имеют несколько примечательных особенностей. Во-первых, они работают в менее привилегированном режиме, чем режим администратора. Например, они могут работать по счетам локальной службы или сетевой службы. Это обеспечивает сервис с дополнительными правами доступа, чем доступ обычного пользователя, но не полный доступ администратора. Кроме того, в Windows, службы разобщены, так что они не могут влиять на другие службы. Служба, которая была скомпрометирована вирусом или другой формой вторжения, не повлияет на другие службы. И, наконец, услугам не разрешается выполнять операции в файловых системах, системном реестре или компьютерной сети.
В Linux большинство услуг порождаются в процессе инициализации, который является первым процессом после того, как Linux запущен (смотрите раздел Создание служб во время загрузки). После того как система запущена, системный администратор может запустить (и остановить) службы по требованию в любое время, так что эти службы не порождены инициализацией. Службы терминалов отделяются так, что они работают в фоновом режиме, не в оболочке, но и не во всей операционной системе. Они устанавливаются как имеющие корневой каталог в качестве своего рабочего каталога, и, таким образом, это не влияет на какие-либо файловые системы.
Службу не следует путать с сервером, хотя они связаны между собой. Мы могли бы сказать, что все серверы предоставляют услугу, но не все службы являются серверами. Один из способов думать о различии в том, что сервер является крупномасштабной частью программного обеспечения (возможно, множество программ). Сервер может иметь

несколько ролей или выполнять многие задачи в рамках услуг, что он предоставляет. Мы также склонны думать, что сервер (программное обеспечение) работает на компьютере, в отведенных службах для выполнения службы. Мы обсудим серверы более подробно в главе 13.

Есть большое количество услуг, доступных как в Windows, и Linux. В операционной системе Windows можно просматривать активные службы, запускать и останавливать их, с помощью диспетчера задач. Рисунок 11.11 показывает вкладку Службы в диспетчере задач. Выбор сервиса позволяет затем запускать или остановить его.

На рисунке 11.11, например, вы можете увидеть такие службы, как Net-logon, KeyIso и PlugPlay, работают, в то время как службы, такие как VaultSvc, EPS и bthserv остановлены. В столбце Описание приводится краткое описание службы, а также для тех, кто работает, ПИД-регулятор указывает, какой процесс начался. На рисунке 11.11, например, несколько процессов были начаты в процессе 564. Это, скорее всего, программа lsass.exe. Процесс 708, который начал Power Services и PlugPlay

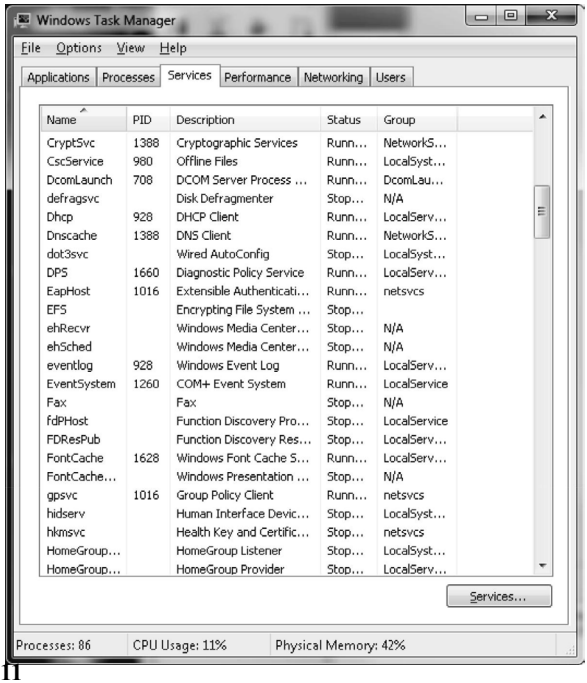


РИСУНОК 11.11
Вкладка Службы диспетчера задач.

является svchost.exe. Для получения процесса ИДП, нажмите на вкладке процесса, и под видом, выберите Выбрать столбцы. Отсюда вы можете выбрать, какие столбцы должны быть доступны для просмотра. PID недоступен по умолчанию, но вы можете сделать его видимым. Обратите внимание на то, что службы, которые останавливаются, не имеют PID в списке.В нижней части вкладки Службы оконного менеджера задач находится кнопка Services. Нажатие на эту кнопку запускает окно служб (смотрите рисунок 11.12). В услугах окошка, вы можете просмотреть информацию об услуге, такие как его описание, и запускается ли она автоматически или требует ручного запуска. Можно также запустить окно служб, щелкнув правой кнопкой по значку компьютера на рабочем столе, и выбрав Управление, а также запуск панели управления, выбрав Администрирование, а из списка инструментов, выбранных услуг. Из окна служб, щелкнув правой кнопкой мыши на службе, и выбрав пункт Свойства (или выделяющие Свойства), отображает свойствами службы. Смотрите рисунок 11.13. Есть четыре вкладки в этом окне: Общие, войдите в систему, восстановления, Зависимость.

На вкладке Общие предоставляется имя сервиса, описание, его расположение в файловой системе, как он запускается (в случае DHCP-клиента, он запускается автоматически во время загрузки), его текущее состояние (если запущен, вы можете остановить его и наоборот), и какие параметры поставляются при запуске. DHCP-клиент

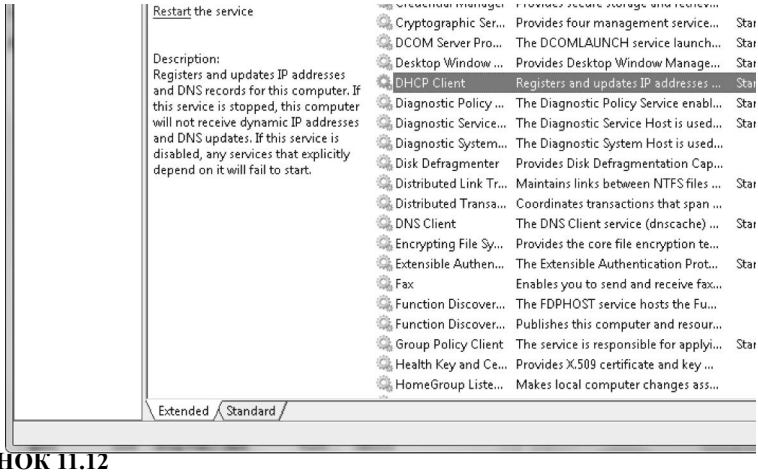


РИСУНОК 11.12
Служба Windows.

не запускается с любыми параметрами.
Под типом запуска, есть четыре варианта: Автоматически (отложенный запуск), автоматический, ручной, с ограниченными физическими возможностями. Они самоочевидны для первого, который был введен в Windows Vista. Поскольку не все автоматические службы необходимы сразу же после загрузки системы, службы могут быть классифицированы по степени их важности. С более низким приоритетом службы могут запускаться автоматически при загрузке, но быть отложены, пока не начались другие, более важные службы. То обозначается "Отложенный старт" пока не начала до инициализации скриптов, процессов и услуг перспективе. Намерение – обеспечить более быстрый процесс загрузки для пользователя. После загрузки, так как система имеет время, остальные автоматические службы запущены. В качестве примера, поиска Windows и обновления Windows службы используют этот параметр, так как их непосредственное обслуживание не потребуется.
Службам может потребоваться доступ к системным или сетевым ресурсам. Если это так, то требуется логический вход (имя учетной

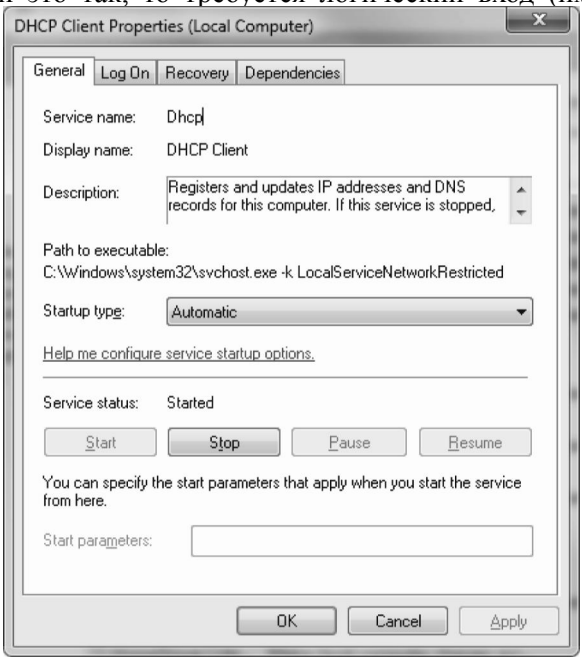


РИСУНОК 11.13
DHCP Возможности окна.

записи и пароль). Бревно в типе указывается под вкладку Вход в систему (вы также можете увидеть это в окне служб как крайнем правом столбце, Log On As). Например, DHCP войдет в систему под именем локальной службы. Другие варианты Network Service и Local System. Локальная служба и Сетевая служба, скорее всего, потребуют журнал в учетной записи и пароль, в то время как локальная служба нет.

На вкладке Восстановление указывается на то, как ОС должна попытаться восстановиться, если услуга не удается. В операционной системе Windows существует три уровня выхода из строя: первый сбой, второй сбой и последующие неудачи. Вкладка содержит следующие элементы для каждого уровня, выбор "не предпринимать никаких действий", "перезапустить службу", "запустить программу" (в этом случае вы указываете программу и любые параметры программы), и "перезагрузить компьютер". Неудачи отсчитываются с момента последнего сброса. По умолчанию, сброс происходит каждый день. Задержка перед перезапуском службы может произойти также управления- Lable. Это значение задается в минутах (от 0 вверх).

На вкладке Зависимости перечислены службы, от которых эта услуга зависит, а службы, которые зависят от данной службы. Это указывает на те службы, которые должны быть запущены для этой службы, чтобы успешно работать, и если эта служба остановила работу, и другие службы будут затронуты. Например, служба DHCP-клиент зависит от вспомогательной функции драйвера для Winsock, Driver NetIO. Наследие TDI Поддержка и магазин сети интерфейса Ser- пороками, в то время как веб-прокси WinHTTP Auto-Discovery Service зависит от DHCP-клиента.

В Linux, инструмент конфигурации службы (смотрите Рисунок 11.14) позволяет просматривать, запускать и останавливать службы. Этот инструмент представляет собой графическое средство взаимодействия с услугами, которые могут также контролироваться с помощью /etc/rc.d каталогов и инструкций командной строки. Автоматический запуск службы подробно объясняется в создании служб во время загрузки.

При выборе службы в этом инструменте, вам дается описание этого сервиса и любые службы, которые могут использовать службу (то есть зависимостей). Например, системный журнал службы требует, чтобы и Syslogd и Klogd были запущены , как показано на рисунке. Вы также дали "уровень запуска" для службы. Это описано также в создании служб во время загрузки . Обратите внимание на то, что этот инструмент предлагает две вкладки, фоновые службы и по требованию служб. Большинство служб Linux предназначены для работы в фоновом режиме.

Тем не менее, вы можете также прекратить действие по запросу службы. Эти службы будут работать при запросе и автоматически закончатся (выход из системы).

В Linux, вы можете также просмотреть состояние любого данного сервиса, запуск, остановку, или перезапустить его из командной строки. Есть два способа управления услугами из командной строки. Общий путь через команды обслуживания, которая находится под / SBIN. Формат команды / SBIN / служба команды имя_службы, где имя_службы это имя службы, например, системный журнал, а команда является одним из запуска, остановки, перезагрузки и статуса.

Команда статуса только отвечает на статус сервиса: выполнение программы, остановка, отказ. Другое свойство командной строки для управления сервисом - это выдача команды непосредственно к самому сервису. Сервис sshd, например, хранится в /etc/init.d. Таким образом, вы можете также выдать команду /etc/init.d/sshd на запуск (перезагрузку, остановку или статус).
Существует большое количество сервисов, как в Windows, так и в Linux, их права также существенно отличаются. Простой категоризации всех сервисов не существует. Общий перечень типов сервисов с некоторыми

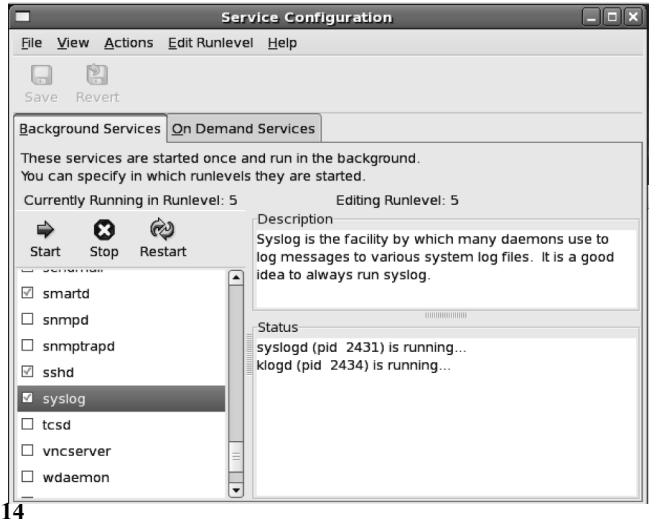


Рисунок 11.14
Linux Services в окне конфигурации сервиса.

программами, которые обслуживают эти категории сервисов, указан ниже.

Служба регистрации. Файлы журналов регистрации играют существенную роль в определении причины необычного или ошибочного события. И Windows и Linux имеют сервисы, которые регулярно регистрируют события, генерируемые ОС, прикладным программным обеспечениям или из какого-либо другого источника. Журнал событий Windows, хранит и осуществляет выбор информации о событиях. В Windows применяются шесть уровней событий: Отказ аудита, успех аудита, информация, предупреждение, ошибка, и критические значения. Программа просмотра событий позволяет просмотреть события, записанные в журнал событий Windows. Записи журнала могут появляться из приложений, проблемных событий оборудования, событий из Internet Explorer, событий из Media Center, из проблем безопасности, системных событий и другие. На рисунке 11.15 указан вид Регистратора событий (Logger events), с перечнем перечисленных предупреждений, а на рисунке 11.16 показаны некоторые из регистрируемых событий из приложений. В системе Linux, демоны Syslogd и Klogd совместно используют ведение учета рутинных событий, система Syslogd служит для регистрации событий нормальной системы и событий приложения, в то время как демон Klogd служит для конкретных событий, генерируемых ядром. В

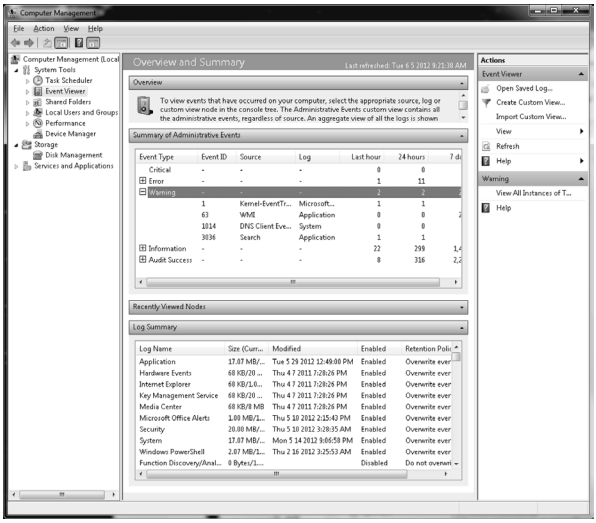


Рисунок 11.15
Просмотр событий Windows.

Службе auditd ОС Linux собираются сведения о событиях, связанных с безопасностью и записываются в audit.log файлах. Модуль ядра Linux - netconsole - регистрирует попытки входа в систему. Другое программное обеспечение может создавать свои собственные файлы журналов, или же они могут вызвать протокол syslogd. Различные файлы журналов в Linux хранятся в каталоге / var / linux, если не указано иное.
Сервисы для планирования. Пользователи, в частности, системные администраторы, обычно планируют процессы для выполнения в определенные моменты времени. Например, противовирусная программа может планировать работу каждую ночь в полночь, или утилита резервного копирования может быть запущена один раз в неделю в определенное время. Планировщик для Windows называется Task Scheduler. Он является как планировщиком, так и сервисом. Linux имеет несколько программ планирования: at, batch, crontab. Сервисы Linux, которые поддерживают планирование, являются планировщиком - anacron для выполнения процессов, запланированных на время самозагрузки, crond выполняет процессы запланированные программами crontab, а atd выполняет процессы запланированные программами at и batch.

ПАКЕТЫ ОБНОВЛЕНИЯ

Пакеты обновления особенно не относятся к сервисам, но из-за того, что здесь рассматриваются сервисы, полезно упомянуть пакеты обновления. Пакет обновления применяется в Microsoft для выпуска обновлений операционной системы. Это позволяет устранить ошибки, обработать жалобы пользователя и убрать слабое место в системе защиты между основными версиями ОС.
Начиная с ОС NT, Microsoft выпускает пакеты обновления для своей

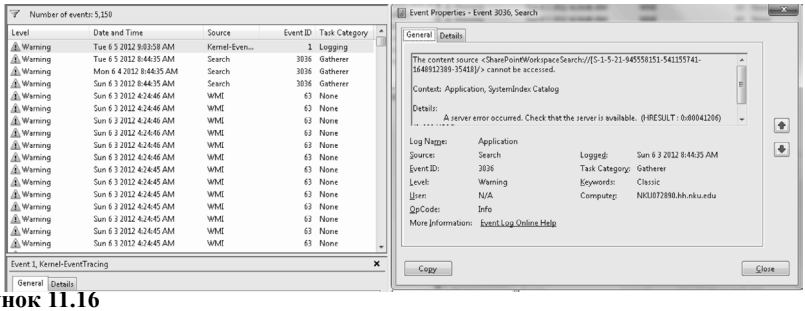


Рисунок 11.16
Зарегистрированное приложение с предупреждением и конкретным типом события.

ОС Windows. В большинстве случаев выпускаемые пакеты обновления заменяют предыдущие. Например, чтобы установить пакет обновления 2, не нужно устанавливать пакет обновления 1. Однако, этот способ не совсем подходит. Пакет обновления 3 XP зависит от пакета обновления 1, который был предварительно установлен.
Применяемые пакеты обновления поставляются на CD-ROM, но сегодня их можно скачать из интернета. При установленной системе Windows 7 пакеты обновления можно автоматически загружать и устанавливать. Однажды вы можете проснуться и увидеть, что ваша ОС работает немного по-другому - новые функции, более строгие меры защиты или странная ошибка больше не появляется. Пакеты обновления не ограничены в ОС Windows. Другие части программного обеспечения сейчас применяют этот метод.

Сетевые сервисы.

Есть целый ряд различных задач, связанных с коммуникацией по компьютерной сети, поэтому существуют многочисленные сетевые услуги, относящиеся к сети. В Linux, существует услуга сети, которая настраивает все сетевые интерфейсы во время загрузки. Но также существует ISDN (Integrated Services Digital Network) для связи по телефонной сети, nfsd (Network File System Daemon), nmbd (Network Message Block Daemon), ntbd (Network Time Protocol Daemon), snmptd (Simple Network Management Protokol Daemon) , а также сервисы для веб-сервера (httpd), ftp-сервер (ftpd), ssh- сервер (sshd), почтовый сервер (Sendmail, Fetchmail), и сервер доменных имен (с именем). В операционной системе Windows существуют NTDS (Network Authentication Server), служба BITS (Background Intelligent Transfer Service), DNSCache и DNS (для DNS-серверов и клиентов), а также NSIS (Network Store Interface Service) для сбора информации о маршрутизации для активных сетевых интерфейсов, если назвать только некоторые. В ОС Windows, рабочая станция создает и поддерживает клиентские сетевые подключения к удаленным серверам. Другие сетевые службы в ОС Windows включают Network Access Protection Agent, а также Network Connections и Network Locations Awareness.
Брандмауэр и Интернет-услуги. В системе Linux приложение dnsmasq запускает кэширующий сервер DNS, а httpd используется для запуска и остановки веб-сервера Apache по умолчанию. Эти утилиты iptables и ip6tables управляют брандмауэром системы Linux по умолчанию. Сервис

nscd системы Linux обрабатывает пароль и группирует попытки поиска для работающей программы, результатов кэширования для будущих запросов. Этот сервис необходим для поддержки других сетевых услуг на основе протоколов аутентификации как NIS и LDAP. В ОС Windows служба Base Filtering Engine управляет безопасностью брандмауэра Windows и Internet Protocol. Сетевой протокол DHCP-клиент регистрирует и обновляет IP-адреса и DNS-записи для компьютера, а модуль DNS-клиент кэширует имена DNS. Если эта служба остановлена, идентификация имен продолжается через DNS, но результаты уже не кэшируются для повышения эффективности, а также другие службы, которые явно зависят от DNS-клиент, покажут сбой. Подобно управлению кибербезопасности ncsd, служба Netlogon системы Windows поддерживает безопасный канал между компьютером и контроллером домена для аутентификации пользователей и услуг.

Файловая система. Как и в случае сетевых услуг, существует множество различных задач, участвующих в файле операционной системы. В Linux, они включают два демона - autofs и amd, которые используются для системных разделов при автоматической установке файловой системы; протокол сетевого доступа netfs, который монтирует и демонтирует любой сетевой файл системы; и файл mdmonitor, который управляет программной частью устройства хранения дискового массива RAID. В ОС Windows, служба архивации Block level backup Engine обеспечивает резервное копирование и восстановление поддержки в файловой системе.

Периферийные устройства. ОС Windows имеет ряд сервисов для поддержки периферийных устройств.

К ним относятся Audio Service для управления конфигурацией аудио разъемов, служба поддержки Bluetooth, и технология - подключи и играй-, который позволяет компьютеру быстро распознавать вновь подключенные к нему устройства, практически без ввода данных пользователем. В Linux, CUPS (общая система печати Unix) поддерживает связь с принтерами. Существует целый ряд услуг, которые все поддерживают управление Bluetooth , включая Bluetooth, hidd, pand и dund. Сервис аферист осуществляет управление с клавиатуры. К сервису gpm добавлена поддержка мыши для текстовых приложений, основанных Linux, например, путем разрешения копирования и вставки действия с помощью мыши.

Разное. Сервис oddjobd в системе Linux интересен тем, что он поддерживает приложения, которые не имеют своих собственных привилегий. Например, если программа запрашивает некоторые

операции, которые должны выполняться, но не имеют соответствующей привилегии для выполнения операции, то программа может запросить сервис oddjobd. В операционной системе Windows существует целый ряд диагностических услуг. Есть также услуги для поддержки, как программа родительский контроль. Обе операционные системы предоставляют услуги управления питанием: AMPD в Linux и Power в Windows. Например, сервис AMPD отслеживает состояние батареи и может применяться автоматически, выключить компьютер, если разряд батареи слишком мал.

Сервисы настройки конфигурации

При запуске сервиса, он обычно читается как файл конфигурации. Файл конфигурации определяет свойства, с помощью которых будет выполняться сервис. Например, сервис брандмауэра Linux, будет работать с использованием некоторых правил. Эти правила задаются в файле конфигурации. Как правило, сервисы Windows используют файлы с расширением.ini (инициализация) для хранения этих технических условий, хотя в некоторых случаях они заканчиваются расширением .cfg или имеют другие, иногда без расширений. В системе Linux файлы конфигурации часто заканчиваются расширением .conf, хотя и не всегда так. Кроме того, подобно скриптам в системе Linux, использование # для начала строки в файле конфигурации указывает на то, что строка является комментарием и должны быть проигнорированы программой, которая считывает файл.
Для того, чтобы изменить поведение сервиса, необходимо остановить сервис, отредактировать файл конфигурации, а затем перезапустить сервис. Следует делать это внимательно, так как ошибки в файле конфигурации могут явиться причиной неисправности или отказа. Здесь мы обращаем внимание на несколько файлов Linux.conf, исследуя их синтаксис и некоторые из изменений, которые вы могли бы сделать. В системе Linux большинство файлов конфигурации можно найти в директории / и так далее.

Мы начнем с одной из услуг - регистрация системы Linux - Syslogd. Этот сервис регистрирует события, генерируемые программным обеспечением приложения и не из центральной части ОС. Для конфигурации Syslogd необходимо знать следующее:

1. Какие виды деятельности должны регистрироваться
2. Какие условия требуют ведения журнала
3. Где должна произойти регистрация.

Файл /etc/syslog.conf описывает эту информацию. Так как это просто текстовый файл, системный администратор может свободно редактировать его, который, в свою очередь, влияет на то, что регистрируется и где. Каждая строка этого файла описывает одну форму ведения журнала событий. Событие протоколирования будет включать в себя один или несколько источников (программное обеспечение), уровень приоритета, который является причиной регистрации события, и действие, которое должно произойти, чтобы источник создал сообщение на данном уровне приоритета. Несколько источников могут перечисляться в одной строке, если источники включены в один и тот же файл журнала.
Форматом для любой отдельной записи является действие source.priority [; source.priority] *. То есть каждая запись будет, по крайней мере, одно действие source.priority, но может быть любое количество пар source.priority до тех пор, пока они не отделены друг от друга точкой с запятой. Использование группового символа (*) в синтаксисе означает "0 или более копий", хотя мы увидим, что можно использовать * в качестве источника, приоритета и действия. При использовании для любого источника или приоритета, * означает "любой источник" или "какой-либо уровень приоритета". Например, можно было бы использовать auth. *, чтобы указать любое сообщение авторизации независимо от уровня приоритета. Или, чтобы указать любое сообщение, чей уровень приоритета является аварийным, можно указать * .emerg. Запись *. * будет регистрировать любое событие с помощью любого программного обеспечения.
Часть действия в записи это почти всегда имя файла, указывающее, где сообщение должно быть зарегистрировано. В большинстве случаев, журналы сохраняются в / вар / директории журнала. Однако, если журнал сообщений поддерживает определенную часть программного обеспечения, то возможно, что различие местонахождения может быть желательным, например, лог-подкаталог в личном каталоге программного обеспечения. Помимо указания местоположения файла журнала, символ * может быть использован вместо отправки сообщения всем зарегистрированным пользователям в открытом окне терминала. Можно также управлять терминалом для вывода с помощью / dev / tty1, например, только для вывода терминала tty1s, вместо любого открытого

окна. Другое альтернативное действие – это вертикальная черта в сообщении регистрации в другой части программного обеспечения. Такая запись может выглядеть следующим образом: | Exec / USR /local / bin / filter.

Существует несколько источников по умолчанию. К ним относятся auth (сервис аутентификации пользователя), cron (планировщик cron-job), daemon (все стандартные услуги), kern (ядро системы Linux), lpr (сервер принтера), mail (почтовый сервер), syslog (сервис sys-logd), и пользовательские (программы запущенные пользователем). Пользовательские программы обозначаются как local0-local7, что позволяет работать в течение восьми дополнительных программ, которые могут быть настроены на ведение журнала. Существует девять уровней приоритетов, которые варьируются от создания сообщения в журнале регистрации до полного отказа системы, чтобы это было просто, как генерация сообщения в журнале для каждого вида деятельности данной части программного обеспечения. Девять уровней приведены в таблице 11.2.

Таблица 11.2 Девять уровней приоритета, используемые syslog.conf

None	Ни одного приоритета нет
Debug	Отладочные сообщения, используемые программистами во время тестирования своих программ.
Инфо	Информационные сообщения о том, что выполняет программа.
Внимание	Нет событий, заслуживающих внимания
Предупреждение	Предупреждения о возможных проблемах
Ошибка	Ошибки, которые возникают во время выполнения.
Крит	Сообщения, описывающие критические ошибки, которые, скорее всего, являются результатом аварийного завершения программы.
Предупреждение	Сообщения, в которых будут описаны ошибки, не только приводящие к аварийному завершению программы, но которые могут также влиять на другие программы.
Emerg	Авар сообщения об ошибках, которые могут привести к сбою системы.

На рисунке 11.17 указано содержимое файла syslog.conf. Первая запись указывает, что любое сообщение ядра должно быть отправлено в / Dev / консоль, которая является консолью главного компьютера (это сообщение не будет отправлено на консоли дистанционного входа в систему или другие окна). Вторая запись определяет многочисленные сообщения: все информационные сообщения и всю почту, сообщения authpriv и cron без какого-либо приоритета. Они все вошли в / Var / Журнал / сообщений. Третья запись запрашивает, чтобы все остальные сообщения authpriv (выше уровня приоритета - "нет приоритета") записывались в / Var / Журнал / безопасности. Можете ли вы понять смысл остальных записей? Использование "-" в действии перед именем файла, например, - / Var / Журнал / MAILLOG на рисунке указывает на то, что сообщения не должны быть синхронизированы.

Брандмауэр Linux требует наличия файла конфигурации. Этот файл хранится в / etc / sysconfig / ipt. Модификация этого файла позволяет добавлять, изменять и удалять правила брандмауэра. Каждая строка этого файла представляет собой тип сообщения, которое должно быть принято или отклонено программным обеспечением брандмауэра.

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                                  -/var/log/maillog


# Log cron stuff
cron.*                                                  /var/log/cron

# Everybody gets emergency messages
*.emerg                                                 *

# Save news errors of level crit and higher in a special file.
uucp,news.crit                                          /var/log/spooler

# Save boot messages also to boot.log
local7.*                                                /var/log/boot.log
```

Рис.11.17

Вход регистрации из /etc/syslog.conf.

Правила имеют особый синтаксис. Они начинаются с типом правила, правило добавления в конец (-A) или флеш правило (-F). Флеш правило удаляет все правила в выбранной таблице, так что большинство или все правила будут начинаться с -A.
Следующий пункт в правиле, это имя потока, из которого сообщение может прийти. Потоки имеют общие названия, такие как INPUT, OUTPUT, или FORWARD, или может быть из определенного места, например, fast – input-queue или icmp-queue - out. Приведенные ниже примеры показывают, input stream RH-Firewall-1-INPUT.
Далее следуют конкретные условия, которым правила соответствуют. Они могут включать в себя порты, типы сообщений TCP / IP, источник или IP-адрес назначения, или протоколы. Например, --dport 50 указывает сообщение через порт 50, в то время как -p udp --dport 5353 указывает сообщение через udp-порт назначения 5353, а -d 1.2.3.4 указывает на IP-адрес назначения 1.2.3.4.
Последняя запись в правиле - это обычно действие, по которому правило должно применяться, если это правило соответствует сообщению. Самые основные действия ACCEPT, LOG, DROP и REJECT. ACCEPT немедленно прекращают обработку сообщения и передают его на ОС (то есть мимо брандмауэра). Действие LOG регистрирует сообщение, но продолжает обработку на базе правила в случае, если другие правила могут подходить для указания дальнейших действий. DROP приводит к тому, чтобы сообщение блокировалось , в то время как REJECT блокирует сообщение, но также отвечает отправкой пакета главному компьютеру, которым был заблокирован пакет.

Ниже приведены несколько примеров правил:

```
-A RH-Firewall-1-INPUT --dport 50 -j ACCEPT
-A RH-Firewall-1-INPUT -p UdP --dport 5353 -d 224.0.0.251 -j ACCEPT
-A RH-Firewall-1-INPUT -p TCP -m TCP --dport 631 -j ACCEPT
```

Как правило, файл Iptables завершается правилом по умолчанию. Если предыдущие правила определяют сообщения, чтобы принять или войти, то следующее правило будет: "backstop", чтобы отклонить все остальные сообщения.

```
-A RH-Firewall-1-INPUT -j REJECT --reject-icmp-host- prohibited
```

В файле / etc / fstab хранится информация об установке. Это применяется всякий раз, когда бы ни выдавалась команда - установить (установить все). Информация о конфигурации перечислена в каждом разделе по установке, точке установки (каталог), а также сообщается конкретная информация об этом разделе, такая как, имеет ли раздел выделенное дисковое пространство на отдельные записи, и является ли раздел доступен только для чтения. Например, одна запись:

LABEL = / Home / ext3 по умолчанию 1 2

В этой статье описывается, что / личный раздел должен быть установлен в директории / личный, тип раздела является ext3, раздел использует параметры по умолчанию, и имеет значения 1 и 2 соответственно, для частоты разгрузки (расписание архивации) и передает номер (контролирует заказ, что fsck программа проверяет разделы). Опции могут включать в себя ro (только для чтения), rw (чтение-запись), auto или noauto (должен ли новый раздел быть автоматически установлен во время загрузки или нет), а также usrquota (установить выделенные объемы на каждом каталоге пользователя). Тип ext3 часто встречается в системах Linux. Тем не менее, если раздел установлен по сети с помощью сетевой файловой системы (nfs), то дополнительный параметр для включения IP-адреса с помощью addr равен:
#. #. #. #, где # знаки представляют собой октеты IP-адреса.
Файл конфигурации /etc/resolv.conf является одним из самых простых конфигурационных файлов. Он хранит IP-адреса системы доменных имен (DNS-серверов) для компьютера. В отличие от предыдущих двух примеров файлов конфигурации, которые содержали ряд правил, с каким-либо уникальным синтаксисом, этот файл состоит из записей, которые выглядят как:

10.11.12.13 сервер имен
10.14.15.16 сервер имен

Файл resolv.conf используется многими программами Linux. Обсудим DNS в главе 12. Почтовый сервер в Linux это sendmail. Он имеет ряд файлов конфигурации. Один из них / etc./ aliases разрешает электронной почте псевдонимы, которые будут установлены. Например, адреса, такие как bin, daemon, adm, остановка и почта снабжены псевдонимами в коренном каталоге, так что, если сообщение посылается на любой из

этих мест, это на самом деле отправляется в корневой каталог. Другой файл конфигурации - это / etc. / mail / sendmail. cf. Тем не менее, предусмотрено, что вы никогда не должны напрямую редактировать этот файл. Вместо этого файл / etc. / mail/ sendmail.mc является файлом макро. Редактирование его позволит вам создать новый файл sendmail.cf. Поскольку sendmail является очень сложной программой, мы не будем рассматривать этот файл конфигурации.
Один последний / и так далее файл конфигурации стоит отметить, это ldap.conf. LDAP является сервером и используется для выполнения сетевой проверки права на доступ. То есть, учитывая сеть компьютеров, сервер LDAP может использоваться для аутентификации пользователя на любом из сетевых компьютеров. Хотя LDAP далеко выходит за рамки данного текста, его файл конфигурации стоит упомянуть. Формат файла конфигурации LDAP отличается от предыдущих файлов, где каждая запись была правилом или инструкцией к программе. Вместо этого, файл конфигурации LDAP содержит *директивы*. Существует много форм директив, каждая из которых выполняет различные задачи по подготовке сервера LDAP. Например, базовая директива указывает "различимое " имя для поиска. Другая директива, uri, объявляет IP-адреса различных типов для сокетов домена. Иные директивы определяют временные ограничения, портами, фильтрами, контексты именования, наименования карты и так далее. Ниже приведен пример файла ldap.conf:

URI LDAP: //127.0.0.1/ ssl no
tls_cacertdir / etc / open ldap / cacerts pam_ password md5

Изменение файла конфигурации сервиса не будет автоматически причиной того, что сервис принимает эти изменения. В большинстве случаев, сервис должен быть перезапущен. Вы можете сделать это, либо остановкой сервиса, а затем запуском его заново, или с помощью команды перезапустить вместо этого. Во многих случаях, имеет смысл остановить сервис до внесения изменений в файл конфигурации.

ОПРЕДЕЛЕНИЕ СЕРВИСА ВО ВРЕМЯ ЗАГРУЗКИ

Мы закончим эту главу, рассматривая, какие сервисы запускаются автоматически при загрузке системы или во время инициализации, а какие нет. Как описано, сервисами Windows можно управлять с помощью инструмента Services. Выбрав любую службу от этого

инструмента, вы можете запустить или остановить сервис. Через окно свойств сервиса, вы можете указать тип запуска. Возможны следующие варианты - автоматически (отложенный запуск), автоматический, ручной, отключение запуска. Как администратор, вы можете изменить тип запуска любых или всех сервисов, как пожелаете. Например, если она считает, что родительский контроль всегда должен быть запущен во время загрузки, вы можете изменить его тип запуска, с ручного на автоматический. Кроме того, служба поддержки Bluetooth это то , что вы можете захотеть запустить. Тем не менее, поскольку пользователями не требуется служба Bluetooth в момент загрузки системы, вы можете установить эту функцию, чтобы начать автоматический (отложенный запуск). Таким образом, сервис запускается вскоре после инициализации системы, как позволяет время.

Linux так же настроен на автоматический запуск определенных сервисов во время инициализации системы, а также сервисы могут регулироваться системными администраторами. Тем не менее, в отличие от Windows, в Linux вы можете установить различные сервисы запуска в зависимости от уровня выполнения. Есть семь уровней в Linux Run. Они приведены в таблице 11.3.

Когда Linux загружается впервые, он запускает script / etc / inittab. Одна из первых вещей, которые inittab устанавливает, является уровень запуска работы. Уровень запуска 5, например, запускает операционную систему в полноэкранном режиме с графическим интерфейсом, в то время как уровень запуска 3 представляет полный текстовый режим, и уровень запуска 1 является режимом единого пользовательского текста. От уровня запуска устанавливается, что службы запущены на основе уровня запуска работ. Inittab скрипт выполняет script/etc/rc.d/rc. Этот скрипт следит за несколькими действиями при запуске, но среди них, он выбирает каталог /etc/rc.d/rc#.d, где # является уровнем запуска (например, rc5.d для 5-го уровня работы).

Просмотр /etc/rc.d/rc5.d показывает ряд символических ссылок. Каждая символьная ссылка имеет имя в виде имени K ## или имя S ##. K и S обозначают "убить" и "старт", соответственно. Это письмо означает, будет ли значить сценарий RC убить или запустить сервис при запуске. Число обозначает порядок, в котором запускается сервис (или останавливается). Смотрите рисунок 11.18, в котором перечислены символические ссылки на уровне выполнения 5. Сравнение между каталогами /etc/rc.d/rc5.d и /etc/rc.d/rc3.d покажут сервисы, необходимые для графического интерфейса по сравнению с теми, которым он не нужен, тогда как

при сравнении между каталогами /etc/rc.d/rc3.d и /etc/rc.d/rc2.d, будут показаны сервисы, необходимые для коммуникации в сети по сравнению с теми, которые не нуждаются в этом.
Поскольку каждый элемент в этом каталоге является просто символической ссылкой, rc script переходит по ссылке к действующему сервису. Все сервисы хранятся в каталоге /etc/rc.d/init.d. Рисунок 11.19.

ТАБЛИЦА 11.3 Уровни запуска ОС Linux

Описание
0 Остановка отключает все службы, когда система не перезапускается
1 Однопользовательский режим, для обслуживания системы, действует как коренной каталог без сетевых возможностей
2 Многопользовательский режим без сети, в первую очередь, используется для технического обслуживания и тестирования
3 Многопользовательский режим с режимом сетевого текста на основе для штатной работы
4 Не используется
5 Многопользовательский режим с сетью и графическим интерфейсом - типичный режим для штатной работы
6 Отключение всех сервисов для перезагрузки системы

Каталог /etc/rc.d также содержит два дополнительных скрипта: rc.sysinit и rc.local. Скрипт rc.sysinit инициализирует ряд системных настроек, клавиатуру для использования, переменные окружающей среды, а также выполняет другие действия по настройке оборудования. Rc.local script доступен для системного администратора для дальнейшей настройки

```
K01dnsmasq          K50snmpd            S07iscsid       S19rpcgssd      S80sendmail
K02avahi-dnsconfd   K50snmptrapd        S08ip6tables    S22messagebus   S85gpm
K02NetworkManager   K69rpcsvcgssd       S08iptables     S25bluetooth    S90crond
K02oddjobd          K73ypbind           S08mcstrans     S25netfs        S90xfs
K05conman           K74nscd             S09isdn         S25pcscd        S95anacron
K05saslauthd        K74ntpd             S10network      S26acpid        S95atd
K05wdaemon          K85mdmpd            S11auditd       S26apmd         S96readahead_later
K10psacct           K87multipathd       S12restorecond  S26haldaemon    S97yum-updatesd
K10tcsd             K88wpa_supplicant   S12syslog       S26hidd         S98avahi-daemon
K15httpd            K89dund             S13cpuspeed     S26lm_sensors   S99firstboot
K20nfs              K89netplugd         S13irqbalance   S26lvm2-monitor S99local
K24irda             K89pand             S13iscsi        S28autofs       S99smartd
K35vncserver        K89rdisc            S13portmap      S50hplip
K35winbind          K91capi             S14nfslock      S55sshd
K50ibmasm           S04readahead_early  S15mdmonitor    S56cups
K50netconsole       S05kudzu            S18rpcidmapd    S56rawdevices
```

Рисунок 11.18
Каталог /etc/rc.d/rc5.d запуска сервисов.

среды. Он выполняет скрипт только один раз, все остальные скрипты инициализации выполняются сами. Этот файл изначально почти пуст, чтобы начать с него.

Как системный администратор, вы можете дополнительно адаптировать уровни выполнения простым изменением имен символических ссылок. Например, если он считается системной администрацией, то служба Bluetooth не будет нужна на уровне запуска 5, системный администратор может переименовать символическую ссылку S25bluetooth под /etc/ rc.d/rc5.d к K25bluetooth. То есть, вместо запуска сервиса Bluetooth, он будет убит. Конечно, если вы не знаете об этом, то лучше оставить эти символические ссылки в покое. Вы всегда можете запустить или остановить сервисы на более позднем этапе, используя / SBIN Bluetooth сервис системы старт / стоп.

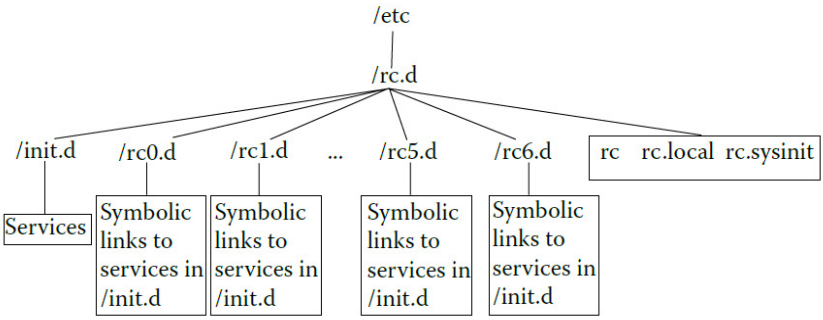


Рисунок 11.19
Структура каталога /etc/rc.d.иллюстрирует структуру поддиректории для сценариев запуска.

Service – Сервисы
Symbolic links to services in/init.d – Символические ссылки на сервера в /init.d

ГЛАВА 12. СЕТИ, СЕТЬ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ИНТЕРНЕТ

В этой главе описываются компьютерные сети. Эта глава начинается с описания компьютерного оборудования, которое составляет физические компоненты компьютерной сети. Далее, сети обсуждаются на уровне архитектуры: классификации сетей и сетевых протоколов. Сетевое программное обеспечение, в том числе специальное программное обеспечение Linux включены в главу. И, наконец, в данной главе рассматривается Интернет: что заставляет его работать, как он появился и каким может быть его будущее.
Цели обучения данной главы это:

• Опишите роль устройств сети вещания.
• Различите типы сетевых сред.
• Сравнение сетевых топологий и классификаций.
• Обсудите роль каждого уровня в стеке протоколов TCP / IP и модели OSI.
• Объясните способ адресации IPv4 и сравните его с IPv6.
• Введите популярные формы сетевого программного обеспечения.
• Представьте краткую историю Интернета.
• Опишите, как происходит обмен данными через Интернет.

Компьютерная сеть представляет собой набор компьютеров и компьютерных ресурсов (например, принтеры, файловые серверы), подключенные таким образом, что компьютеры могут обмениваться данными и ресурсами. Через компьютерные сети, люди могут общаться, обмениваться информацией, совместно использовать оборудование, изолировать и защищать данные, а также обеспечить платформу для удобного резервного копирования данных. Сети (в частности, Интернет) также предлагают средства для торговли и продаж. На самом деле, существует целый ряд различных преимуществ, которые предоставил Интернет.
Хотя сети признаны для повышения эффективности на рабочем месте путем совместного использования данных, ресурсов и коммуникации,

существует целый ряд затрат, связанных с какой-либо компьютерной сетью. Существует стоимость, чтобы приобрести и установить физические носители и администрировать сеть. Однако, это далеко не проблема, поскольку стоимость защиты сети должным образом, и стоимость наличия незащищенной сети существенно различаются. Кроме того, с помощью компьютерных сетей, большинство людей имеют доступ к Интернету, а также на рабочем месте, это может привести к нежелательному поведению и неэффективному использованию времени. Некоторые из угроз компьютерной сети и мероприятий, необходимых в обеспечении компьютерной сети рассматриваются в главе 15.

Хотя сеть может состоять из многих видов ресурсов, это компьютер, который является основным инструментом, используемым для общения по сети. В сети, мы определяем компьютер как локальный или удаленный. Локальный компьютер это компьютер, который использует пользователь. То есть пользователь физически присутствует около этого компьютера. Удаленный компьютер представляет собой компьютер для осуществления доступа по сети. Например, с помощью подключения к удаленному рабочему столу или telnet [или ssh (secure shell)], вы можете войти на другой компьютер. Компьютер на входе упоминается как удаленный компьютер. Хост представляет собой тип компьютера, который может быть зарегистрирован из удаленного местоположения. Раньше это было важным различием в том, что многие персональные компьютеры не могут быть хост-компьютерами. Но сегодня совсем не нужно, чтобы большинство компьютеров позволяли некоторую форму удаленного входа или подключения к удаленному рабочему столу.

Компьютерные сети можно просматривать на физическом уровне (на подключениях или физических носителях, по которым возможна связь), на логическом уровне (сетевой топологии) или на программном уровне (протоколов и программ, которые позволяют компьютерам связываться). В этой главе рассматриваются некоторые из идей, лежащих в основе компьютерных сетей на каждом из этих уровней.

РАБОЧИЕ СЕТИ НА ФИЗИЧЕСКОМ УРОВНЕ

Физический уровень сети определяет то, как осуществляется обмен информацией по сети. На этом уровне информация передается в виде битов в течение некоторого типа медиа-физического подсоединения. Средства массовой информации передают информацию в виде

электрического тока, электромагнитных волн, световых импульсов, или радиоволн (иногда при сверхвысоких частотах, таких как микроволновые печи). Форма передачи основана на типе выбранного носителя. Тип носителя выбран, по меньшей мере, частично, на основании расстояния между ресурсами в сети. Форма СМИ диктует также, в некоторой степени, пропускную способность сети. Ширина полосы пропускания это скорость передачи данных, допустимой для СМИ, описывается как некоторое число битов в секунду (бит или б / с). Современные полосы пропускания порядка миллионов бит в секунду (Мбит / сек, Mbps). Более старые технологии, такие как компьютерные модемы (описанные ниже), были ограничены до сотен или тысяч бит в секунду, например, 56 Кбит. Наиболее распространенной формой сетевого подключения, используемый сегодня коаксиальный кабель и волоконно-оптический кабель. В прошлом наиболее распространенным соединение было по витой паре проводов, как оно используется в большей части телефонной сети Соединенных Штатов. Мы до сих пор широко используем витую пару, потому что это дешево, и потому что большая часть его уже проложена. И коаксиальный кабель и витая пара передают информацию с помощью электромагнитных волн, в то время как волоконно-оптический кабель использует световые импульсы. Для больших расстояний, кабель слишком дорогой, и поэтому радиосигналы передаются по радио вышкам, вышкам сотовой связи, микроволновым башням, и отражается от спутников на орбите. Смотрите рисунок 12.1 для сравнения витой пары проводов (четыре в одном кабеле на этом рисунке), коаксиальный кабель, а также десятки нитей волоконно-оптического кабеля.

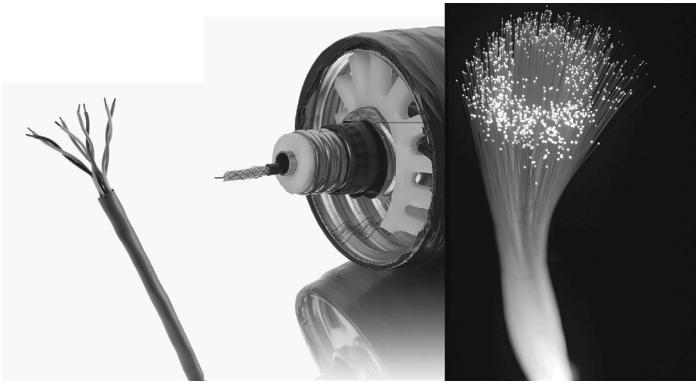


Рис. 12.1
Виды кабеля

На этом физическом уровне сеть отвечает за кодирование или декодирование данных в сигналы, модуляции и демодуляции сигналов, передачи и приема сигналов и маршрутизации сигналов. Передача является последним шагом, что сеть выполняет это при отправке сообщения, а прием является первым шагом при получении сообщения. Кодирование / декодирование требует перевода отдельных битов в сообщении от формы, хранящейся в компьютере к форме, которую требует сеть. Сообщения , которые хранятся в памяти компьютера, состоят из электрических зарядов (тока), тогда как при передаче по оптоволоконному кабелю, то сообщения становятся серией световых импульсов. Если физический носитель переносит аналоговый сигнал, а не цифровой сигнал, дальнейший перевод необходим, он известен как модуляция . Когда сигнал передается по телефонной линии, последовательность 1 и 0 переводится в тон, который и будет транслироваться. Демодуляция переводит от аналогового сигнала к исходному цифровому сигналу. И, наконец, маршрутизации направляет сообщение от одного местоположения в сети к другому.

Форма маршрутизации зависит от того, включен ли сетевой пакет или выполняется коммутация каналов. Сети с коммутацией каналов требуют, чтобы полный путь, или цепь, были установлены до того, когда передача может начаться и поддерживаться в течение всей длительности. Телефонная сеть представляет собой сеть с коммутацией каналов. Поскольку существует путь, то в течение всего разговора связь в любом направлении может происходить одновременно. В сети с коммутацией пакетов, путь сообщения только устанавливается так, как оно отправляется. Когда сообщение получено в одном месте, и если это не место назначения, то сообщение переадресовано в другое место. Выбор пути основан на доступности сети и объеме трафика сообщений. Большинство компьютерных сетей с коммутацией пакетов, с Интернетом

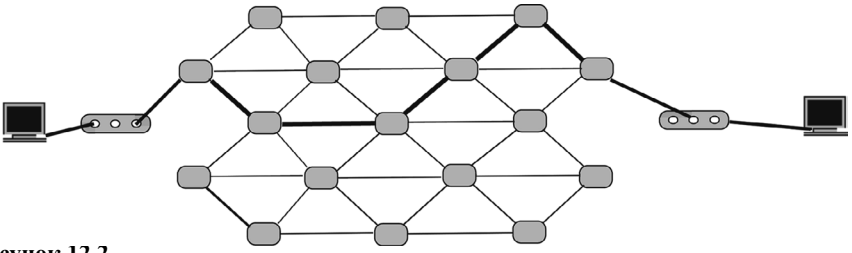


Рисунок 12.2
Путь через сеть.

является наиболее известным и часто приводимый пример. На рисунке 12.2 показана сеть, в которой сообщение передается от одного места в другое, пока не прибудет в пункт назначения. Если сеть с коммутацией каналов, маршрут будет создан заранее. Если сеть с коммутацией пакетов, маршрут будет создан, как одна ветвь за один раз. Таким образом, в пакетной коммутации, несколько сообщений между двумя ресурсами может пересечь сеть, используя различные пути.

Кроме физических сред, которые соединяют сетевые ресурсы вместе, существуют и другие устройства, которые используются для передачи сообщений из одного места в другое. Вещательные устройства состоят из концентраторов, коммутаторов, маршрутизаторов и шлюзов. В совокупности эти устройства «склеивают» сеть вместе посредством предоставления точек, где сеть может подключаться к устройствам и к другим сетям.

Сетевой концентратор это просто устройство, которое соединяет несколько компьютеров вместе. При наличии нескольких компьютеров, подключенных к концентратору, сообщение, принятое концентратором, становится доступным для всех компьютеров в сети. Адрес цели, прикрепленной к сообщению, указывает, к какому компьютеру сообщение предназначено, но это не мешает другим компьютерам подбирать сообщение. Сетевой коммутатор является более восприимчивым соединением, чем концентратор. С одной стороны, он записывает локальные сетевые адреса [управления доступом к среде (MAC) адреса] всех компьютеров, подключенных к коммутатору. Сообщение затем передается только вместе с компьютером, который соответствует адресу назначения. Таким образом, коммутатор способен более эффективно использовать пропускную способность сети.

Маршрутизатор представляет собой устройство, которое соединяет несколько сетей вместе. Таким образом, маршрутизатор как переключатель для переключателей. Хотя вы можете использовать маршрутизатор для непосредственного подключения компьютеров в сети, маршрутизаторы обычно используются в определенных местах в сети, так чтобы сообщения могли быть направлены на другие сети. Рисунок 12.3 представляет собой сетевой концентратор и рисунок 12.4 показывает несколько сетевых коммутаторов с подключенными устройствами. Внешне, немного различий, чтобы дифференцировать концентратор от коммутатора с маршрутизатором (кроме, возможно, размера). Внутренне, коммутатор имеет больше оборудования, в том числе

места для хранения MAC-адресов, чем концентратор. Маршрутизатор содержит программируемые таблицы маршрутизации и включает в себя, по меньшей мере, один вход, который поступает из другой сети.

Сеть, ПО и интернет.

Коммутатор и маршрутизатор используют адрес получателя в сообщении, чтобы выбрать линию для маршрутизации входящего сообщения. Маршрутизатор имеет дополнительные возможности для принятия решений. Например, если сообщение трафика превышает его емкость, маршрутизатор, возможно, придется целенаправленно опустить сообщения. Кроме того, при наличии нескольких входящих сообщений, маршрутизатор должен выбрать, что направить в первую очередь. Решение маршрутизации это сеть назначения, в которую сообщение помещено. Это решение, как правило, оставляют до таблицы маршрутизации, которая хранится в памяти маршрутизатора.
Сетевой шлюз представляет собой маршрутизатор, который соединяет

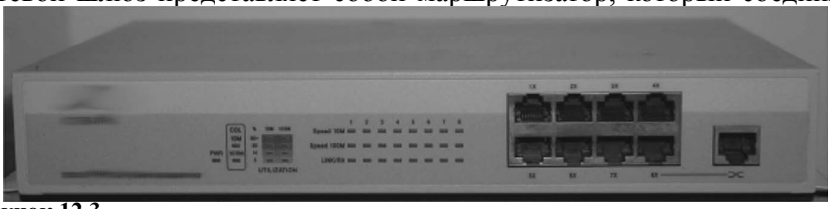


Рисунок 12.3
Сетевой концентратор.

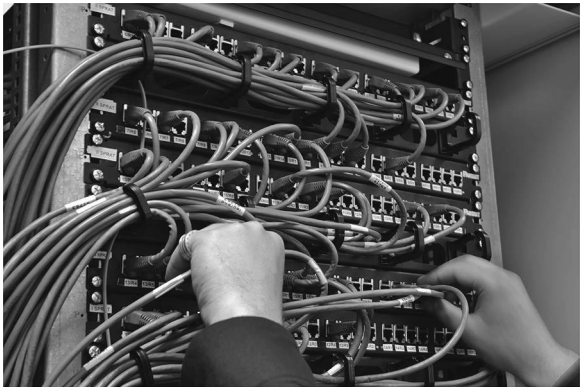


Рисунок 12.4
Сетевой коммутатор.

сети различных типов. То есть, если есть две сети, которые используют различные протоколы, шлюз способен не только маршрутизировать сообщения из одной сети в другую, но и обрабатывать различия в самих сообщениях из-за различных протоколов (мы обсудим протоколы сети на логическом уровне). Шлюзы появляются на границе сети, поскольку они соединяют разные типы сетей. Шлюзы, в отличие от маршрутизаторов, коммутаторов и концентраторов, не являются основными компонентами внутри сети. По этой причине, шлюзы могут также служить в качестве брандмауэров.

На рисунке 12.5 показано, как отличаются эти четыре вещательных устройства. Концентратор просто передает любое сообщение для всех устройств, таким образом, это общая связь (независимо от того, было предназначено сообщение или нет). Это показано в верхней левой части рисунка 12.5, где первый (крайний левый) компьютеру посылает сообщение к концентратору, который затем распространяется на остальные компьютеры. С другой стороны, коммутатор, показанный в части рисунка 12.5 снизу слева, передает сообщение только на одно устройство назначения, так что связь выделена. Маршрутизатор, как показано в правой части рисунка 12.5, соединяет несколько локальных сетей таким образом, чтобы один концентратор или коммутатор мог быть подключен к другому. На чертеже, второй справа компьютер в верхней сети посылает сообщение на крайнем левом компьютере в нижней сети.

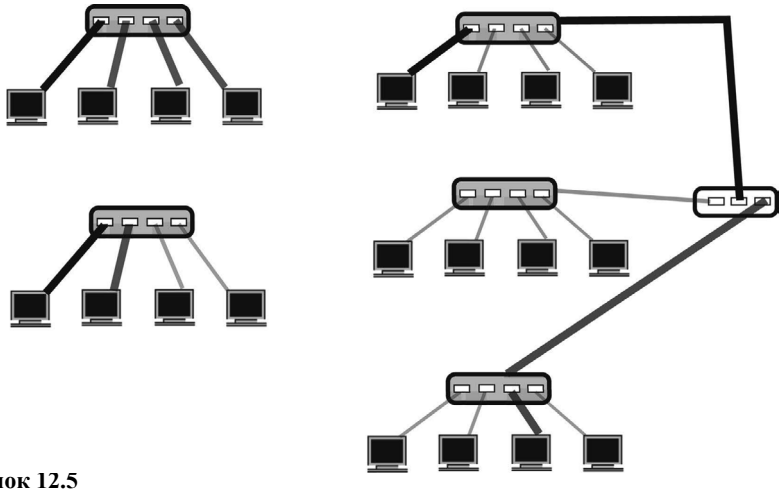


Рисунок 12.5
Концентратор (вверху слева), коммутатор (слева ниже), и маршрутизаторы (справа).

Как правило, маршрутизатор использует адрес получателя в сообщении, чтобы определить сеть для маршрутизации сообщения. Шлюз, который также будет выглядеть, как показано в правой части рисунка 12.5, служит той же цели, что и маршрутизатор за исключением того, что он может конвертировать из одного протокола в другой при прохождении сообщения из одной сети в другую. Затем шлюз используется для подключения различных типов сетей вместе, в то время как маршрутизатор соединяет сети одного и того же типа (протокол) вместе.

Помимо общения по компьютерным сетям, пользователи могут общаться с помощью телефонной системы. На самом деле, это было наиболее распространенное средство для телекоммуникации десятилетий.

Телефонная сеть настроена для передачи сигналов в аналоговой форме по витой паре проводов. Данные компьютера хранятся в бинарном виде, либо в виде электрического тока внутри проводов или магнитных зарядов, хранящихся на диске или ленте. Двоичные данные должны сначала быть преобразованы в аналоговую форму. Это требует модуляции (и преобразования цифрового сигнала в аналоговый). Смотрите рисунок 12.6, где последовательность 1001 (или высокий, низкий, низкий, высокий ток) должны быть переведены в аналоговую форму, или звук, в котором волновые формы ближе друг к другу, чтобы представить 1S и дальше друг от друга для 0s, который слышен человеческим ухом, как разные тона. В таком виде, данные могут быть переданы по телефонным линиям. Приемное устройство должно преобразовать аналоговый сигнал обратно в цифровую форму. Это демодуляция (аналого-цифровое преобразование). Модем представляет собой устройство, которое выполняет модуляцию и демодуляцию. Пользователь присоединяет модем к телефонной линии (например, телефонная розетка), а затем поместить телефонный звонок к пункту назначения, который также имеет модем. Далее, телефонная

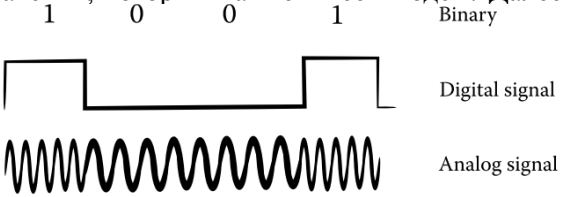


Рисунок 12.6
Модуляция сигнала.

Бинарный
Цифровой сигнал
Аналоговый сигнал

трубка вставляется в MODEM подставку, как показано на рисунке 12.7. Компьютеры на обоих концах теперь могут взаимодействовать друг с другом. После общения, трубка помещается обратно на телефон, чтобы закончить телефонный звонок. Сегодня модемы встроены в компьютер так, что вы можете подключить ваш компьютер непосредственно к телефонной розетке, хотя чаще всего, беспроводная карта позволяет компьютеру взаимодействовать с беспроводным модемом, который можно разместить на уровне почти любой точки в доме.

Сеть на логическом уровне

Есть много способов, чтобы компьютерные ресурсы могли быть соединены друг с другом через компьютерную сеть. Различные макеты вместе называются топологией сети. Выбранная топология обуславливает стоимость сети, а также количество времени, которое может потребоваться для сообщения, которое будет направляться к месту назначения. Кроме того, топология может воздействовать на надежность сети. Рисунок 12.8 демонстрирует множество различных топологий. На рисунке компьютеры и компьютерные ресурсы (файловые серверы, принтеры, концентраторы, коммутаторы, маршрутизаторы, CD-ROM антенны, ленточные накопители и так далее), все показано в виде круга. Линии, соединяющие узлы вместе представляют собой соединения в сети, как правило, состоят из кабеля (как показано на рисунке 12.1, это может быть скрученная пара проводов: коаксиальный кабель, волоконно-

Рисунок 12.7
МОДЕМ люлька.

оптический кабель, или любые комбинации этих вариантов).Различные топологии шинной топологии (вверху слева), топология звезды (средний левый), кольцевой топологии (справа от звезды), полный меш (внизу слева), и между ближайшими соседями топологий на правой стороне фигуры [один мерный (1-D), 2-Д, дерево, 3-D]. Не показанная на рисунке - это топология 4-D известная как гиперкуб.

Самая простая форма сети представляет собой сеть точка-точка, где существует специальная связь между двумя ресурсами. В топологии сети, точка-точка не часто используется, поскольку это ограничивает взаимодействие устройства друг с другом. Мы видим, что точка-точка подключение в ходах внутри компьютера, таких как шины, соединяющие ALU регистры в CPU, или соединение между монитором и системной платой. Мы также можем видеть, из прямых отрезков между сетями, точку в виде междугородней связи, таких как выделенные телефонные линии, между двумя точками (например, знаменитый красный телефон, который соединяет Белый дом и Кремль).

Топология шины сети - это следующая простая форма, которая является очень распространенным явлением. В этой топологии, каждый ресурс подключен к одному кабелю. Кабель сети несет одно сообщение за один раз. Устройства подключаются к шине через связь "T". Каждое устройство будет подключено к разъему "T", чтобы подключиться к двум кабелям, каждый из которых ведет к следующему устройству и разъему "T". Каждый конец сети будет подключен к терминатору. Кабель будет иметь ограниченное количество соединений "T" так, что размер сети

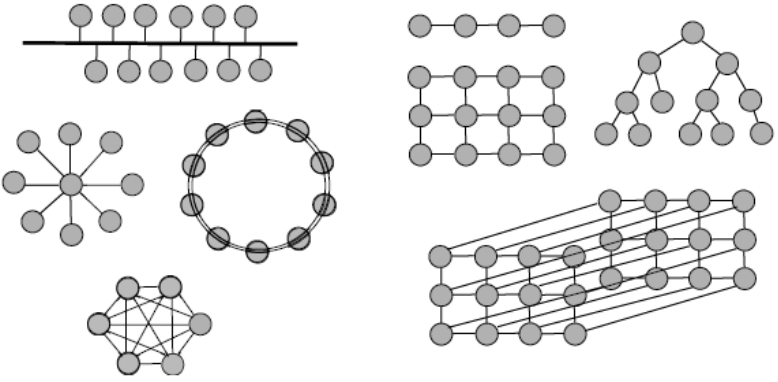
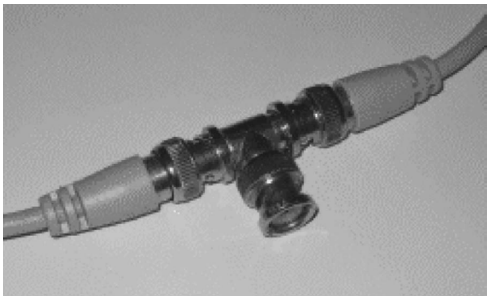


Рисунок 12.8
Различные сетевые топологии, узлы - это компьютерные ресурсы опор соединения.

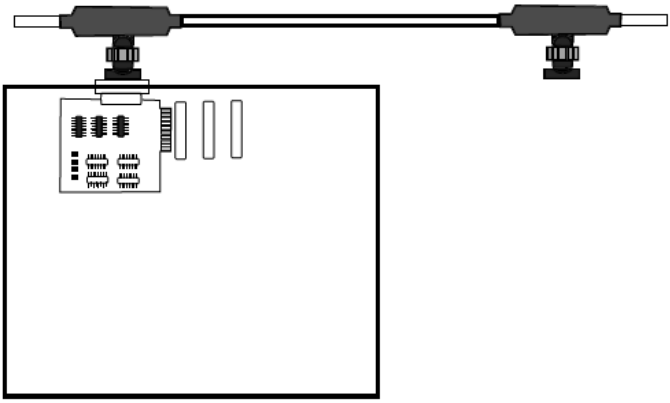
будет ограничен. На рисунке 12.9a показан разъем "T" и на рисунке 12.9b показано, как компьютеры подключаются к разъемам "T" через сетевые карты. Сетевая карта вставляется в слот расширения на материнской плате компьютера. Соединитель "T" прорезан в порт таким образом, чтобы он выглядывал из задней части системного блока.

Есть и другое средство для подключения устройств к одной сети через концентратор или коммутатор. В этом случае, устройства подключаются к одному соединению через заднюю часть концентратора или коммутатора. Если сеть нуждается в расширении, концентраторы или коммутаторы можно последовательно соединить в цепь.

На трассе сети, все устройства для прослушивания любых коммуникаций через единый кабель можно игнорировать все сообщения, которые не предназначены для них. Шина обеспечивает динамическую сеть, в которой могут быть добавлены и удалены из сети устройства в любое



(а)



(б)

Рисунок 12.9
Соединение "T" (а) и соединения "T" сетевых карт (б).

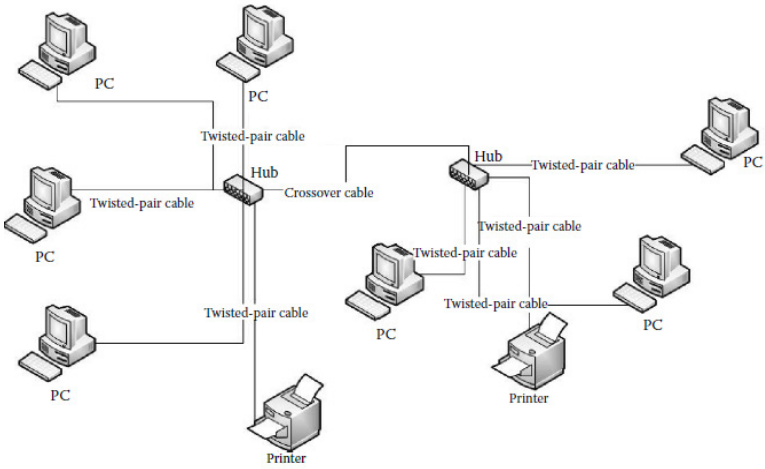
время до тех пор, пока есть в наличии разъемы "T". Трасса сети - это самая дешевая из всех сетевых топологий. Это надежная форма сети, которая не ломается, если ресурс либо удален из сети или возник сам по себе. Мы увидим, что другие формы сетей могут привести к снижению подачи сети, когда ресурс подачи падает. Несмотря на это, по топологии шины, сеть имеет большой недостаток. Все сообщения проходят по одному кабелю, и поэтому эффективность работы сети ухудшается, по мере большого количества устройств, подключённых к нему. Чем больше количество устройств, подключенных к нему, тем больше издержек будет для сети. Это, в свою очередь, создает больший трафик сообщений. Таким образом, вероятность того, что два устройства, которые необходимо использовать сеть в то же время добавляются по мере добавления устройств к сети.
Что произойдет, если два устройства попытаются общаться в то же самое время? Это известно как сообщение утверждения, а результат – это то, что сообщения будут мешать друг другу. Таким образом, после того, как обнаруживается конфликт, по меньшей мере, одно из устройств, должно ждать, в то время как другое устройство связывается. Стратегия обработки сообщений конфликтом обсуждается далее в этой главе, разработанной для технологии Интернет.

Звездные сети в некотором роде противоположность дорожной сети. В звездной сети, все устройства имеют одно соединение точка-точка с центральным сервером. Этот сервер представляет собой устройство, которое действует в качестве концентратора связи, маршрутизации сообщений от одной машины к месту назначения. Преимущества звездообразной сети это его простота, легкость добавления (или удаления) устройств, а также эффективность перенесения сообщений. Звездная сеть не ведет борьбу с трафиком сообщений, как дорожные сети (хотя концентратор может быстро стать узкой пропускной способностью, если есть много одновременных сообщений). Количество ссылок, которые он принимает для любого сообщения, чтобы достичь своего назначения всегда одна, если сообщение предназначено для концентратора устройства.
Звездная сеть имеет две проблемы. Во-первых, это требует специального устройства, что делает его более дорогой сетью, чем дорожная сеть. Концентратор звездной сети может быть концентратором или коммутатором, как описано в сети на аппаратном уровне. Ступица звездообразной сети также может быть сервером, который сделает сеть

звездной и более дорогой. Во-вторых, хотя потеря одного устройства не будет ухудшать сеть, если концентратор теряется, все устройства теряют подключение.

Общий подход к построению локальных сетей на сегодняшний день - это подключение звездной сети, работающей вместе. Это делается путем последовательного подключения узлов вместе. Так, например, концентратор используется для подключения ресурсов одной сети вместе, и концентратор соединен с другим концентратором, который сам по себе соединяет ресурсы второй сети вместе. Это создает большую единую сеть, в которой все устройства могут взаимодействовать друг с другом. В этом случае имеется дополнительная передача, которая требуется между некоторыми из ресурсов, если они подключены к разным ступицам. Как концентраторы (а также коммутаторы, маршрутизаторы и шлюзы) будут иметь ограниченное число соединений, этот подход позволяет сети легко увеличиваться. Все, что необходимо это дополнительные концентраторы. Рисунок 12.10 иллюстрирует эту концепцию. "Перекрестный кабель" используется для соединения двух концентраторов вместе. В данном конкретном чертеже, устройства в сети все связаны с концентраторами по витой паре проводов.

В кольцевой сети, устройства имеют соединения типа точка-точка со своими соседями. Связь между устройствами требует отправки



Перевод: twisted pair cable - Витая пара кабеля ; PK - ПК ; printer - принтер

Рисунок 12.10

Две дорожные сети, соединенные хабами.

сообщения к соседу, и сосед пересылает сообщение по кольцу, пока не достигнет своего пункта назначения. Кольцо легко расширяется за счет добавления новых устройств между любыми двумя узлами. Тем не менее, чем больше кольцо, тем больше возможное расстояние, которое должно преодолеть сообщение. Это, в свою очередь, может создать продолжительные время передачи. Кольца дешевые, как дорожные сети, поскольку не существует выделенный ресурс, который служит в качестве центральной точки. Кольцевая сеть может быть универсально направленной или двунаправленной с точки зрения направления, по которым путешествуют сообщения. В однонаправленное кольцо, если устройство перешло бы в автономный режим, разделило бы сеть на две части таким образом, что некоторые устройства не могли больше общаться с другими. Предположим, например, что топология это кольцо, показанное на рисунке 12.8, было однонаправленное и все коммуникации путешествовали по часовой стрелке вокруг кольца. Если верхнее устройство выйдет из строя, то устройство непосредственно слева от отказавшего устройства не сможет добраться до любого другого устройства, в то время как устройство непосредственно справа от отказавшего устройства не сможет получать сообщения с любого другого устройства. Двунаправленное кольцо потребует потери двух устройств, прежде чем части сети окажутся изолированными.

Ячеистая сеть, также известная как полностью подключенная сеть, которая требует, чтобы каждое устройство имело соединение точка-точка с каждым другим устройством. Если есть N устройств в сети, много, как (п - 1) 2 соединений. Любое устройство может непосредственно взаимодействовать с любым другим устройством, не заботясь о трафике сообщений, и если устройство было удалено, оно не повредит связности сети. Тем не менее, ячеистая сеть является ограниченной из-за стоимости. Мало того, что это требует чрезмерного количества соединений, сеть требует, чтобы каждое устройство имело п - 1 доступных портов, чтобы подключить эти соединения. Ячеистая сеть, как правило, находится в компьютерах параллельной обработки высокой производительности, но не используются в компьютерных сетях. Небольшая ячеистая сеть из шести устройств показана на рисунке 12.8. С шестью устройствами, есть в общей сложности 25 соединений. Представьте себе, количество соединений, необходимых для сетки 50 устройств. Для этого потребуется 2401 соединений!

Кольцевая сеть является одной из форм ближайшего соседа сети: соседи представляют собой устройства, которые непосредственно связаны друг

с другом. Ячеистая сеть является конечной формой ближайшего соседа в том, что каждое устройство подключено к любому другому устройству, так как каждое устройство является ближайшим соседом. Есть много промежуточных форм ближайших соседних сетей. К ним относятся дерево сети: одно устройство является корневым узлом дерева, и оно подключается к одному или нескольким дочерним узлам, и каждый ребенок подключается к одному или нескольким узлам, и так далее. Общее дерево представляет собой бинарное дерево, в котором каждый узел имеет до двух отраслей. Если бинарное дерево сети имеет n уровни, то будет 2^n - 1 узел, и сообщение может потребовать больше, чем 2 * (п - 1) трансмиссии между узлами источника и назначения. На рисунке 12.8 показаны четыре формы ближайшего соседа сети, 1-D сеть A, 2-D сеть A, 3-D сеть (не полностью погашенная) и дерево сети. A 4-D сеть носит название гиперкуб.

Помимо топологии, мы можем классифицировать сети, основанные на их физических размерах. Размер не является мерой количества ресурсов, так же, как расстояние между ресурсами. Наиболее распространенный размер сети сегодня является ЛС (Локальная сеть). Локальная сеть может быть подключена (компоненты соединены с помощью физической среды, такие как витая пара, коаксиальный кабель или волоконно-оптический кабель) или беспроводная сеть (компоненты взаимодействуют между собой либо с помощью инфракрасного света или радиосигналов некоторого вида). Большинство локальных сетей - это проводная связь, или сочетание проводной и беспроводной связи. Чисто беспроводная локальная сеть не является распространенным явлением.

Многие локальные сети на сегодняшний день представляют собой коллекции небольших локальных сетей. Рассмотрим университет в качестве примера. Университет будет иметь компьютерные лаборатории. В какой-либо из одной лаборатории, есть одна ЛС. Ресурсы лаборатории соединены между собой с помощью некоторых широковещательных устройств, таких как концентратор или (что более вероятно) переключатели. Если помещение достаточно велико, может быть несколько концентраторов или коммутаторов соединённых друг с другом. На самом деле, если есть слишком много устройств, может быть маршрутизатор, который подключается к нескольким коммутаторам. Это позволило бы создать несколько локальных сетей в пределах одной комнаты.

Учитывая, что это университет, здание, без сомнения, содержит несколько компьютерных лабораторий. Каждая лаборатория (комната)

соединена с другими лабораториями через маршрутизатор. Это дает нам локальную сеть. Эти локальные сети подключатся к другим локальным сетям в здании снова через маршрутизаторы, и, возможно, шлюзы. Например, там может быть четыре локальные сети на первом этаже здания, машины, дом, окна и две локальные сети на втором этаже, это домашние компьютеры Macintosh. Четыре первые локальные сети полу соединены друг с другом с помощью маршрутизатора, а также две лаборатории, проводимые второй, соединены друг с другом с помощью маршрутизатора. Два маршрутизатора соединены между собой шлюзом. Другие этажи здания имеют свои собственные локальные сети, чтобы соединить вместе аудитории и офисы факультета. Таким образом, здание содержит локальную сеть. И это здание соединяется с другими зданиями, в свою очередь. Весь студенческий городок представляет собой локальные сети локальных сетей.

А может (Сеть в масштабе кампуса), как следует из названия это сеть, которая охватывает сайт какой-либо организации, такой, как в университетском городке. Это, очевидно, локальная сеть локальных сетей, как описано в предыдущем параграфе. А также, возможно, может выходить за пределы одной региональной области, если кампус имеет филиал в городе. На другом полюсе локальных сетей является ПС (Персональная сеть). ПС - это небольшая локальная сеть и, возможно, подключение воедино ресурсов, находящаяся в домашнем хозяйстве человека. Многие пользователи домашних компьютеров теперь имеют сети, в которых все домашние компьютеры и принтеры соединены друг с другом. Они могут быть подключены полностью через беспроводные соединения, или некоторые ресурсы, такие как принтер. Устройство подключения этих устройств может быть сетевым коммутатором или концентратором, или это может быть модем, который имеет возможность служить в качестве концентратора.

Локальная сеть, которая использует протокол Интернета (смотрите в разделе Протоколы Сети) известен как интрасеть. Преимуществом интрасети является то, что приложения, предназначенные для использования Интернета (электронная почта, веб-браузер, Telnet / SSH и так далее) будут работать внутри так же легко, как они работают с внешним сигналом. Экстранет это расширение интрасети по сравнению с другими сетями, так что доступ к внутренней сети может быть выполнен с удаленных сайтов. Для этого требуется дополнительная безопасность, чтобы гарантировать, что удаленные пользователи фактически допущены. Как правило, используется процесс входа в систему. Виртуальная

частная сети (VPN) является распространенной формой внешней сети. Виртуальная частная сети часто использует технологию шифрования, так что сообщения, которые физически выходят за пределы внутренней сети не могут быть поняты, даже если перехватываются с помощью таких методов, как анализ пакетов (это обсуждается далее в этой главе).
Возможно, что ЛС и ПС небольшие сети в том, что они соединяют ресурсы в пределах маленькой близости (футах, возможно, на расстоянии мили). Большая сеть в городской сети (СС), которая охватывает часть или весь город. На маленьком конце, человек перекрывает МС в определении. Например, университеты, которые располагаются на протяжении нескольких квадратных миль и могут содержать СС, а не МС (различие здесь практически не имеет значения). На большем конце, человек покрывает весь город. В таком случае, высокоскоростные физические соединения волоконно-оптического кабеля или радио могут быть непрактичными. СС можно найти в таких городах как Лондон (Англия), Женева (Швейцария) и Сан-Франциско (Калифорния, США).
Мы можем продолжать расширять охват сети за пределы МС, чтобы создать ГС (Глобальная сеть). Они могут подключать ресурсы через округа, штаты, страны или весь мир. Одним из таких ГС называется «Sohonet», которая связывает воедино различные производственные линии и постпродакции компаний по всему миру (из Лос-Анджелеса в Нью-Йорк, Канаду, Новую Зеландию и многие европейские города). Самый большой глобальной сети Интернет.
Мы кратко рассмотрим одну из форм реализации для локальной сети. Один из наиболее популярных реализаций называется сеть Ethernet. Во-первых, разработанный в начале 1970-х годов, оригинал коммерческий релиз Ethernet был произведен компанией 3Com в 1980 году, Ethernet был стандартизированный в 1982 году и был конкурентом форм Token Bus и Token Ring сетей. Хотя изначально реализован с использованием топологии шины по коаксиальному кабелю (с Т-разъемами), современные сети Ethernet могут использовать витую пару и волоконно-оптический кабель с устройствами, подключенных к концентраторам или коммутаторам. Подключив ресурсы непосредственно в концентратор или коммутатор, это не только более эффективно, но это также снижает затраты на установку и улучшает управление сетью.
Технология Ethernet улучшалась на протяжении многих лет, с теоретической пропускной способностью 10 Мбит к текущей верхней полосой пропускания 100 Гбит. Ethernet ввел ряд сетевых концепций, которые обычно встречаются сегодня. Они включают в себя механизмы

обнаружения столкновений, ретрансляторы Ethernet, 48-битный MAC (управление доступом к среде передачи) схемы адресации для источника и назначения адресов, формат кадра Ethernet, обработка ошибок механизмов, а также различные адаптеры, так что много различных типов компьютеров может подключиться к Ethernet. Здесь мы кратко рассмотрим лишь схемы обнаружения столкновений.

Форма обнаружения столкновений, введенная Ethernet, называется Carrier Sense Multiple Access с обнаружением коллизий (CSMA / CD). Во-первых, устройство готовит сообщение для трансмита. Он пытается ощутить, занят ли сетевой медиа. Если да, то эти устройства не ждут, пока носитель будет свободен. Затем он помещает свое сообщение на носитель. Тем не менее, если другое устройство ожидает, оба устройства могут размещать сообщения на медиа, в тот же или почти тот же самый момент . Поэтому, даже если он передает, устройство также пытается передать , если любое другое сообщение приходит через средства массовой информации. Если да, то обнаруживается столкновение. Устройство немедленно прекращает передачу данных и посылает сигнал о заторе. Сигнал джем используется для предупреждения других ресурсов, чтобы не передавать в тот момент. Сигнал вареньем также, так как она передается через тот же носитель, который содержит фактическое сообщение, перекрывает сообщение. Любое приемное устройство будет принимать сигнал о заторе, и вы знаете, что оно получало сообщение, но оно было повреждено. В настоящее время, оба передающие устройства ждут некоторое время перед повторной попыткой их передачи. Когда переключатели были добавлены к Ethernet через концентраторы и автобусных остановок, были уменьшены столкновения, но CSMA / CD продолжали использоваться.

Мы также можем классифицировать сети по той роли, которую играют компьютеры в сети. В частности, мы имеем в виду сети либо как равный -равному или клиент-серверных сетей. Одноранговый узел означает, что каждый компьютер примерно равен любому другому компьютеру. Это отличает сеть от модели клиент-сервер. Клиент представляет собой компьютер, который будет запрашивать информацию с другого компьютера. Сервер представляет собой компьютер (или устройство), который принимает запросы и отвечает запрашиваемой информации. Затем сетевой клиент-сервер представляет собой сеть, которая содержит один или несколько серверов. Равные сети дешевле, хотя потребность серверов сил большинства сетей следует модели клиент-сервер. Серверы, во многих случаях, более дорогие компьютеры, так как они

обычно требуют большего хранения на жестком диске и быстрого времени отклика, чем другие компьютеры в сети. Серверы часто выглядят как большие системы единицы, возможно, с несколькими жесткими дисками. Большие файловые серверы часто это коллекция небольших блоков, установленных в шкафу, как показано на рисунке 12.11.

Есть различные типы серверов, основанных на типе желаемого сервиса. Один тип сервера это файловый сервер. В этом случае роль сервера есть отправки файлов по сети по требованию клиентов. Типичный файловый сервер отвечает на запросы через локальную сеть и услуги только на компьютерах локальной сети. Файл-серверы могут быть использованы для поддержки как прикладное программное обеспечение и файлы данных. То есть, клиенты будут хранить только свои собственные операционные системы. Пользователю, желающему запустить программное обеспечение, возможно, придется загрузить программное обеспечение по сети по требованию. Или файловый сервер может быть ограничен лишь немногими, совместными прикладным программным обеспечением и / или файлами данных.

Другой формой сервера является веб-сервер. Это особый тип файлового сервера, который долго хранит файлы данных (веб-страниц) и программ

Рисунок 12.11
Файловый сервер стеллажного типа.

(скриптов). Тем не менее, веб-сервер отвечает на запросы клиентов из любого места в Интернете, а не только в локальной сети, а также запросы специальные HTTP (протокол передачи гипертекста) запросы. Ответы могут быть HTML-файлы, документы, хранящиеся на файловом сервере, или веб-страницы, которые были динамически генерируемые сервером через CGI (Common Interface шлюз) скриптов. Веб-сервер имеет несколько обязанностей: файл-сервер не работает, в том числе на стороне сервера сценариев, регистрации запросов и ошибок, а также обработку безопасности. Еще одна форма сервера является сервером базы данных. Он отвечает на запросы к базе данных клиента с ответами вытянутыми из базы данных. Кроме того, сервер может генерировать отчеты на основе данных, полученных из системы управления базами данных. Как и файловый сервер, сервер базы данных типично реагирует только на местных клиентов. Как файловый сервер и сервер базы данных может реагировать на авторизованных клиентов из удаленных мест, если сервер был доступен более широкой сети. Но, как веб-сервер, обязанности сервера базы данных выходят далеко за рамки передачи файлов. Другие серверы включают в себя серверы печати, почтовые серверы и FTP-серверов. В отличие от других серверов, сервер печати не возвращает файл, но вместо этого контролирует задания на печать и отвечает с определением того, что задание на печать завершено, или сообщением об ошибке. В случае почтового сервера, один отправляет электронную почту на сервер, чтобы отправить на другой сервер электронной почты. Два сервера электронной почты общаются друг с другом, и после того, как электронная почта поступает в пункт назначения, сервер получателя информирует пользователя о том, что получатель имеет новое сообщение электронной почты. Сервер FTP это немного как веб-сервера или файлового сервера в этом запросов для файлов и ответы файлы.

Сетевые протоколы

Протокол является сводом правил, установленных для управления, как люди ведут себя и взаимодействуют друг с другом. Это может рассматриваться как форма дипломатии или этикета, или средства, с помощью которых исследователь сообщит результаты других исследователей. Компьютерная сеть существует в нескольких различных слоях. Мы могли бы подумать о том, как сеть работает на аппаратном уровне при обсуждении медиа и физической формы, которую связь будет принимать. Мы могли бы обсудить, как сеть будет обеспечивать

надежность за счет обработки ошибок информации. Или мы могли бы сосредоточиться на том, как прикладное программное обеспечение готовит сообщения для трансляции. На самом деле, каждый из этих шагов необходим, и мы должны определить не только то, как они работают, но как этапы взаимодействуют. То есть, мы будем рассматривать связь в сети как ряд слоев, и сообщение должно быть переведено из одной формы в другую, как оно перемещается от слоя к слою. Сетевой протокол обеспечивает правила, по которым слои сети взаимодействуют друг с другом. Это, в свою очередь, обеспечивает правила, по которым различные сети могут обмениваться данными с друг с другом. Это также информирует сетевых программистов и администраторов о том, как внедрять управление и обслуживание сети.

* Наиболее распространенный протокол на физическом уровне является протокол Ethernet, который определяет такие аспекты сети, как и типов кабелей, которые могут быть использованы, механизм обработки столкновений (CSMA / CD), а также допустимые типы топологий: шина, звезда, дерево. Другие существующие популярные протоколы LocalTalk, разработанные компьютеры Apple, Token Ring (разработанная компанией IBM) и ATM (асинхронный режим передачи), который непосредственно поддерживает аудио и видео передачи.

* Пожалуй, наиболее часто используемый сетевой протокол TCP / IP (протокол управления передачей / межсетевой протокол), * который является требованием всех компьютеров, которые обмениваются данными через Интернет. Таким образом, хотя компьютеры могут использовать другие сетевые протоколы, они также должны работать TCP / IP. В качестве альтернативы, модель взаимодействия открытых систем (OSI), был предложен стандарт для всех сетевых коммуникаций. Таким образом, несмотря на то, что TCP / IP является конкретным протоколом, OSI часто используется в качестве мишени для новых разработчиков сети. Сегодня, как TCP / IP и OSI, как правило, приводятся модели для сетевых протоколов, поэтому мы будем рассматривать их здесь. Модель OSI состоит из семи слоев, как показано в таблице 12.1. Каждый слой имеет номер, с самого верхнего слоя, пронумерованные как 7, и нижнем слое, как 1. Четыре верхние слои известны как принимающие слои, поскольку деятельность этих уровней имеет место на главном компьютере. Эти слои упаковки вместе это сообщение для передачи или приема сообщения и его анализ. Нижние три слоя известны как слои медиа, потому что в этих слоях, сообщения, адресованные, прокладываемые и физически переданные. Хотя OSI не является конкретным протоколом,

он был использован для создания сетевых протоколов, включая Протокол информации общего управления, X.400 электронная почта, обмен X.500 каталога сжатие, и IS-IS (Межшлюзовый системе) протокол маршрутизации.

* Слой 1, самый нижний слой, который является физическим уровнем. Этот слой определяется, как устройство, которое хочет общаться (например, компьютер или концентратор), и будет осуществлять связь по среде передачи (коаксиальный кабель, волоконно-оптический кабель и так далее). Этот слой требует таких деталей, как напряжение, необходимое для передачи, и для модуляции сигнала (при необходимости), TCP / IP, на самом деле это набор протоколов и иногда упоминается как стек протоколов.

Таблица 12.1 Модель OSI Слои

Блок данных	**Номер и название уровня**	**Функции**
Данные	7. Применение	Взаимодействие пользователя с прикладным программным обеспечением
	6. Представление	Предоставление данных, шифрование/дешифрование
	5. Сеанс	Узел сеансов уровня связи, управление сессиями
Сегменты	4. Транспортировка	Надежность и управление потоком
Связка/пакет с адресом доставки через сеть	3. Сеть	Сетевая логическая адресация, маршрутизация
Рамка	2. Канал передачи данных	Адрессация
Бит	1. Физически	Физические носители, передача сигнала в двоичном коде

как установить и как прекратить подключение к среде передачи данных. При установлении соединения, слой требует возможности обнаружения трафика сообщений (конкурирующий) и средство разрешения при наличии трафика. При передаче данных, этот слой получает данные из слоя 2 и должен преобразовать его в форму, подходящую для передачи. При получении данных, этот слой получает данные из среды передачи

Что такое 802?

Возможно, вы видели нотации, такие как IEEE 802.xx. что это значит? IEEE является Институт инженеров электротехники и электроники, среди различных усилий организации целый ряд стандартов, которые они установили. Хотя стандарты без законов или требований означает, что большинство исполнителей пытаются соответствовать установленным стандартам, чтобы гарантировать, что их усилия будут использоваться и годны к употреблению.
Большинство стандартов телекоммуникаций выдвигаются и они IEEE помечены как 802.xx, IEEE, где 802 обозначает "февраля 1980 года" за первый месяц, что IEEE встретились для обсуждения телекоммуникационных стандартов (это был также первый свободно доступный номер).
Что некоторые из стандартов, которые идут под IEEE 802?

- IEEE 802.3-Ethernet
- IEEE 802,7-Broadband LaN
- безопасность IEEE 802.10-Lan
- IEEE 802.11 беспроводная локальная сеть (число вариаций было произведено, каждое данное письмо, такую как 802.11 б, г, 802,11 и 802,11 п)
- Сертификация по стандарту IEEE 802.15.1-Bluetooth
- сеть 802.22 беспроводная зона IEEE
- IEEE 803,23-аварийно-спасательных служб для рабочих групп

в стороне от стандарта IEEE 802 IEEE установила ряд других стандартов, начиная с плавающей точкой представления в компьютерах (IEEE 754) к стандарту POSIX (Portable Работа системного интерфейса ING), что Unix и Linux системы отвечают (IEEE 1003) стандартам в области программной инженерии (IEEE 610).

(электрический ток, звуковые волны, световые импульсы) и преобразует данные в двоичном формате для совместного использования с слоя 2. Таким образом, этот слой имеет дело с сообщением в виде последовательности бит. Есть множество реализаций слоя 1, включая IEEE 802.3, IEEE 802.11, Bluetooth, USB и концентраторов.

Слой 2 является уровень канала передачи данных. Если сообщение предназначено для устройства в той же сети (например, тот, который разделяет один и тот же концентратор или коммутатор), коммуникация будет происходить на этом уровне, и не требует слоя 1. На этом уровне данные формируются в блоки, называемые кадрами. Рама должна указывать синхронизацию кадров, которая представляет собой последовательность битов в начале сообщения . Этот слой содержит два подслоя. Верхний подслой управления логическим каналом (LLC), подуровень. Этот слой обеспечивает мультиплексирование, которая является способность нести несколько перекрывающихся сообщений одновременно. Через мультиплексирование, то возможно, что несколько почтовые сообщения имеют разные сетевые протоколы, сосуществующие в то же время, разделяя той же сети средств массовой информации. Нижний подуровень MAC. MAC-адреса используются для обозначения местоположения устройства в пределах данной сети. Подуровень MAC позволяет устройствам в одной сети общаться друг с другом, используя только их адреса MAC. Этот подслой служит интерфейсом между LLC подслой и на физическом уровне. Слой 2 реализаций включают в себя IEEE 802.2, IEEE 802.3, PPP (точка-Point Protocol), X-25 пакетов обмена коммутатора, а также банкомат. Ethernet является реализацией для обоих слоев 1 и 2.
Слой 3 представляет собой сетевой уровень. На этом слое, физические характеристики сети не являются проблемой. Вместо этого слоя просматривает данные в качестве переменных последовательностей длины с хоста и адресами, назначенными операций. Хождения этого слоя обрабатывает маршрутизации операций. Сообщения, которые поступают на устройство, чья работа состоит в том, чтобы направить сообщение вперед рассмотрит адрес получателя с записями в своей таблице маршрутизации, чтобы определить, к какой сети, чтобы поместить сообщение . Поэтому маршрутизаторы работают на этом уровне. На этом слое, компоненты сообщения формируются в отдельные пакеты. Сообщение передается передающим устройством, скорее всего, состоять из нескольких пакетов, возможно, десятки или сотни, в

зависимости от размера сообщения. 3-го уровня реализаций включают в себя IP (см TCP / IP ниже), AppleTalk, IPX (Internet Packet Exchange), ICMP (протокол Internet Control Message) и ARP (Address Resolution Protocol).

Слой 4 является транспортный уровень. Его главная обязанность заключается в том, чтобы обеспечить прозрачность между верхними уровнями протокола и физической передачи отдельных пакетов. На этом уровне, сообщения представлены в виде сегментов, которые будут разделены на более мелкие единицы (например, пакеты). Среди услуг, предоставляемых на этом уровне, являются надежность и управление потоком. Для надежности, этот слой должен гарантировать, что пакет получен, и получен без ошибок. Детали для обработки надежности описаны ниже. Управление потоком происходит, когда два устройства взаимодействуют друг с другом, через СМИ общаются с разной скоростью, например, когда два компьютера имеют разные скорости модема.

Для обработки надежности, слой 4 должен гарантировать, что пакеты теряются, в пути заменяются. Существуют различные механизмы для обработки этого. Простой подход штамповать каждый пакет его порядковый номер в общем сообщении, такие как 4 7. Приемное устройство ожидает семь пакетов, чтобы прибыть. Если все пакеты прибывают, кроме номера 4, передающее устройство должно его повторную передачу. Кроме того, каждый пакет будет содержать информацию обнаружения ошибок, таких как контрольные суммы. Контрольная сумма представляет собой вычисление на основе двоичных значений, что составляет сообщение. Одно простое вычисление для контрольной суммы сложить число 1 бит в сообщении . Например, если бинарное сообщение состоит из 256 битов, и 104 из них являются 1s, а остальные 152 0s, то контрольная сумма будет 104. Тем не менее, как правило, контрольная сумма должна быть установлена в размере. Для достижения этой цели, функция контрольной суммы может сложить ряде 1 бит, а затем разделить это на заранее выбранное значение. Контрольная сумма становится остаток от деления (это мод, или по модулю, оператор). Идея использования мод известна как хэш-функция. Есть целый ряд различных алгоритмов контрольных сумм, включая отпечатки пальцев, рандомизации функций и криптографических функций. В любом случае, после получения пакета, слой 4 определяет, является ли какой-либо из данных в пакете ошибочным путем сравнения значения контрольной суммы с данными в пакете. Если обнаружена ошибка, слой 4 посылает

запрос таким образом, что пакет может быть повторно. 4-го уровня реализации включают TCP, UDP (оба из них обсуждаются вместе с TCP / IP ниже) и SCTP (Stream Protocol управления передачей).

Слой 5 является уровень сеанса. Этот слой обеспечивает соединение между двумя устройствами. Когда два устройства обмениваются данными, они сначала установить сеанс связи. Сессия остается открытым до тех пор, пока устройства разорвать соединение. Между установления и завершения сеанса, сеанс должен быть сохранен. Кроме того, сеанс, который преждевременно обрываться, может быть восстановлен в этом слое. Именно этот слой, который обрабатывает эти задачи (ИНГ заведениями, поддержание, восстановление, завершение). Слой 5 реализаций включают NetBIOS, SAP (протокол Объявление сессии), PPTP (к-Point Tunneling Protocol) и SOCKS (Socket Secure).

Слой 6 является уровень представления. Этот слой отвечает за перевод сообщения из данного приложения, которое генерирующий сообщение, в форме синтаксиса, требуемого нижних слоев. Из-за этого, слой 6 иногда называют синтаксическим уровнем. Именно в этом слое, что первоначальное представление сообщения преобразуется в единое представление. Например, в языке программирования Си, строки завершаются специальным символом \ 0. На этом слое, любые \ 0 символы могут быть удалены из сообщения. Другим примером может служить для иерархически структурированных данных, таких как данные в XML-представлении. Такие данные должны быть преобразованы в плоский формат. Шифрование (для исходящих сообщений) и дешифрование (для входящих сообщений) происходит на этом уровне. Слой 6 реализаций включает SSL (Secure Sockets Layer), TLS (Transport Layer Security) и MIME (Multipurpose Internet Media Extensions).

Слой 7, самый высокий слой, представляет собой слой приложения. На этом уровне, конечный пользователь или прикладная программа создает сообщение, которое будет передаваться, или в этом уровне, прикладная программа представляет принятое сообщение для конечного пользователя. Этот слой включает в себя различные сетевые коммуникационные программы, такие как Telnet; FTP; электронные почтовые протоколы, такие как POP, SMTP и IMAP; и поддержки сетевых услуг, таких как системы доменных имен (DNS).

Модель OSI работает следующим образом. Сообщение создается через какое-то приложение программным обеспечением на уровне 7. Это исходное сообщение считается данные, которые должны быть переданы. Слой 7 прикрепляет заголовок к передней части сообщения и отображает

сообщение на слой 6. На слое 6, особый заголовок также добавляется в сообщение. Сообщение, в настоящее время, состоящее из исходных данных и двух заголовков, отображается в слой 5, а другой заголовок прикреплен к передней части сообщения. Верхние три слоя работают на сообщении в целом, а не сегментированных блоков, работающих, начиная со слоя 4 . На слое 4, сообщение не делится на пакеты, а затем на более мелкие единицы, пока, на уровне 1, сообщение рассматривается, как отдельные биты. Слои 4 и 3 также проставляют информацию заголовка и слой 2 добавляет и заголовок и информацию нижней сноски. И, наконец, на уровне 1, передается сообщение. Полученное сообщение аналогично отображается вверх через каждый слой, удаление заголовка (и нижний), как не нужно, пока сообщение поступает в верхний слой и доставлен в прикладное программное обеспечение. Рисунок 12.12 иллюстрация процесса отображения в OSI.

TCP / IP является другой, обычно используется протокол передачи данных. TCP / IP включает в себя два отдельных протокола , которые были объединены в начале развития Интернета. TCP обрабатывает надежность и упорядоченную доставку пакетов. Он работает на более высоких уровнях стека протоколов, сохраняя детали для того, как пакеты передаются в части IP. Хотя TCP широко используется для связи через Интернет, некоторые приложения используют протокол передачи дейтаграмм пользователя (UDP) на своем месте. Оба TCP и UDP сидят поверх IP, Интернет-протокола. Здесь мы кратко рассмотрим слои TCP / IP и сравнить их слоев, найденных в OSI. В отличие от OSI, слои TCP / IP не столь запретны . То есть они описывают то, что происходит в этих слоях, но они не предлагают подробную информацию о том, как эти действия должны иметь место. Таким образом, существует множество различных способов реализации стека TCP / IP.

Самый низкий слой IP называется Link Layer. Link Layer соответствует непосредственно на двух нижних уровнях OSI. Этот слой выполняет все услуги в физических и данных слоев связи. Второй слой IP является Интернет Layer. Этот слой отвечает за передачу пакетов данных через одну или более сетей. На этом уровне, адресации и маршрутизации происходит. Именно на этом уровне, что IP-адреса используется. TCP / IP слой 2 подобен слою OSI 3. Тем не менее, OSI позволяет связь между устройствами в той же сети в слое 2, в то время как все маршрутизации происходит на уровне 2 TCP / IP. Есть и другие существенные различия между OSI Layer 3 и TCP / IP слоя 2, но они здесь не рассматриваются.

Следует отметить, что адресации и маршрутизации, упомянутые

здесь с IP версии 4 (IPv4), текущей и более популярной версии IP. Тем не менее, из-за ограничения на количество IP-адресов, а также из-за интереса к улучшению Интернета, IP-версия ПО 6 стала доступной. IPv6 сделает существенные различия в использовании Интернета, так как спрос увеличивается в течение следующего десятилетия. Одна из самых больших различий между IPv4 и IPv6 является форма адресов. В IPv4, машинные адреса состоят из 32 бит, разделенных на 4 октета. Каждый октет составляет 8 бит (1 байт) и, таким образом, может хранить число в диапазоне от 0 до 255. Октета IP-адреса, разделенная точками. Например, IP-адрес может быть 127.31.49.6.

Есть три класса сетей по отношению к IP-адресации (практика классов ИНГ сетей было прекращено в 1993 году, хотя сетевые адреса часто по-прежнему присвоен этот путь). Класс сети определяет, какие биты четыре октета указывают сеть, на которой расположен компьютер, а остальные биты обозначают адрес компьютера в своей сети. Класс A сети используют только первые 7 битов первого октета для обозначения сетевого адреса. Это ограничивает количество сетей класса A до 128, но оставляет 25 бит для адресов в сети и, таким образом, класса A сети могут иметь до 16 млн. (более 16 миллионов) внутренних адресов. Класс B сети обозначаются с первого октета адреса 128-191, и используют

Рис. 12.12
OSI модель отображения слой за слоем.

Перевод:
Data - данные

первые два октета для идентификации сети. Это оставляет 16 бит для сетевых адресов, или 65536. сетей класса C, наиболее распространенный, используют первые три октета адрес, позволяющий для миллионов сетей класса C. Тем не менее, класс C сетью, то есть только один октет для адреса каждого устройства в сети. Таким образом, сеть класса C может содержать только до 256 адресных устройств. Таблицу 12.2 для деталей. Обратите внимание на то, что два класса зарезервированы значения первого октета адреса, но не использовались до настоящего времени.
IPv4-адреса имеют длину 32 бита. Это обеспечивает 232 различных (разных) адресов, или около 4 млрд. уникальных IP-адресов. Хотя это выглядит как большое количество, ограничение создало проблему в том, что мы достигли этого предела из-за ручного терминала устройств (например, сотовые телефоны), подключенных к Интернету. Кроме того, многие из адресов, которые обычно доступны не были использованы. Например, класс A сети могут иметь достаточное количество адресов для более чем 16 миллионов внутренних устройств, но это вовсе не означает, что каждый класс сеть использует все доступные адреса. С другой стороны, организация может быть предоставлен сетевой адрес класса C, но может потребоваться более чем 256 различных адресов.
В IPv6-адреса 128 бит в длину и часто отображается с использованием шестнадцатеричных обозначений . Преимущество более длинного адреса является то, что он обеспечивает больше, чем 2128 различных IP-адресов. Это дает нам много адресов на очень, очень долгое время, даже если мы обеспечиваем другой адрес для каждого процессора на планете (в том числе в сотовых телефонах, датчиков и других электронных устройств). 128-битный адрес, как правило, состоит из двух частей, 64-битного сетевого адреса префикса, используемого для маршрутизации через Интернет, а также 64-битный идентификатор интерфейса для обозначения хоста внутри сети. Примером может быть адрес 1234: 5678: 90ab: CDEF: 1234: 5678: 90ab: CDEF. Обратите внимание на использование шестнадцатеричный в адресе, а не бинарный или десятичный , как мы обычно просматриваем адреса IPv4.

TCP, верхние слои TCP / IP, также состоит из двух слоев: Transport Layer и на прикладном уровне. Транспортный уровень похож на транспортном уровне ИОО (OSI Layer 4). Тем не менее, протокол TCP допускает две различные формы потоков данных, использующие протокол TCP и те, которые используют UDP.

Таблица 12.2 Интернет Сетевые классы

Класс	Октеты для сетевых Адресов	Октеты для Адреса в сети	Количество Адресов для сети	Юридические Адреса (1ых Октетов)	Комментарии
A	1	3	16,777,216	0–127	Многие адреса больше не используются
B	2	2	65,536	128–191	
C	3	1	256	192–223	
D	Неопределено	Неопределено	Неопределено	224–239	Только мультикастовые адреса

UDP не обеспечивает надежную форму коммуникации, что TCP делает. Когда пакет UDP удаляется во время передачи, не существует никаких усилий, чтобы отправить его. Хотя это звучит как негативно , это может быть выгодно, когда общение в реальном времени является более важным, чем надежность данных. В качестве примера, в режиме реального времени передачи звука, можно использовать UDP вместо TCP. Обоснованием этого является то, что получатель не хотел бы, чтобы звуковой сигнал прерывался, в то время как пакет повторяется. Упущение одного (или даже несколько пакетов), почти наверняка не препятствует способности получателя, чтобы понять звуковой сигнал.
Рисунок 12.13 иллюстрирует содержимое пакета TCP против пакета UDP. Пакет UDP является гораздо более кратким, так как ему не хватает такой информации, как порядковый номер, подтверждение информации, управления флагами , чтобы указать, каким образом пакет должен быть обработан, и данные смещения, указывая размер поля данных. Срочный указатель не является обязательным в пакете TCP, но может быть полезно для обозначения последнего байта данных, актуальных в поле данных.
Самый высокий слой TCP / IP является уровень приложений, что примерно синоним трех верхних слоев ИОО. В случае сессии уровня BOC, где соединение

сохраняется до окончания, нет ничего точно эквивалентного в TCP / IP. Тем не менее, подобные возможности обрабатываются в транспортном уровне TCP / IP.

Выставление четырех слоев TCP / IP вместе, мы видим большую часть той же функциональности , как OSI. Как и OSI, сообщение передается или получено преобразуется между слоями. Рисунок 12.14 дает пример того, как пакет конвертируется слой за слоем в TCP / IP, где каждый нижний слой добавляет свой заголовок к предыдущему слою и Link Layer добавляет также нижние колонтитулы.

Напомним, что оба OSI и TCP / IP протоколы. Они диктуют некоторые

Рисунок 12.13
Стандартный TCP пакет (а) и пакет UDP (б).

аспекты сети связи, хотя они оставляют другие детали реализаторам. Оба они требуют, чтобы протоколы использовались для перевода сообщения из одного формата (сгенерированные некоторым программным обеспечением применения катиона) в формат, который может быть передан, а затем переведены обратно из переданного формы в формат, что целевое приложение программного обеспечения может обрабатывать. Вернитесь к рисункам 12.12 и 12.14 для отображения OSI и TCP / IP, соответственно. Как вы могли заметить, отображения происходят сверху вниз и снизу вверх. По мере того, как сообщение перемещается вниз по протоколу, все больше и больше деталей будут добавлены, такие как заголовки, ошибки регистрации по прибытию и адресации информации. Как сообщение перемещается вверх протокола , эти добавленные детали сняли сообщения.

Еще один аспект записки для TCP / IP являются сети рукопожатия. На самом деле, всякий раз, когда два компьютера обмениваются данными по сети, до того происходит любое сообщение, два устройства должны выполнить сетевое рукопожатие. По сути, это означает, что первая машина контактирует со вторым сообщением, которое указывает "Я хочу общаться с вами", а затем его ждет. Когда вторая машина готова, он отвечает: "Я готов". После того, как квитирующий сделан , машины могут свободно вступать в контакт, пока связь не завершается в одном конце. TCP / IP было введено понятие трехходовым рукопожатием. При этом первый аппарат посылает пакет синхронизации (SYN), на вторую машину. Вторая машина, если таковая имеется, отвечает обратно с синхронизацией и подтверждением приема пакета (SYN /

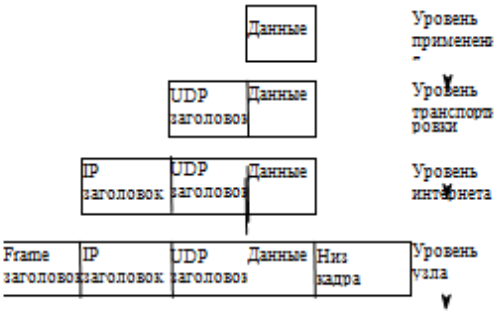


Рис. 12.14. TCP / IP формирования пакетов.

Рис. 12.14.
TCP / IP формирования пакетов.

ACK). И, наконец, первая машина реагирует на второе подтверждение приема пакета своего собственного (ACK). В этот момент две машины синхронизированы и готовы к работе, пока сообщение не будет прекращено. Возвращаясь к рисунку 12.13, вы можете увидеть "квитирование номеров " в пакете TCP. Это используется для поддержки трехстороннего рукопожатия .

Одна концепция не обсуждается в нашем рассмотрении, TCP / IP является то, что из порта. TCP и UDP сообщения требуют не только адреса назначения, но и адреса портов. Порт является лишь показателем предполагаемого программного обеспечения назначения для сообщения. Таким образом, при получении сообщения на машине назначения, операционная система может определить, как обрабатывать сообщение по адресу порта при условии. TCP / IP определяет, как это сообщение должно быть разложено, но не прикладное программное обеспечение, которое должно справиться с этим. Порты предназначены для различных целей. Например, общий порт, используемый для SMPT электронной почты, является 25, а также общие порты для HTTP, используемые для просмотра веб-страниц являются 80 и 8080. Есть номера портов, посвященные FTP (20), Telnet (23), SSH (22), и протокол HTTPS (443).

Адреса портов служат не только ОС, но также может быть использовано программное обеспечение для защиты. Например, брандмауэр может запретить определенные адреса портов, таких как 23 (так как Telnet не является безопасным). Точно так же веб-сервер может игнорировать сообщения, поступающие из других, чем ожидалось портов (например, 80, 8080 и 443). Хотя мы ограничим наше обсуждение портов здесь, вы, без сомнения, получите опыт работы с ними, как вам в дальнейшем исследовать его.

Мы закончим этот раздел один TCP / IP теме, связанной с сетевым адресом трансляции (NAT). NAT является преобразование одного IP- адреса на другой. Basic NAT просто использует один IP- адрес в качестве псевдонима для другого. То есть существует отображение один к одному из одного адреса на другой. Эта форма перевода может быть полезной , если две сети с несовместимыми сетевыми адресами пытается взаимодействовать друг с другом. В базовой NAT, процесс перевода должен изменить IP-адреса сообщения, заголовок IP, а также любую контрольную сумму сообщения, которое было вычисляемо как содержание сообщения и информацию заголовка.

Тем не менее, она является гораздо более распространенной , чтобы использовать NAT, чтобы скрыть целую серию IP-адресов. Например,

организация может не иметь достаточного количества IP-адресов для всех своих внутренних устройств. В этом случае, только некоторые из машин в локальной сети приведены фактические IP-адреса, которые могут быть "видны" извне. Другие устройства скрыты от взгляда извне. При поступлении сообщения, адрес получателя должен быть преобразован во внутренний, частный IP-адрес. Эта форма называется NAT много-к-одному, или перегрузка NAT или IP-маскировкой. Сервер должен быть в состоянии определить, какой частный IP-адрес должен быть использован вместо внешнего адреса. Перевод на частный IP-адрес требует дополнительных модификаций, так как пакет должен быть проложен правильно на правильный внутренний маршрутизатор. Среди модификаций, возможно или даже вероятно, что адреса портов будут изменены, а также IP-адреса. Преимущества многие-к-одному NAT является то, что они предлагают степень защиты из-за секретности внутренних адресов, и разрешить локальные сети, чтобы иметь дополнительные IP-адреса. Основным недостатком многих к одному NAT является сложность, он вводит на сервере, который обрабатывает подключение к Интернету.

В этом разделе мы рассмотрим общие формы сетевого программного обеспечения. Это программы, которые мы могли бы использовать для обмена данными между устройствами по сети. Они все используются в TCP / IP интрасетях, так что это не обязательно ограничивается только использованием Интернета, но также может обеспечить обслуживание в пределах локальной сети.

Telnet - это программа, которая позволяет человеку войти в другой компьютер по сети. Человек, выполняющий операцию телнет, необходимо иметь учетную запись на пульте дистанционного управления (хост) компьютера. Разновидностью телнета называется Rlogin, доступный в Linux. С Rlogin, пользователь не должен фактически войти в целевом компьютере, потому что информация об учетной записи пользователя (имя, пароль) будет одинаковым на всех компьютерах сети. Linux, на самом деле, имеет несколько "R-коммунальных услуг ", где "R" обозначает "пульт дистанционного управления." Кроме удаленного входа , есть также RFile, RSH, rstatus и rwho назвать несколько. Программа RFile позволяет выполнять различные операции управления файлами удаленно. Программа RSH открывает оболочку на другую машину удаленно. Программа rstatus содержит основную информацию о состоянии приемника. Rwho программа выполняет Linux, которая выполняет команду на удаленной машине, показывая вам, кто вошел в

Морриса Интернет «Червь»

2 ноября 1988 года аспирант Корнельского университета, Роберт Моррис, выпустил программу в Интернете, предназначенную для демонстрации дыр в безопасности в Unix. Теперь известный как Моррис "интернет-червь", в течение трех дней программа проникла примерно в 6000 машин Unix, в результате чего значительную часть скорости Интернета «зависло», или упала вниз полностью (согласно оценкам, насчитывалось около 60 000 всех компьютеров в Интернете, так что червь повлиял на 10% от всего Интернета). «Червь» попытался получить вход в систему Unix с помощью четырех подходов:

• Подбор слабых паролей (в том числе один пароль на всех)
• использование известных недостатков в Sendmail программы Unix, такие как переполнение буфера
• использование известных недостатков в команде Unix методом подбора
• Использование R-утилиты для распространения на другие компьютеры в сети.

После того, как «червь» попал в машину, он будет загружать полную версию себя и делать копии для отправок на другие сетевые машины, машины, перечисленные в таблице / и т.д. / хост текущей машины, и другие машины через те же четыре подхода, как указано выше.
В сущности, «червь» действовал как отрицание сервисной программы в том, что он сделал так много копий себе, что зараженные компьютеры могли бы сделать еще больше, чем запускать копии «червя».
За свои усилия, Моррис заплатил $10000 в виде штрафа и 400 часов общественных работ. Стоимость повреждения червя оценивается в пределах $100000 и $10 млн. После инцидента с «червем» создана DARPA (Агентство передовых оборонных исследовательских проектов) - группа, которая будет заниматься чрезвычайными ситуациями в интернете и угрозами в будущем.

систему. Идея r -коммунальных услуг является то, что они работают в локальной сети компьютеров, которые совместно используют один и тот же журнал в сервере. R-утилиты не требуют, чтобы вы вошли в удаленный компьютер до тех пор, пока вы не вошли в свой локальный компьютер. Тем не менее, r -коммунальные услуги должны быть установлены в явном виде для использования системным администратором.
Telnet и Rlogin доля проблемы; ни одна из программ не является безопасной . Сообщения, передаваемые между локальными и удаленными компьютерами в текстовом формате. Это включает в себя, в случае Telnet, в какие-либо пароли, посланные от локального к удаленному компьютеру. Поэтому возможно, что сообщения могут быть выбраны третьей стороной, которая может затем просмотреть пароль или другой материал, чувствительный. Полезность SSH это программа, Linux, которая выполняет шифрование и дешифрование автоматически на любое сообщение, отправленное, обеспечивая тем самым безопасную форму общения, в отличие от Telnet и Rlogin. Хотя SSH является частью Linux, она также доступна через программу замазки для Windows.

Пинг это программа, которая будет генерировать, и передавать пакеты повторно к устройству назначения, вывода сообщения подтверждения. Это позволяет вам знать (1), если ваша машина может получить доступ к сети и (2), если удаленный компьютер доступен. Вы можете также использовать пинг, чтобы увидеть, как долго сообщения и ответ принять, чтобы определить, есть у вас какая-то необычная задержка в сети .
Аналогичная программа для пинг трассировка . Эта программа выводит маршрут, между вашим компьютером и компьютером назначения. Вы можете использовать трассировку, чтобы увидеть, как сообщение (пакет) делает его через Интернет, сколько времени он принимает, и если есть маршрутизаторы по пути, которые вызывают проблемы.
Оба пинга и трассировка часто считаются небезопасными программами . Умные хакеры могут использовать один или оба, чтобы исследовать компьютеры в локальной сети. Таким образом, хакеры могут получить IP-адреса, которые не могли бы быть достоянием общественности, и использовать эти IP-адреса, чтобы организовать атаку. Поэтому, вполне возможно, что система или сетевой администратор может отключить обе эти программы .
HTTP является основой для веб-браузера коммуникации. Запрос HTTP отправляется на веб-сервер, приводит к странице, возвращается и отображается в браузере. Тем не менее, могут быть случаи, когда вы

хотите, чтобы загрузить файл быстро и без отображения его в браузере. Вы можете сделать это из командной строки в Linux с помощью программы Wget. В Wget, необходимо указать полный URL-адрес (имя сервера, путь к каталогу на сервере, имя файла). Затем этот файл загружается в текущий каталог.

Вы также можете загрузить файлы непосредственно с помощью FTP-программы. Тем не менее, в FTP, сеанс создается между вами и сервером, который остается открытым до тех пор, как вы не завершите сеанс, или тайм-аута сеанса. Хотя большинство FTP программ являются графическими, есть также версия командной строки. Графические версии FTP включают FileZilla, Win-Ftp и Win- SCP (который включает в себя безопасную форму FTP, подобно тому, как SSH является безопасной формой Telnet). При использовании FTP для доступа к файлам на удаленном компьютере, вы должны либо иметь учетную запись на удаленном компьютере, или вы должны войти в систему как анонимный пользователь. Анонимные пользователи часто имеют доступ только к публичным файлам на сервере FTP.

Одна большая разница между HTTP и FTP является URL. Вы обычно отправляете запрос HTTP, нажав на ссылку в HTML - документа в веб-браузере. Ссылка кодирует местоположение документа, который вы хотите получить, так что вы не должны запоминать такую информацию. В FTP, однако, вы должны знать адрес сервера, а затем вы должны либо знать путь к каталогу и имя файла, или вы должны искать вокруг на сервере, чтобы найти то, что вам интересно . В обоих HTTP и FTP, то адрес сервера должен быть IP-адрес устройства. Однако, поскольку IP-адреса трудно запомнить, мы изобрели ярлык, в котором вы можете использовать IP-псевдоним аппарата вместо этого. Псевдоним является английским , как описание (или аббревиатура) машины назначения. Например, у нас есть такие псевдонимы как www.google.com для предприятий, www.nku.edu для университетов и www.usa.gov для правительственных сайтов.

Маршрутизаторы могут работать только с IP-адресов, а не IP-псевдонимы, поэтому мы должны обеспечить форму трансляции от псевдонима адреса. Это обрабатывается DNSS. Сервер DNS является всего лишь компьютером , работает программа DNS и файлы данных, которые предоставляют информацию отображения. Кроме того, DNS имеет указатели на одну или более DNSS в Интернете, так что, если локальный DNS-делает нет информации отображения, он может передать запрос другому DNS. DNSS расположены в иерархическом порядке через Интернет, таким образом, что запрос может быть передан вместе

с одного DNS в другой к другому к другому до другого , отображение может быть получено в конце .
Программа Nslookup обеспечивает перевод с IP-псевдонима IP-адреса для вас. Это удобно, если вы хотите, чтобы убедиться, что конкретный псевдоним был установлен в таблице DNS. Это также полезно, если вы хотите знать, физический IP-адрес компьютера, зная только его псевдоним. Если вы хотите узнать IP-адрес вашего собственного компьютера, используйте Ifconfig (в Linux) или IPCONFIG (в Windows).
Три другие команды сети Linux являются агры, используются для определения подключения к Ethernet и сетевой карты реагирования; NetStat используется для определения состояния подключения к сети и состояния сокета; и маршрут, в котором перечислены таблицы маршрутизации (включая такие вещи, как сети IP-адрес шлюза работы и IP-адрес локальной сети и маски). Также стоит обратить внимание, в Linux является команда сетевой службы. Вы можете запустить или остановить службу сети или проверить ее состояние. Команда выглядит следующим образом:

/ SBIN / услуга командной сети

Где команда вниз, запуск, перезапуск, статус. Услуга сети это демон, который должен быть запущен для вас, чтобы иметь возможность использовать сеть (отправлять сообщения или получать сообщения).
Linux имеет множество файлов, связанных с сетью, которые стоит исследовать. Некоторые из них перечислены ниже.

• / и т.д. / хосты: IP-псевдоним для отображения адресов для машин, что ваша машина будет часто общаться . Размещая IP-адреса в этом файле, ваша машина не должна связываться с DNS, во-первых, экономия времени.
• /etc/resolv.conf: хранит адрес вашего DNS. Там может быть несколько записей.
• / и т.д. / sysconfig / сети-скрипты /: конфигурационные файлы хранятся, которые запускаются в процессе инициализации системы Linux для настройки сетевых подключений.
• /etc/xinetd.d/: содержит услуги, которые полагаются на сетевой демон, XINETD.
• /etc/hosts.allow и / и т.д. / hosts.deny: разрешать или ограничивать доступ к вашему компьютеру.

• /etc/hosts.equiv: содержит IP-адреса "доверенных" машин, так что r -коммунальное хозяйство будет работать на всех компьютерах, определенных как эквивалент.

Размещение вашего компьютера в сети приглашает хакеров атаковать ваш компьютер. Это, пожалуй, одна из наиболее уязвимых аспектов в современной вычислительной технике . Конечная безопасность, чтобы не подключать ваш компьютер к сети. Естественно, это резко ограничивает возможности компьютера и самостоятельно. Таким образом, программное обеспечение сетевой безопасности доступно . Такое программное обеспечение не только полезно, но в эти дни важно для тех, кто хочет получить доступ к Интернету безопасно. Два из наиболее распространенных сетевых программных обеспечений безопасности являются брандмауэры и противовирусное программное обеспечение.

Брандмауэр может быть либо программного, или аппаратного обеспечения (в последнем случае, это выделенный сервер, который работает программным обеспечением брандмауэра). Программное обеспечение брандмауэра содержит список правил, которые описывают типы сообщений, которые либо должны быть позволены сделать это через брандмауэр и к компьютеру, и те, которые должны быть заблокированы. Правила могут быть основаны на порте, ряде сообщения, тип сообщения, IP-адрес источника сообщения, является ли сообщение в TCP или UDP (или другого) формата и протокола (например, HTTP, HTTPS, FTP). Мы кратко рассмотрели брандмауэр Linux в главе 11.

Антивирусная программа пытается определить, если файл содержит вирус, или в более общем плане, некоторые формы вредоносного ПО. Антивирусная программа может быть запущена по требованию, или вы можете настроить его так, чтобы все входящие сообщения проверялись . К сожалению, противовирусное программное обеспечение не обязательно ловит каждый кусок вредоносного программного обеспечения. Как программисты становятся более изобретательными в том, чтобы атаковать ничего не подозревающего пользователя, противовирусное программное обеспечение должно стать более сложным. Но антипрограммное обеспечение всегда отстает от цвета RAL за нововведения нападавших. Кроме того, вы часто должны быть прилежными в обновлении программного обеспечения противовирусной (новые релизы могут выходить ежедневно или еженедельно).

Сети могут иметь свои собственные формы безопасности, такие как

обнаружение вторжений и интрузива программного обеспечения профилактики Сьон. Там нет общего решения для предотвращения вторжений в этот момент, как все операционные системы имеют дыры в безопасности для эксплуатации. На самом деле, хакеры часто пытаются прощупать оборону собой систему, прежде чем они смонтируют нападение. Некоторые из отверстий, которые хакеры могут найти, включая , оставляя пробелы в брандмауэре (например, порты, которые не проверяются), оставляя беспроводное подключение к незащищенному, вышеприведенному шифрованию, не требующему надежных паролей, отсутствие физической безопасности вокруг точек доступа к сети, и допускающие такие операции, как пинг и трассировка .
Среди наиболее распространенных форм атак, найденных сегодня атак отказа в обслуживании на веб-серверах, IP-спуфинга для перехвата сообщений, отравление ARP для изменения IP-адреса в маршрутизаторе, чтобы быть машиной атакующего, а не локальной машиной , переполнение буфера, что позволяет хакеру вторгнуться в компьютерную память со своими инъекциями кода и SQL, которые позволяют хакерам вторгаться в базу данных со своими командами SQL. Хотя это все известные формы атак, это не обязательно означает, что система защищена от них. И, как известно, эти атаки, новые атаки в настоящее время думали. Любая организация , которая ценит целостность данных и функциональность своих компьютерных систем, потребует уровень операционной системы и сетевой безопасности. Это характерно для средних и крупных организаций нанять администраторов сетевой безопасности, в дополнение к системным администраторам. Мы рассматриваем целостность данных и безопасность в главе 15.
Интернет

IP-адреса сложные для запоминания и поэтому пользователи ссылаются на псевдонимы как www.google.com . Для того чтобы перевести псевдоним фактического IP-адреса, вам необходимо получить доступ к серверу DNS. Каждая сеть либо имеет DNS или знает, где найти DNS. Большая часть этого скрыта от вас, как пользователю, но если вы являетесь администратором сети, вы сами должны настроить DNS или знать где он . Есть дополнительные псевдонимы, чтобы ограничить то, что пользователю необходимо вспомнить. Например, он является общим для последующих псевдонимов веб-сервера, так что пользователю не требуется вводить WWW , и, следовательно, хотя www.google.com это псевдоним веб-поисковиков сервера Google, и получаем google.

com. DNS представляет собой компьютер содержащий таблицу с IP-адресами и псевданимами .При использовании интернет псевдонимов DNS проверяет ввод через свою таблицу DNS затем возвращает соответствующий IP-адрес. IP-адрес, затем отправляется обратно на компьютер.
Компьютерные сети существовали на протяжении десятилетий. В прошлом большинство компьютерных сетей были локальными сетями (LAN), и они были изолированы, вы могли общаться внутри сети, но не за пределами . Все это изменил Интернет . Для общения по Интернету , машина должна запустить протокол TCP / IP. С ростом популярности Интернета, все компьютеры сегодня запускают протокол TCP / IP. Компьютер, подключенный к локальной сети, может также соединяться через Интернет, если локальная сеть подключена к Интернету. Мы размыли линии, где заканчивается одна сеть и начинается другая . Сеть соединяющая компьютеры в одной комнате подсоединены маршрутизатору сетей для соединения компьютеров с другой комнаты . Сеть, которая существует на одном этаже, подключен к сети на другом этаже. Сети одного здания подключены к сети в других зданиях. LAN может рассматриваться как любая сеть из этих сетей, охватывающая весь университетский городок. С помощью VPN, сеть может расшириться, чтобы включить тех, кто вошел в систему удаленно с других сайтов. Сеть организации, скорее всего также подключена к Интернету. Благодаря Интернет провайдерам компьютеры в домах также подключены к Интернету, делая его главной сетью из сетей . Так что, можете воспринимать Интернет как расширением локальной сети LAN вашего домашнего компьютера . Существуют три способа «склеивания» данных компьютеров:

1. Коммутация пакетов. Телефонная сеть с коммутацией каналов используется, когда вы набираете номер, путь устанавливается между источником телефона и телефона получателя и путь остается фиксированным и определяемым , до отключения одного телефона . Мобильные телефонные сети, передающие голосовое звучание (но не данные), такие же , хотя схема может меняться по мере продвижения, так как направляются в ближайший доступные мобильные телефонные башни. Коммутация пакетов, с другой стороны, разрывает связь на небольшие пакеты данных, и каждый из них передается по сети отдельно . Каждый пакет потенциально может принять другой путь, чтобы добраться до места назначения. Пакетная коммутация была впервые

использована в 1960-е годы, когда Интернет был впервые разработан .
2. Маршрутизаторы. Маршрутизатор представляет собой устройство, которое "на лету" отправляет сообщение по сети . Маршрутизатор является устройством, которое принимает сообщение, анализирует IP-адрес назначения, и посылает сообщение по следующему местоположению в сети, к другому маршрутизатору. В сети назначения, маршрутизаторы создают путь сообщений, до достижения LAN, и, наконец, маршрутизатор локальной сети или коммутатор отправляет сообщение, на конечный компьютер.
3. IP-адреса. Часть протокола TCP / IP формирует правила адресации . IP-адрес состоит из четырех чисел, где каждое число в диапазоне от 0 до 255. Например, 10.11.241.105 является IP-адресом. Каждое число на самом деле хранится в 1 байт памяти (1 байт составляет 8 бит, а 8 бит может хранить любое число от 0 до 255). Таким образом, объем IP-адреса 4 байта или 32 бита. Так как IP-адреса трудно запомнить, мы часто ссылаемся на IP-псевдонимы, как англоязычное наименование (например, www.google.com). Для того, чтобы перевести от псевдонима IP на IP-адрес, мы используем DNS. Как упоминалось ранее, из-за ограничений в 32-битовые адреса, IPv6 использует 128 бита для IP-адресов .
Три класса IP-адресов, используемых в настоящее время диктуют, понимание 4 октетных адресов. В классе A IP-адреса, первый октет является идентификатором сети, а остальная часть октета составляют номер компьютера. Класс B адреса разделить адрес в сети ID для первых двух октетов и количества машин в последние два октета. Класс C-адреса используют первые три октета для сетевого адреса, оставляя один октет (256) для числа машин. Определение класса сетевой части адреса можно еще больше разделить через процесс называемого как формирование подсетей .
Возможно, вы знакомы с термином маска подсети. Что такое маска (или маска подсети)? Мы часто вычисляем значение в компьютере с помощью операции AND или OR по двоичным разрядам. Например, два байта 10101010 и 00001111, когда вместе дают операции AND 00001010 и когда OR вместе дают 10101111. Эти операции работают побитно, то есть, мы выполняем операции AND или OR на первый бит обоих чисел, чтобы получить новый первый бит (1 и 0 равен 0, 1 или 0 1). Мы делаем это со всеми восьми парами битов. AND операция возвращает 1, если оба бита, равны 1 (0 в противном случае) и операция OR возвращается к 1, если даже 1бит (0 в противном случае).

И так, почему мы используем их? Представьте себе, что у вас есть следующий IP-адрес в двоичном разряде:

00001010.00001011.11110001.01101001 (10.11.241.105)

Если AND это с 11111111.11111111.11111111.00000000, что мы получаем? Мы получаем

00001010.00001011.11110001.00000000

Что представляет собой это значение? Мы начали с 10.11.241.105 и результат операции AND дает нам 10.11.241.0. Для сети класса C, что мы получили это IP-адрес сети, где находится наш компьютер. С другой стороны, если AND с количеством 00000000.00000000.00000000.11111111, мы получаем

00000000.00000000.00000000.01101001

Это значение, 0.0.0.105, является конкретным адресом нашего компьютера в сети.
Битовые модели 00000000 и 11111111 являются целыми числами от 0 до 255, соответственно. Мы используем значения 255.255.255.0 и 255.0.0.0 как сетевые маски. Две комбинации, которые мы не используем 0.0.0.0 и 255.255.255.255, потому что такие маски сети вернут 0.0.0.0 или весь исходный IP-адрес и не предоставит нам полезную информацию. Маска подсети используется, чтобы легко получить номер сети, и номер подсети используется для получения адреса устройства в сети. Вы также можете увидеть IP-адрес в таком виде: 10.11.241.0/24 или 10.0.0.0/8. Это всего лишь еще ожин способ определения маска подсети. Число после / относится к числу двоичных 1-х в маске. Таким образом, маска 255.255.0.0 также может упоминаться как / 16 .

Как работает Интернет
Кратко, Интернет работает следующим образом. Вы хотите отправить сообщение с компьютера на другой компьютер (например, сообщение электронной почты, запрос HTTP для веб-страницы, запрос FTP, SSH, Telnet или связи). Ваш компьютер принимает сообщение и пакеты его на один или несколько пакетов. Пакеты являются относительно небольшими. Если сообщение короткое, оно может поместиться в

одном пакете. В качестве примера, запрос http://www.google.com/ будет коротким, но по электронной почте сообщение может быть длиннее на тысячи миллионов байт, так что он может быть помещен на несколько пакетов. Каждый пакет получает адрес получателя (псевдоним IP), сигнатура (обнаружения ошибок информационной базы, которая часто вычисляется путем суммирования значений байтов нескольких байтов, а затем сжимают ту сумму а затем суммирует суммы) и обратный адрес (IP-адрес вашего компьютера, чтобы конечный компьютер отправил ответ).

Когда пакет собран, дестинация IP-псевдонима переводится в IP-адрес. Ваш компьютер хранит адрес DNS вашего сайта (в Linux, это хранится в /etc/resolv.conf). Ваш компьютер посылает сообщение DNS сайта, чтобы перевести IP псевдоним на IP-адрес. Если локальный DNS не распознает псевдоним IP, то запрос передается на другой DNS. DNS в Интернете по иерархии. В конечном счете, DNS будет распознавать псевдоним, и реагировать на те с адресом. Каждый принимающий DNS будет обновлять свои собственные таблицы (при необходимости). Тем не менее, если DNS на самом высоком уровне иерархии не распознает псевдоним, и возвращает сообщение об ошибке, и ваше сообщение не может уйти с вашего компьютера.

После предоставления IP-адреса посылаются пакеты. Они отправляются с вашего компьютера к интернет-точке вашего сайта присутствия (или шлюза). Сообщение направляется из текущей локальной сети на этот шлюз. Оттуда, сервер отправляет каждый пакет по одному из его путей, которые соединяют его к Интернету. Каждый пакет находит свой собственный путь через Интернет, перейдя от одного маршрутизатора к другому. Каждый маршрутизатор выбирает следующий путь, основанный по адресу назначения и текущего трафика сообщений по различным путям доступа. Пакет может пройти через одну ссылку, две ссылки, или несколько десятков ссылок. Используйте команду исследовать, для определения сколько транзитных шлюзов потребуется для данного сообщения.

После получения сообщения в пункте назначения, IP-адрес назначения может или не может быть фактическим IP-адресом хоста назначения. Если NAT используется на удаленном узле, то IP-адрес переводится с адреса "внешнего публичного" к "внутреннему частному" адресу. А теперь с внутренним частным адресом, сообщение отправляется по внутренней сети к сети назначения и переключается на компьютер получателя. Как правило, существует меньше проводящих путей выбора внутри страны

и на самом деле сообщение может иметь только один маршрут, чтобы взять. После того как, конечный компьютер получил пакет, операционная система компьютера сохраняет его. Пакет отображается в стеке протоколов, и все пакеты одного и того же сообщения объединяется по мере пребывания. Они сортируются согласно последовательности чисел. (например, пакет 3 из 5). После получения всех пакетов, окончательное сообщение представляется соответствующему прикладному программному обеспечению (или хранится на диске для FTP или Wget). Обратите внимание, что упорядочение пакетов не осуществляется IP, а по протоколу TCP более высокого уровня. Пакеты назначения отправляют уведомление, что сообщение было получено в полном объеме и правильно (или случаи когда какие-либо пакеты были сброшены или прибыли с ошибками). Весь процесс занимает всего несколько секунд или меньше (как правило), даже несмотря на то расстояние в 1000 миль, и дюжины количества хопов .

ИНТЕРНЕТ ПРОТИВ ВСЕМИРНОЙ ПАУТИНЫ

Хотя большинство людей используют термины взаимозаменяемо они все разные. Интернет в данной главе - это компьютерная сеть. Это всемирная сеть но, также и сеть – компьютеров и компьютерных ресурсов, что мы используем для общения друг с другом, медиа, которая соединяет все средства воедино (будь то кабель, радио, микроволновые или спутниковые соединения) и устройство вещания, как свитчи и маршрутизаторы. Интернет это – физический объект; он содержит части, которые можно потрогать.

Всемирная паутина это – сборник документов, объединенные гиперссылками. Данные документы хранятся в компьютерах, которые мы называем серверами. И эти серверы среди всех компьютеров объединяются в один Интернет.

Всемирная паутина во главе Интернета. У нас может быть Интернет без web (паутины), но без Интернета не будет никакой паутины.

Следует отметить еще один пункт то, что люди часто комментируют и говорят, что они потеряли Интернет или Интернета нет. На самом деле они потеряли возможность подсоединения к Интернету или возможность общения по Интернету. Но Интернет не может не быть. Он был разработан таким образом, что может пережить ядерную войну и должно случиться, много различных разрушений для его исчезновения.

Краткая история Интернета

В 1968 году четыре научно-исследовательские организации финансировались Министерством обороны США (DOD), для создания сети для связи на дальние дистанции. Эти организации были в Университете штата Юта, Калифорнийский университет в Лос-Анджелесе (UCLA), Калифорнийский университет в Санта-Барбаре и Стэнфордском исследовательском институте (SRI). Их намерение состояло в том, чтобы построить "электронный мозг", имея компьютерную сеть, которая может передавать данные файлов и делает возможным удаленный доступ к компьютерам друг друга. Они улучшили сравнительно недавно предложенной технологии коммутации пакетов идеи на практике, и использовать телефонные сети для подключения этих компьютеров вместе. Эта сеть была названа ARPANET. Первое сообщение было послано между UCLA и SRI 29 октября 1969 года.
Оригинальный протокол, используемый назывался 1822 протокол. Этот протокол был не очень эффективным, и в конечном итоге заменить TCP / IP в 1983. Первое сообщение по электронной почте было отправлено через ARPANET в 1971 г. FTP был добавлен в 1973 году.
Со временем все больше компьютеры добавлялись к сети. К июню 1970 года насчитывалось девять компьютеров в сети. К декабрю 1970 года насчитывалось 13 компьютеров, а к сентябрю 1971 года насчитывалось 18. В период с 1972 по 1973 год, 29 и затем 40 компьютеров были размещены на ARPANET.
Два спутника, предоставленные для поддержки ARPANET и компьютеры в обоих Гавайских островов и Норвегии были добавлены в 1973 году, что делает сеть международной. От 1975, сеть достигла 57 компьютеров. Кроме того, в 1975 году Министерство обороны передало контроль над ARPANET от ARPA Агентства обороны коммуникаций (DOD). В 1983 году военная часть сети (MILNET) отделяли. До этого раскола, было 113 компьютеров и после этого, 68.
В 1974 году в Великобритании разработали свою собственную форму сети с коммутацией пакетов, X.25. X.25 стала общедоступной. Другие сети были созданы в конце 1970-х и начале 1980-х годов. Они, наряду с ARPANET, в итоге стали компонентами Интернета.
В 1979 году два студента Дьюк создали UUCP для обмена сообщениями через электронную почту. Это стало известно как новость Usenet. Usenet была доступна в течении нескольких сетей и люди могли получить доступ к сети с помощью удаленного доступа через модем с домашнего

компьютера или с сетевой машины.
В 1983 году, с созданием протокола TCP / IP, то ARPANET был переименован в Интернет. Различные предыдущие сети, которые были соединены друг с другом стали основой NSF, который стал основой Интернета, так как Интернет продолжает расти. В 1980-х и начале 1990-х, НФС спонсировал инициативу по предоставлению доступа в Интернет во многие университеты США. В то же время, оплачиваемые сети (AOL, Genie, CompuServe и т.д.) стали подключаться Интернету.
С конца 1980-х до середины 1990-х годов, было несколько других событий, которые изменили Интернет. Во-первых, доступ к нему стал доступным для общественности через платные сети, библиотек, школ и университетов. Во-вторых, гипертекстовые документы проложили путь. В-третьих, правительство США приняло законодательство, для облегчения компаниям предоставления услуг Интернета для массового использования. Тем не менее, для многих людей процедура подключения интернета является не удобной.
Примерно в 1994 году была разработана идея веб-браузера. Первый из них был известен как мозаика. Идея веб-браузера, что браузер будет загружать и автоматически отображать документ. Документ будет включать гипертекстовую информацию (ссылки). При нажатии на гипертекстовую ссылку активирует команду для браузера, чтобы отправить запрос HTTP для нового документа, который будет отправлен с сервера. Документ будет отправлен обратно и отображается в браузере. До этого времени, большинство пользователей не понимали , как отправлять сообщения (будь то FTP, Telnet, электронная почта, или иное), включая то, как указать IP-псевдонимы или адреса. Но с веб-браузерами, пользователь только должен был знать, как нажимать на ссылки. Со временем, веб-браузер заменил большинство других методов интернет-общения за исключением общения по электронной почте для обычного пользователя,.
Где мы находимся сегодня? Сотни миллионов компьютеров составляют Интернет. Точное число трудно сказать, потому что пользователи домашних компьютеров не пользуются ими всю жизнь Если учитывать все портативные устройства, то число превышает 4 млрд. Количество фактических пользователей Интернета это предварительная оценка, но между домашних компьютеров и портативных устройств, по оценкам, (по состоянию на 2012 год) более 2 миллиардов пользователей (примерно одна треть населения планеты). Рисунок 12.15 показывает рост, интернет-пользователей с 1995 года.

А веб? Через Интернет есть доступ к триллиону документов. Многие из этих документов являются гипертекстовые (HTML) документы.
Однако большое количество документов, представляют собой файлы данных (текстовые документы, документы в формате PDF, PowerPoint документы и т.д.).
Сегодня управление Интернет в значительной степени в руках корпораций, которые предоставляют доступ в Интернет. К ним относятся многие телекоммуникационные компании благодаря Закону о телекоммуникациях США 1996 года. Этот акт, помимо всего прочего, позволил телефонным компаниям, кабельным компаниям, компаниям мобильной связи, а также другим, чтобы служить в качестве интернет-провайдеров. До этого момента, доступ в Интернет был доступен через работодателей и университетов, либо через WANs оплаты, таких как CompuServe, America On-Line и Prodigy.
Таким образом, телекоммуникационные компании контролируют доступ к сети Интернет. Но кто контролирует сам Интернет? В каком-то отношении все пользователи контролируют Интернет, в том смысле что,

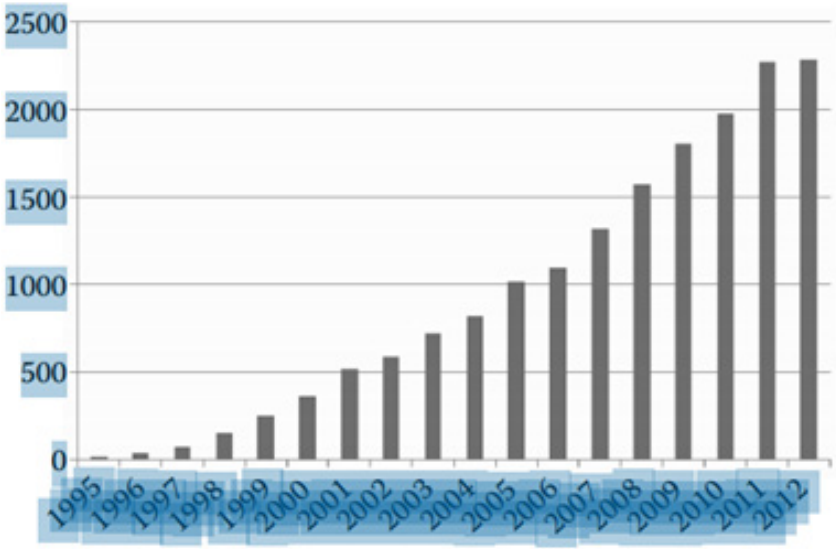


Рисунок 12.15
Количество интернет пользователей с 1995 по 2012 гг.
Количество интернет пользователей (в млн)

ответственны за содержание размещаемых данных в веб-страницы, с помощью электронной почты, которые мы посылаем, и файлов, которые мы загружаем и выгружаем, а также через наше участие на досках объявлений, сайтах, блогах и так далее. Некоторые аспекты Интернета, однако, были переданы различным международным консорциумам.
Одна такая организация известна как Интернет-корпорация по присвоению имен и номеров (ICANN). Эта организация предоставляет IP-адреса и обрабатывает доменное имя (псевдоним) IP-споров. Отдел по ICANN является Интернет Assigned Numbers Authority, который, как следует из их названия, раздает IP-адреса. Эти адреса основаны на географическом местоположении, и к примеру, один тип IP-адреса выдается для сайтов находящихся в Северной Америке, а другой в Европе и Западной Азии.
Другая организация является Консорциум World Wide Web (W3C). Эта организация разрабатывает стандарты для создания веб-сайта, включая следующее поколение языка разметки HTML, использование каскадных таблиц стилей, использование XML и понятий, связанных с семантической сети.

Будущее Интернета

Очевидно, что Интернет оказывает огромное влияние на наше общество. Преимущества, которые получает общество из Интернета, довольно таки огромен для перечисления. Однако, существуют также явные проблемы, связанные с текущим состоянием Интернета. В настоящее время решения данных проблем изучается.

1. Коммутируемый доступ . Многие люди использовали телефонную сеть, чтобы получить доступ к Интернету через поставщиков услуг Интернета. Проблема заключается в том, что телефонная сеть состоит прежде из всей совокупности витой пары проводов, и эта среда может поддерживать только ограниченную пропускную способность трафика сообщений (примерно 56000 бит). Загрузка 1 MB файл (маленькая картинка, например) займет 18 секунд в этой полосе частот. Хотя теперь доступны другие технологии, такие как цифровой кабель и прямые цифровые абонентские линий (DSL) в население людей, вынужденные использовать более медленный интернет. Это было названо как «последней милей технологии», поскольку большая часть телефонных сетей используют волоконно-оптические кабеля кроме последнего

участка между разделением и самого дома. Замена данной последней мили или двух витых проводов является невозможной. Телефонные компании только сейчас постепенно заменяют его. Несмотря на то, что кажется маловероятным, что последней мили технологии воздействует на людей в крупных городах, многие люди, которые живут на окраине города или в сельской местности, в значительной степени должны полагаться на телефонные линии для доступа к сети Интернет. Одним из решений этой проблемы является соединение через спутниковую антенну. Информация грузится очень быстро, так как спутник имеет большую пропускную способность, чем телефонные линии. Тем не менее, загрузка должна быть еще через телефонные линии, так что это лишь частичное решение. Другим решением этой проблемы является использование сотовых телефонов. Тем не менее, мобильные телефоны сети часто переполнены, и поэтому их пропускная способность в конечном счете. будет также ограничена. На самом деле, 3G-сети настроены отбрасывать пакеты, когда проходит слишком много сообщений. Сети 4G обещают решить эту проблему, но охват 4G пятнистый.

2. TCP / IP, казалось бы, предлагает 4 млрд уникальных IP-адресов (32 бита дает 232 или около 4 миллиардов комбинаций адресов). Тем не менее, не все адреса используются. Рассмотрим, например, небольшую организацию, которая дает адрес сети класса C. Они получают 256 различных адресов, но не обязательно использовать их все. Другая организация может потребовать два класса C сетей, поскольку они имеют более 256 адресуемых устройств, но меньше, чем 512. Через NAT, организации может обеспечить большее количество IP-адресов для своих устройств, но это не может распространяться, например, на все портативные устройства, которые в настоящее время используют Интернет. Учитывая, что существует 7 миллиардов человек на планете, мы исчерпали адреса для всех.

3. Интернет-магистраль не была создана для поддержки миллиардов пользователей, особенно когда пользователи запрашивающих большие документы, такие как изображения, фильмы и музыку. В результате чего скорость интернета медленная.

Для решения некоторых проблем, некоторыми сетями используется IPv6 (IP версии 6). Однако IPv6 не может решить все проблемы, и поэтому ученые исследуют технологии для Internet2, новую форму Интернета. Будет ли это использовать текущий источник или требует совершенно новое оборудование, в ближайшие годы будет принято решение.

Наряду с обновлениями инфраструктуры Интернета и протоколов, два дополнительных понятия присутствуют при обсуждении "Web 2.0". Это облачные вычислений и семантическая сеть. В этой главе мы коротко расскажем о них Облачные вычисления сочетает в себе несколько различных технологий: распределенные вычисления, компьютерные сети, сети хранения данных. Идея заключается в том, чтобы предложить, в качестве сервиса, способность разгрузить вычисление и хранение с локального компьютера в облаке. Через облако, пользователи могут получить доступ к ресурсам, которые они обычно не могут иметь доступ, например, увеличение мощности процессора для поддержки поисковых систем, программного обеспечения, так как, они не имеют лицензии на их локальном компьютере, а также хранения большого объема данных локального файлового сервера или компьютера. Концептуально, облако является шагом на пути к повсеместной вычислительной-способности иметь вычисления в любом месте в любое время. По сути дела, облако является конечной в сети клиент-сервер. Клиенты являются пользователями, которые имеют доступ облако удаленно и сервер сам облако, хотя в отличие от традиционной модели клиент-сервер, облако представляет пользование на очень больших масштабах.

Чтобы получить доступ к облаку, обычно интерфейс осуществляется через веб-портал. Таким образом, клиентские компьютеры не нуждаются в каких-либо конкретных программных обеспечений, загруженные на свои компьютеры. В некоторых случаях клиенты загружают специальные приложения. Они также обеспечили безопасный счет (то есть учетная запись, которая требует журнал, чтобы получить доступ). Услуги, предоставляемые облачных вычислений в диапазоне от простого хранения в базу данных и веб-сервисы, электронная почта, игры и балансировки нагрузки для крупномасштабной вычислительной обработки.

Для поддержки облака, компания нуждается в сложной сетевой инфраструктуре. Эта инфраструктура должна работать с очень высокой пропускной способностью для поддержки сотни, тысячи или миллионов пользователей. емкость для хранения данных может потребовать запуска порядка петабайт (следующая величина хранения вверх от терабайт). Если облако для поддержки вычислений, а также хранения, может потребоваться от десятков до сотен высокопроизводительных процессоров. Компании, которые делают облачные сервисы доступными (например, Amazon, Google, Oracle) вкладывают миллиарды долларов в инфраструктуру. В результате, за небольшую плату, в том, что

предприятия и отдельных лиц могут иметь безопасное хранение данных (обеспечить как с точки зрения безопасного доступа и безопасности в виду, того, что их данные копируются регулярно) и удаленный доступ к мощной обработке. Облачные вычисления могут достичь точки, могут заполнится, нам не нужны наши собственные подразделения настольных компьютеров, но вместо этого только интерфейсы (например, портативные устройства), и все хранения и вычисления имеют место в другом месте.

Следует отметить, что вычисление облаком не должен делать Интернет, но для коммерческих и практических целей, большинство облачных вычислений существует как часть Интернета. Если компания должна была ссоздавать свои собственные частные облака, его использование будет ограничено для тех, кто может получить доступ к сети компании. На самом деле, мы обычно видим это с виртуальными частными сетями и экстранет. Идея заключается в том, что компания обеспечивает обработку и хранение, но через VPN, пользователи вычислительных ресурсов компании могут удаленно получить доступ к ним. Таким образом, они несут свои вычисления с ними независимо от того, куда они идут.

Наконец, мы рассмотрим семантическую сеть. World Wide Web представляет собой набор файлов данных, вместе с гиперссылками, и хранится на серверах в Интернете. WWW содержит массу информации, возможно, достигающего все человеческие знаний, т.к. библиотеки доступны через Интернет. Тем не менее, информация в основном плохо индексируются и не организована.

Это вплоть до интеллекта (человека), чтобы просеять информацию и осмысливать ее. Семантическая паутина является стремление во главе с исследователями в области искусственного интеллекта, чтобы организовать знания и предоставить автоматизированные для поиска рассуждений не только через информацию, но найти полезные части представить человека пользователей.

В качестве простого примера, представьте, что вы хотите взять отпуск на Гавайи. Через редактирование разных веб-сайтов и поисковые системы, вы можете забронировать рейс, забронировать отель, арендовать автомобиль, и открыть для себя события и места для изучения на Гавайях. Насколько этот процесс можно автоматизировать? Предположим, что у вас есть представительный бот (часть программного обеспечения, чтобы представлять вас). Вы описали ваши интересы, любит, и не любит. Вы предоставляете свой репрезентативный-бот календарь важных дат, которые ограничивают даты поездок. Можно также указать другие

критерии, такие как ценовые ограничения. Представитель-бот будет использовать эту информацию и общаться с различными веб-сайтами, чтобы забронировать весь отпуск для вас, вплоть до получения карты в местах, вы должны увидеть во время вашего визита. Кроме того, бот будет рассуждать на события, чтобы координировать их так, что время близость, путешествия между различными местами, время дня / вечера, а также другие факторы принимаются во внимание.

Семантический веб-сайт состоит из нескольких различных компонентов. Данные веб-сайта в настоящее время организованы в онтологии: знания баз, которые содержат не только информацию, но и как эта информация может интерпретироваться и использоваться. Небольшие программы, интеллектуальные агенты, строятся для обработки различных задач, таких как планирование, связи и торговли принятию текущих решений. Новые типы языков запросов создаются для облегчения связи между агентами и между агентами и онтологий. В сочетании, можно надеяться, что эти новые технологии будут двигаться Всемирная паутина ближе к видению семантической сети, интернет-ресурсы которые легко доступны, и чьи знания могут быть использованы без чрезмерного человеческого взаимодействия. Семантическая сеть, чьи идеи восходит к 2001 году, только в настоящее время используются в таких областях, как сбор разведывательных данных и медицинских исследований. Это, вероятно, будет еще одно десятилетие или больше, прежде чем мы видим практические, каждый день использует.

ГЛАВА 13.
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Эта глава начинается с введения программных классификаций и терминологии. Тем не менее, основное внимание в этой главе уделяется управления программным обеспечением, в частности установки программного обеспечения в Windows, так и Linux. В Linux, обсуждение концентрируется на использовании менеджеров пакетов и установки программного обеспечения с открытым исходным кодом. Глава завершается рассмотрением серверного программного обеспечения с особым вниманием к установке и конфигураций во время веб-сервер Apache на вводном уровне.
Цели обучения данной главы:
• Обсудить типы программного обеспечения и классификации оригинального и бесплатного программного обеспечения.
• Описать процесс установки программного обеспечения из мастера установки и менеджера пакетов.
• Проиллюстрировать в примере , как устанавливать программное обеспечение с открытым исходным кодом в Linux с помощью конфигурации , рекомпиляции , и процесса установки.
• Опишите функцию популярного серверного программного обеспечения.
• Ознакомить с установкой и конфигурацией веб-сервера Apache .

Здесь мы рассмотрим различные типы программного обеспечения и рассмотрим то, что IT специалисту необходимо знать . Помимо внедрения программных концепций и терминологии, мы сосредоточим внимание на обслуживании и установке программного обеспечения. Мы завершаем эту главу с проверки серверов , установки сервера и технического обслуживания.

Типы программного обеспечения

Термин программное обеспечение относится к программам, которые мы запускаем на компьютере. Мы используем термин программное обеспечение, потому что программы не существует ни в одной

физической, материальной форме. При хранении в памяти, они существуют в виде электрического тока, и когда хранится на диске, они существуют как магнитные заряды. Таким образом, мы различаем программное обеспечение от аппаратного обеспечения, компоненты которого можно указать, или подобрать и потрогать .

Это различие может показаться очевидным ; Тем не менее, это на самом деле не так просто, как это звучит. Наши компьютеры общего назначения, а это означает, что они могут запустить любую программу. Самые ранние компьютеры с 1930-х и 1940-х годов не были для общего назначения, а наоборот, только для определенных программ , таких как интегральные вычисления математического анализа или процессов кода взлома. Сегодня мы видим, что некоторые из наших устройств, хотя и имеют процессоры и память, также специального назначения (например, навигационные системы, mp3-плееры, игровые платформы). Но кроме этих исключений, компьютеры могут запускать любую программу, которая может быть сложной для этой платформы. С другой стороны, все, что можно записать в виде программного обеспечения также могут быть реализованы с использованием схем. Напомним, что на самом низком уровне, компьютер является просто набором AND, OR, NOT, XOR и схемы (вместе с проводами, чтобы перемещать данные). Таким образом, мы могли обойти остальную часть компьютера и осуществить данную программу на уровне схемы. Захватив программу непосредственно в аппаратных средствах, часто называют в качестве встроенного программного обеспечения (или проводной реализации). Аналогичным образом, любая программа, которая может быть встроена, непосредственно в схемы также могут быть написаны на некотором языке программирования и работать на компьютере общего назначения. Эта концепция известна как эквивалентность аппаратного и программного обеспечения. Таким образом, с помощью вызова одного программного обеспечения лица и другое оборудование, мы различаем, как они были реализованы, но не так, как они должны быть реализованы. Как правило, мы захватить наши программы в виде программного обеспечения, а не реализовывать их в качестве аппаратных средств, потому что программное обеспечение является гибким. Мы можем изменить его на более позднем этапе. Кроме того, создание встроенного программного обеспечения зачастую намного дороже, чем эквивалентное программное обеспечение. С другой стороны, выборки выполняют процесс CPU, которое добавляет время на выполнение программы. Программа реализована в прошивке будет выполняться быстрее, возможно, хорошее

Краткая история производительности ПО

1963 – IBM разрабатывает GUAM, первую систему управления данными для использования в NASA.

1963- На основании докторской диссертации, студент MIT разрабатывает Sketchpad, систему компьютерного рисования в реальном времени с использованием световой ручки для ввода данных.

1964 – IBM выпускает MT/ST, первый в текстовый процессор; ввод/вывод данных осуществлялся на клавиатуре Selectric.

Ранние 1970-е – INGRES, первое ПО, выпущенное RelationalTechnology, затем был выпущен Oracle.

1972 – Lexitron и Linolex выпускают машины, имеющие возможность отображать слова текстового процесса на экране.

1976 – ElectricPencil – первый ПК текстовый процессор был выпущен.

1979 – VisiCalc – первая программа электронных таблиц, выпущенная для Apple II.

1979 – Был разработан текстовый процессор WordStar.

1982 – Был выпущен Lotus 1-2-3, объединяющий таблицы, графические изображения и поиск данных в одном ПО.

1984- Apple выпускает Appleworks.

1985 – PageMaker для Macintosh – текстовый процессов на основе GUI с возможностью публикации рабочего стола на ПК.

дело быстрее, чем один письменном виде программного обеспечения. Таким образом, любая программа теоретически может быть реализована в виде программного обеспечения или встроенного программного обеспечения. Решение сводится к тому, есть ли большая потребность в скорости или стремление к гибкости и более дешевой себестоимости продукции. Сегодня большинство программ существуют в виде программного обеспечения. Те, захваченный в прошивке включают com- программ и программ, найденных в других устройствах, таких, как антиблокировочной системы или системы впрыска топлива в автомобиле загрузочные компьютера.

Мы предпочитаем, хранить свои программы в качестве программного обеспечения, так как программное обеспечение может быть изменено. Сегодня выпущено большинство программного изделия, а затем пересмотрено и вновь пересмотрено. Мы разработали номенклатуру для

отражения выпусков программного обеспечения. Большинство названий программного обеспечения имеют два набора телефонных номеров, указанных после заголовка. Первое число является версия основного выпуска, а второй несовершеннолетний версии.

Например, Mozilla Firefox 5.2 будет пятым основная версия Firefox, а второй небольшой релиз версии 5. Незначительный выпуск, как правило, означает, что ошибки были исправлены и дыры в безопасности были подключены. Основной релиз, как правило, означает, что программное обеспечение содержит много новых функций, в том числе, вероятно, новый графический интерфейс пользователя (GUI). Также возможно, что основной выпуск переписывается с совершенно новым кодом. Мелкие релизы могут появиться каждый месяц или два, в то время как основная часть выбросов может появиться каждый год или два. Пакеты обновления являются еще одним средством выполнения незначительных выбросов; Тем не менее, пакеты обновления используют прежде всего совокупность выпускать обновления операционной системы.
Как правило, мы категоризируем программное обеспечение, либо как системное программное обеспечение или прикладного программного обеспечения. Прикладное программное обеспечение состоит из программ, с чем работают конечные пользователи работают, чтобы выполнить какую-то задачу(и). Типы прикладного программного обеспечения, столь же разнообразны, как есть карьера, потому что каждая карьера его собственная поддержка программного обеспечения. Производительность программного обеспечения состоит из приложений, которые полезны для почти всех. Производительность программного обеспечения включает в себя текстовый процессор, программное обеспечение презентации (например, PowerPoint), электронные таблицы, системы управления базами данных, программа календарь, адресную книгу и органайзер данных. Программа рисования также могут быть сгруппированы здесь, хотя Черчение более конкретно к художников и графических дизайнеров. Другой тип прикладного программного обеспечения является то, что базируется на использовании Интернета. Программное обеспечение в этом классе включает в себя клиент электронной почты, веб-браузер и FTP-клиент. Есть также компьютерные игры и развлекательного программного обеспечения (например, видео-плееры, DVD-плееры, проигрыватели компакт-дисков); хотя и не считается производительности программного обеспечения, они используются только о всех.
Производительность программного обеспечения иногда называют также

горизонтальным программным обеспечением, так как программное обеспечение может быть использовано во всех отделах в организации. Вертикальное программное обеспечение применяется вместо того, чтобы программное обеспечение, используемое отдельных групп с организацией или конкретной дисциплины. Например, существуют классы программного обеспечения, используемые музыкантами, такими как музыка, секвенсоры, образы и цифровые записывающие устройства. Программисты используют платформы разработки, которые обеспечивают не только компилятор языка программирования, но и поддержку в форме отладки по- мощи, библиотеки кода, отслеживание изменений и визуализации кода. Кинематографисты и художники используют возможности редактирования видео, фотографические редактирования и манипуляции, а также программное обеспечение для редактирования звука. Существует большое разнообразие бухгалтерского программного обеспечения, доступного от специализированного налогового программного обеспечения для цифровых регистров.

Системное программное обеспечение состоит из программ, входящих в состав операционной системы. Системное ПО является программное обеспечение, которое непосредственно поддерживает саму компьютерную систему. Такое программное обеспечение часто запускается автоматически, а не по требованию конечного пользователя. Некоторые куски операционной системы работать в фоновом режиме все время. Они часто называют демонами или услуг. Они ждут события, которые возникают, прежде чем они принимают меры. Другие части операционной системы выполняются на основании планировщика. Например, ваш антивирусный программное обеспечение может работать через каждые 24 часа, а программа, которая проверяет веб-сайт для любых обновлений и может выполняться один раз в неделю. Тем не менее, другие предметы, часто называемые коммунальные услуги, выполняются по требованию пользователя. Например, пользователь может запустить программу дефрагментации диска для повышения производительности жесткого диска. Противовирусные ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ является еще одной утилиты, хотя, как было указано выше, оно может быть выполнено на основании планировщика.

Различные уровни операционной системы, а также их связь с аппаратными средствами, прикладного программного обеспечения и пользователем, показаны на рисунке 13.1. Фигура передает наслоение, который имеет место в компьютерной системе, в которой аппаратное обеспечение работает на самом нижнем уровне, и пользователь в верхней

части. Каждое действие пользователя разлагается на все более и более первобытных операций при перемещении вниз слой.
Основные компоненты операционной системы, называются ядром. Ядро загружается, когда компьютер первой загрузке. В ядре являются компоненты, которые отвечают за управление процессами, управление ресурсами и управление памятью. Без них, вы не смогли бы начать новую программу или иметь, что выполнение программы эффективно. Ядро находится в верхней части аппаратных средств и является интерфейсом между аппаратным и программным обеспечением.
Другая часть операционной системы является коллекция драйверов устройств. Драйвер устройства является важной частью программного обеспечения. Программное обеспечение пользователя и приложения будут просматривать аппаратные устройства в общем-например, устройство ввода может реагировать на такие команды, как внести свой вклад, проверить состояние, перезагрузка. Но каждое устройство требует уникальных команд. Таким образом, операционная система переводит с более общим командам к более конкретным командам. Трансляционный ция выполняется с помощью программного обеспечения драйвера устройства. Хотя несколько драйверов устройств захватываются в BIOS ROM в качестве встроенного программного обеспечения, большинство драйверов устройств программного обеспечения. Многие популярные драйверы устройств предварительно загруженные в операционную систему, в то время как другие должны быть выгрузили из Интернета или CD-ROM, когда новое аппаратное устройство было приобретено. В любом случае, драйвер устройства должен быть установлен до того, как новое аппаратное устройство может быть использовано. Есть буквально сотни тысяч водителей доступных (согласно driverguide.com, более 1,5 млн), и нет необходимости заполнять жесткий диск пользователя со всеми из них, когда большинство никогда не будет использоваться. Другие компоненты системного программного обеспечения сидят "на вершине" ядра (опять же, смотрите Рисунок 13.1).
Компоненты, которые находятся на верхней части ядра, включают подогнанный окружением пользователя (оболочки или рабочий стол пользователя), системные программы и услуги / демонами. Как уже говорилось в главе 9, оболочки Linux позволяют пользователям создавать свои собственные псевдонимы и переменные. Desktop элементы включают в себя ярлыки, темы окон, меню и фон, появление панелей инструментов и программ в меню Пуск, так и с Windows 7, настольных гаджетов. Компоненты операционной системы (системное программное

обеспечение).

Системные утилиты программы, которые позволяют пользователю контролировать и улучшать систему эксплуатационные характеристики. Мы обсудили некоторые из них, когда мы говорили о процессах. Для повышения производительности системы, утилиты существуют для дефрагментации жесткого диска, и проверять файлы на наличие вирусов и других видов вредоносных программ. Как правило, системные утилиты должны быть установлены отдельно, поскольку они не могут прийти с операционной системой (часто, они требуют покупки и так многие пользователи могут игнорировать их наличие).

Услуги, или демоны, работают системные программы, которые обычно начинаются, когда операционная система загружается и инициализируется, но работающие в фоновом режиме. Это означает, что программа, в то время как она активна, не занимает процессорного времени, пока он не призывается. Имеющейся операционной системе будет, когда возникает определенное событие, вызовите необходимую услугу или демон для обработки события. Есть многочисленные услуги

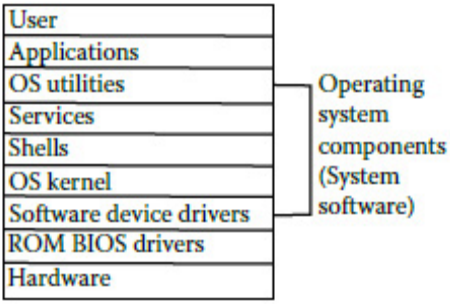


Рис. 13.1.
Слои компьютерной системы

User- Пользователь; applications – Приложения; OS utilities - Утилиты ОС; services – Услуги; shells – Корпуса; OS kernel - Ядро ОС; software device drivers - Драйверы устройств программного обеспечения; ROM BIOS drivers - Драйверы BIOS; hardware - Аппаратное оборудование (железо); operating system components (System software) - Компоненты операционной системы (Системное программное обеспечение)

OLE!

Одним из ключевых компонентов во многих формах ПО на основе Windows является возможность связывать и включать созданные объекты из одной части ПО в другую. OLE означает связывание и включение объектов. Оно было разработано в 90х годах в качестве приемника обменника динамическими данными для Windows 2.0, что позволило осуществлять передачу данными между двумя рабочими приложениями. Например, для копирования и вставки записи из MicrosoftAccess в таблицу MicrosoftExcel. Изменение записи в Access приведет к автоматическому изменению значений, хранящихся в Excel.

Тем не менее, обмен данными был ограничен. С помощью OLE, появилась возможность захватывать объект в качестве растрового изображения и наглядно переносить его в другое приложение через буфер обмена. Помимо этого, пользователь может копировать и вставлять объекты с помощью мыши.

OLE является технологией, которой владеет Microsoft. На данный момент, Microsoft требует, чтобы любое приложение имело возможность использовать OLE, если такое приложение сертифицировано и может быть совмещено с ОС Windows.

в любой операционной системе. Мы покрыли услуги в Главе 11, Услуги, Настройка Sevices и создания служб во время загрузки.

Помимо системного программного обеспечения и прикладного программного обеспечения, есть также сервера программным обеспечением. Серверы включают веб-серверов, серверов электронной почты и серверов баз данных, чтобы назвать несколько. Мы рассмотрим серверы служб и серверов, с акцентом на веб-сервере Apache.

ПРОГРАММНО-Сопутствующая терминология

В этом разделе мы кратко рассмотрим дополнительные условия, которые помогают нам понять, программное обеспечение.

Совместимость описывает платформы, часть программного обеспечения может выполняться. Большинство программного изделия, что наши компьютеры работают под управлением, требует, чтобы программное

обеспечение быть скомпилированы для этой компьютерной платформы. Есть три вопроса здесь. Во-первых, программное обеспечение должно быть скомпилированы для класса процессора данного компьютера. Pentium процессоры Intel, например, имеют разные наборы команд, чем процессоры MIPS. Если кто-то написать программу и собрать его для процессора Pentium, собранная программа не будет работать на процессоре MIPS.

Из-за близкого характера между оборудованием и операционной системы, процесс должно также быть для данной платформы операционной системы. Компьютер с процессором Pentium под управлением Linux, а не Windows, не могут выполнять программы, скомпилированные для Windows, несмотря на наличие такой же процессор. Таким образом, программное обеспечение должно быть скомпилированы как для процессора и операционной системы. И, наконец, программы могут иметь дополнительные требования для запуска, или, по крайней мере, чтобы эффективно работать, например, минимальное количество оперативной памяти или места на жестком диске или определенного типа видеокарты.

Обратная совместимость означает, что часть программного обеспечения может считывать и использовать файлы данных из старых версий программного обеспечения. Большинство поставщиков программного обеспечения обеспечить обратную совместимость, так что клиент имеет возможность обновить программное обеспечение, не опасаясь потери старых данных. Без сохранения обратной совместимости, клиент может не захотеть обновить вообще. Поддержание обратной совместимости может быть серьезной проблемой, хотя, потому что многие функции в более старых версиях могут быть устаревшими и поэтому не стоит сохранять, или может вступать в противоречие с новыми функциями. Обратная совместимость также относится к новых процессоров, которые могут работать старое программное обеспечение. Напомним, что программное обеспечение компилируется для конкретного процессора. Если новая версия процессора освобождается, оно будет поддерживать обратную совместимость? Если нет, то компьютер пользователя, которому принадлежит старый компьютер, возможно, придется приобрести новое программное обеспечение для новой процессор. Intel сохранила обратную совместимость в их x86 линии переработчиками, начиная с 8086 и через самый последний Pentium. Apple Macintosh не всегда поддерживают обратную совместимость со своими процессорами. Обновление происходит, когда компания выпускает новый большой

или малой выпуск программного обеспечения. Сегодня большинство обновлений будет происходить автоматически, когда программа запрашивает веб-сайт компании для нового выпуска. Если он найден, он загружается и устанавливается, не требуя авторизации пользователя. Вы можете установить эту функцию практически в любом программном обеспечении, так что программное обеспечение сначала спрашивает вам разрешение на выполнение обновления, чтобы вы могли запретить обновление, если это необходимо. Тем не менее, модернизация часто исправить ошибки и дыры в безопасности. Это в ваших интересах модернизации всякий раз, когда компания рекомендует его.

Патчи релизы кода, которые помогут исправить насущные проблемы с куском программного обеспечения. К ним, как правило, не рассматриваются обновления, и могут быть пронумерованы сами от раздельного проживания, номер версии программного обеспечения. Пластыри могут быть выпущены в любой момент времени и может быть освобожден часто или нечасто в зависимости от стабильности программного обеспечения.

Бета-релиз (или бета-тест) является выражение, используемое для описания новой версии программного обеспечения, которое выпускается до следующего релиза. Бета-релиз часто отправляется в списке выбора пользователей, которым доверяют, чтобы проверить программное обеспечение и сообщать об ошибках и обеспечить обратную связь об особенностях программного обеспечения. Компания сортирует ошибки и обратной связи и использует это, чтобы исправить проблемы и улучшить программное обеспечение до следующей версии. Бета-релиз может появиться через несколько месяцев или за год до предполагаемого полного освобождения.

Установка представляет собой процесс получения нового программного обеспечения и размещение его на вашем компьютере. Процесс установки не просто вопрос копирования кода на жесткий диск. Как правило, установка требует тестирования файлов на наличие ошибок, тестирование операционной системы для общих файлов, размещение файлов в различных каталогах, и изменение системных переменных, таких как переменная пути. Установка производится легко сегодня с использованием установки волшебники ARD. Тем не менее, некоторые установки программного обеспечения требует больше усилий. Этот вопрос будет рассмотрен в

Управление программным обеспечением. Пользователь может

установить программное обеспечение с расфасованный CD-ROM[1] или отключение Интернета. В последнем случае установки часто называют загрузки. Термин загрузки также относится к действию копирования файлов с некоторого сервера в Интернете, например, как скачать музыку. Установка из Интернета является довольно распространенным явлением сегодня потому, что у большинства людей есть быстрый доступ в Интернет.

Есть несколько категорий, чтобы описать наличие программного обеспечения. Собственность, или коммерческое программное обеспечение приобретается у поставщика. Такое программное обеспечение может быть получено компанией, которая продает программное обеспечение коммерчески (обычно называемый как дом программного обеспечения), или компанией, которая производит свое собственное программное обеспечение в доме, или консультантом (или консультант компании) нанял для производства программного обеспечения. Приобретая программное обеспечение, оно предоставляет вам две или три вещи. Во-первых, вы получаете само программное обеспечение в форме исполняемого кода. Если вы приобрели программное обеспечение в магазине или через Интернет, программное обеспечение устанавливается с помощью программы установки. Если вы приобрели его у консультанта, они могут выполнить установку самостоятельно. Во-вторых, вы получаете лицензию на использование программного обеспечения. Часто бывает так, что покупка программного обеспечения просто дает вам право использовать программное обеспечение, но это не ваше к своим собственным. В-третьих, вполне вероятно, что покупка дает вам доступ к полезным ресурсам, существуют ли они в качестве руководств пользователя, технической поддержки, интерактивной справки, или какой-то комбинации.

Другие категории программного обеспечения всего описывают свободное программное обеспечение, то есть программное обеспечение, которое вы можете получить бесплатно. Свободный, не обязательно означает, что вы можете использовать программное обеспечение на неопределенное время или как угодно, но это означает, что вы можете использовать его сейчас, не заплатив за него. Одной из наиболее распространенных

1 До появления оптических дисков установка часто выполнялась с дискет. Поскольку дискеты могут хранить данные около

1.5 МБ, для установки программного обеспечения требовалось использовать несколько отдельных дискет, пожалуй, не меньше дюжины. Пользователю предлагалось вставить диск 1 и диск 2, а затем диск 3 и т. д.; однако, кроме того, могло быть предложено вставить предыдущие дискеты. Часто это называли передергиванием дисков.

форм программное обеспечение, доступное в Интернете под названием категории условно-бесплатных. Shareware обычно дает вам пробную версию программного обеспечения. Компании предоставляют условно-бесплатные, как способ, чтобы побудить пользователей приобрести программное обеспечение. В пробной версии обычно будет иметь некоторые ограничения (ы), такие, как заданное количество применений, заданное количество дней, или ограничениях на особенности. Если вы устанавливаете условно-бесплатные, вы можете часто легко обновить ее до полной версии, купив лицензию на регистрацию. Это, как правило, некоторая комбинация символов, которая служит для "разблокировать" программное обеспечение. Процесс в том, что вы скачать условно-бесплатную версию. Вы используете его. Заходите на сайт, и приобрести его (обычно через покупку кредитной картой), и вы получите лицензию. Вы вводите лицензию и теперь программа переходит от условно-бесплатных до коммерческого программного обеспечения.
С другой стороны, некоторое программное обеспечение действительно бесплатно. Они делятся на несколько категорий. Во-первых, бесплатное программное обеспечение. Такое программное обеспечение является свободным от покупки, но не обязательно свободны в том, как вы его используете. Freeware, как правило, программное обеспечение, которое стало устаревшим, поскольку более новое программное обеспечение, заменил его, или это программное обеспечение, что оригинальный производитель больше не желает (будь то физическое лицо или компания), чтобы поддерживать и оказывать поддержку. Вы связаны некоторыми соглашениями, когда вы установите его, но в противном случае вы можете использовать его, как если бы вы купили его. Программное обеспечение общественным достоянием, с другой стороны, это программа, которая была перемещена в общественное достояние. Как и бесплатное программное обеспечение, это программное обеспечение, которое никто не хочет, чтобы заработать деньги прочь или поддержки. Все, что находится в свободном доступе, можно использовать, однако вы чувствуете; никто не претендует на какие-либо права

Несмотря на простоту установки через окна инсталляторов, есть еще вещи, которые пользователь должен знать. Наиболее важной проблемой является ли у вас ресурсы, необходимые для запуска программного обеспечения. Наиболее важные ресурсы являются достаточно свободного места на жестком диске, достаточно быстрый процессор и достаточное количество оперативной памяти. Перед

установкой программного обеспечения, вы можете захотеть, чтобы проверить требования к установке. Вы также должны убедиться, что вы устанавливаете правильную версию программного обеспечения, то есть версия, скомпилированные для вашего конкретного оборудования и операционной системы платформы. Вы также должны убедиться, что у вас есть достаточные права доступа для установки программного обеспечения. Являетесь ли вы администратором? Вы вошли в систему как администратору выполнить установку? Или, как пользователь, имеет к администратору разрешено вам доступ к установке нового программного обеспечения?

В Linux, существует несколько различных способов установки программного обеспечения. Во-первых, есть упрощал подход с использованием мастера установки из загруженного программного обеспечения, так же, как вы делаете в Windows. Второй подход, также как и Windows, чтобы установить программное обеспечение с CD-ROM (или DVD). Одно из отличий в том, что много программного обеспечения, можно найти на компакт-диске установки системы Linux операционной. Для установки с компакт-диска, вы должны получить установочный компакт-диск и использовать компонент Установка и удаление программ (находится в меню Applications).

Другой вариант в Linux является установка программных пакетов из командной строки. Есть несколько подходов, которые можно предпринять.

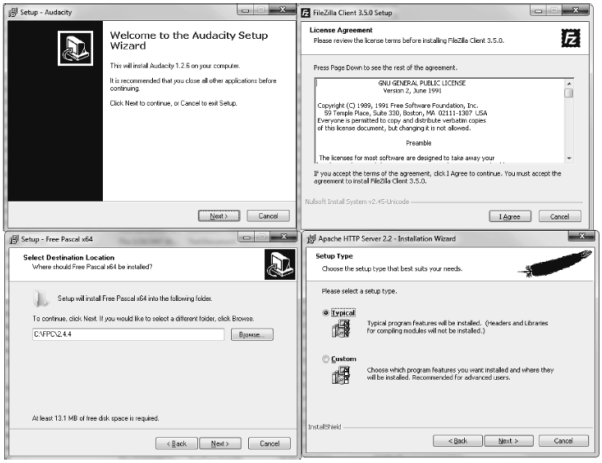


Рис. 13.2.
Варианты окон установки Windows

Самый простой командной строкой будет подход через менеджер пакетов программ. В Red Hat, есть два, ням (Желтая собака Updater, Modified) и оборотов в минуту (менеджер Red Hat Package). Программа RPM работает на оборотах файлов. Файл RPM представляет собой пакет: архив сжатых файлов, которые содержат исполняемую программу (ы) и связанные с ним файлы, а также описание зависимостей, необходимых для запуска файлов. Зависимости файлы, эти файлы полагаются на. Программа RPM будет тестировать эти зависимостями и предупреждает пользователя о любых отсутствующих файлов. Программа использует ням оборотов в минуту, но обрабатывает зависимостей для вас, что делает его гораздо проще.

Для установки программного обеспечения с помощью оборотов в минуту, то инструкция мин -i packagename.rpm. Для неинформированного программного обеспечения стойло, используйте -e (для стирания), как в оборотах в минуту -e PackageName. Чтобы обновить пакет, используйте -u. Наконец, вы можете также использовать -q для запроса базы данных, чтобы найти оборотов в минуту больше информации о либо уже установленных пакетов или пакетов, которые еще должны быть установлены (до тех пор, как файлы RPM присутствуют). Вы можете комбинировать -q и -l, чтобы получить список файлов в данном пакете или -q -a и опций, чтобы получить список всех уже установленных пакетов. Есть оборотов в минуту хранилищами доступны во многих местах в Интернете, такие как HTTP: //centos.karan.org / EL4 / дополнительные / стабильный / i386 / RPMS /. Помимо использования RPM из командной строки, вы можете также запустить менеджеров пакетов с графическим интерфейсом.

Yum является продуктом из Университета Дьюка для пользователей Linux. Yum призывает оборотов в минуту, чтобы выполнить установку; Тем не менее, он имеет возможность отследить зависимости и устанавливать любые зависимые пакеты по мере необходимости. Как и при оборотах в минуту, ням может установить или обновить программное обеспечение, в том числе самого Linux ядра. Это делает его легче поддерживать группы машин без необходимости вручную обновлять каждый из них с помощью оборотов в минуту. Использование Yum очень просто; он делает большую часть работы сам по себе (хотя это может занять много времени, чтобы обновить пакеты и установить новые). Простая ням команда позволит установить заголовок, где название пакета программного обеспечения. Другие ням команды (вместо установки) включают в себя список (в котором перечислены пакеты, которые

имеют данное имя в них, как и в списке ням "Foobar"), chkconfig ням на (который планирует Yum выполнить обновление каждую ночь), ням удалить название (для удаления целевого пакета, сняв все зависимости), и ням -у установить заголовок (который делает установку, не отвечая "да" на любой вопрос да / нет, чтобы установить может быть сделано без вмешательства человека).

Более сложной формой установки в Linux является установка программного обеспечения из исходного кода. Исходный код программы код, написанный на языке высокого уровня (например, C ++ или Java). Такая программа не может быть выполнена непосредственно на компьютере, и, следовательно, должен быть собран первый (мы обсудим компиляцию в следующей главе). Потому что большая часть программным обеспечением доступного в Linux это программное обеспечение с открытым исходным кодом, вполне вероятно, что пользователь Linux нужно будет понять, как установить программное обеспечение из исходного кода. Перед установкой программного обеспечения из исходного кода, вы можете все еще смотреть, чтобы увидеть, если программное обеспечение доступно в исполняемом формате, либо с помощью мастера установки или в качестве пакета оборотов в минуту.

Установка из исходного кода первого требуется вручную распаковке в комплект программного обеспечения. Большая часть программного обеспечения с открытым исходным кодом поставляется в архив. Есть несколько механизмов для архивирования файлов, но наиболее распространенным подходом через Unix на основе архива ленты программы под названием деготь. Программа дегтя была первоначально предназначена для архивирования файлов, которые будут сохранены на ленту. Тем не менее, это удобно и довольно проста в использовании, поэтому многие программисты используют его, чтобы связать их программное обеспечение вместе.

Файл смолы будет иметь .tar как расширение. Кроме того, как только бетонирование произойдет вместе, расслоение может быть сжато для экономии места и позволить ускорить процесс передачи расслоения через Интернет. Сегодня программа Zip проекта GNU, GZIP, часто используется. Это добавит расширение .gz к имени файла. Например, в комплекте и промелькнутый архив может появиться как bundle.tar.gz. Пользователь должен сначала распаковать сжатый файл с помощью Gun-zip. Это восстанавливает архив или сверток. Для того, чтобы разделять его, следующим шагом является распаковка сверток, используя смолу.

При создании файла TAR, необходимо указать как исходный файл (ы) и имя файла гудрона с параметрами, которые указывают, что вы создаете деготь файл (c) и что адресат является файлом (e) вместо ленты. Эта команда может выглядеть следующим образом: смола -cf bundle1.tar
* .txt Или он может выглядеть следующим образом: смола -cf bundle2.tar / TEMP. В первом случае будет принимать all.txt файлы и очернить их в архив, в то время как последний пример принимает все файлы в каталоге / Температуры, в том числе самого каталога, и помещает их в архив. Распаковка файла, команда TAR -xf bundle2.tar. Параметр x обозначает экстракт, для извлечения файла. Если бы вы были распакуйте bundle2. tar, было бы создать каталог / TEMP, а затем поместить содержимое, которые были сохранены в директории оригинал / TEMP в этот новый / временный каталог.
После того как программа была разделены, вы должны скомпилировать его. Наиболее открытым исходным кодом про- граммы написаны на C или C ++. Составители наиболее часто используемые в Linux для C и C ++ являются НКУ (C проекта GNU / C ++ компилятор) и г ++ (GCC выполнения определенных параметров C ++). К сожалению, в случае компиляции C или C ++ программ, вполне вероятно, что там многочисленны файлы для решения в том числе файлы заголовков (.h), файлы исходного кода (.c или .cpp) и объектные файлы (.o). Без хорошей сделки обучения, пользователь пытается собрать и установить программное обеспечение может быть полностью потеряна. Linux, однако, имеет три очень полезные команды, которые делают процесс простым. Эти команды настраивают, сделать, и сделать установку. Эти команды запуска скриптов, написанных программистом, для выполнения различных задач, необходимых для успешной компиляции и установки программного обеспечения.
Команда конфигурирования используется для создания или изменения файла сборки сценария. Сценарий Makefile является инструкции программиста о том, как компилировать исходный код. Команда может настроить, например, использоваться системным администратором, чтобы указать директорий RIES для размещения различных программных файлов и указать модули, которые должны быть скомпилированы и включены в программное обеспечение. Если Makefile не существует, то команда конфигурирования должна быть выполнена. Если Makefile уже существует, то команда конфигурирования иногда необязательна. Команда конфигурации на самом деле сценарий, также созданный программистом. Чтобы запустить скрипт конфигурации, один входит

в ./configure (а не более)./ Означает "выполнить этот скрипт". После завершения, есть теперь Makefile.
Команда сделает выполняет Makefile сценарий (Примечание: сделать будет выполняться либо Makefile или Makefile, в зависимости от того, что находит). Сценарий Makefile содержит все команды для компиляции программного обеспечения. Сценарий Makefile может также содержать другие разделы помимо инструкции компиляции. Они включают в себя установочный раздел для выполнения этапов установки (если программное обеспечение должно быть установлено в один или несколько каталогов), чистый раздел, чтобы очистить любые временно создаваемые файлы, которые больше не нужны, и участок смолы таким образом, чтобы комбинировать файлы могут быть сохранены. Если все идет гладко, все пользователь должен сделать, это ввести команды ./ configure, заставлю установить. Хороший Makefile будет называть себя сделать чистой после выполнения операций по установке. Эти шаги могут занять от нескольких секунд до нескольких минут в зависимости от количества кода, который требует компиляции.
Прежде чем пытаться выполнить компиляцию и установку программного обеспечения, администратор должен сначала прочитать текстовый файл README, который поставляется с распределением. Подобно Makefile сценарий, README написан программистом, чтобы объяснить, инструкции по установке. Чтение файла README поможет администратору выполнять этапы компиляции и монтаже, особенно если Есть конкретные шаги, которые программист ожидает, являющимися администраторами для выполнения заранее, потому что программист сделал некоторые предположения целевой системы Linux. Например, программист может предположить, что GCC устанавливается

```
FILES = file1.c file2.c file3.c    // define program files to be compiled
CC = gcc                           // C compiler
CFLAGS = -c -Wall                  // C compiler options
LFLAGS = -Wall                     // C compiler options for linking
all:                               // start here - call files
     files
files: $(FILES)                    // for each item in the FILES variable
     $(CC) $(LFLAGS) $(FILES) -o   // perform linking
.c:                                // for each .c file, compile it
     $(CC) $(CFLAGS) -o
clean:                             // if make is called with clean, do this
     \rm *.o myprog                // step (remove all files except the c files)
tar:                               // if make called with tar, tar all files
     tar cfv myprogram.tar file1.c file2.c file3.c myprogram.h
```

Рис. 13.3.
Пример скрипта Makefile

в определенном месте или / USR / бен доступна. Файл README может также включать в себя намеки администратору для исправления ошибок в случае скрипт Makefile не работает.

Это короткий пример. Многие Makefile сценарии достаточно велики. Пример Makefile сценарий из рисунка 13.3, показывает, что содержимое файла сборки не всегда легко понять. Первые четыре строки этого сценария определяют переменные, которые будут использоваться в файле Makefile. В этом примере есть три программных файлов: file1.c, file2.c и file3.c. Они хранятся в переменной FILES. Следующие три строки определяют переменные для использования на стадиях компиляции. Эти переменные определяют имя компилятора (GCC, C компилятор проекта GNU), флаги компилятора для использования во время компиляции, а также флаги компоновщика для использования на стадии связывания компиляции. Обратите внимание, что элементы на правой части файла являются комментарии (указанные будучи предшествуют // символов). Комментарии игнорируются интерпретатором Баш, при выполнении сценария.
Следующие шесть строк представляют собой команды для выполнения, когда команда выдается. Идея этих шести линий является то, что каждая первая определяет метку, возможно, с аргументами (например, $ (FILES)), а на следующей строке, с отступом, команда для выполнения. Всего: метка не имеет аргументов, и его единственная команда, чтобы выполнить команду под файлы: метка. Это часто используется в make-файла для команды сделать все. Файлы этикетки перебирает все файлы в переменной $ (FILES) (т.е. три C-файлы) и вопросы НКУ, связывающую команду на каждом. Под это линия с, которая используется для компиляции каждой программы C. Компиляцию и компоновку шагов можно сделать вместе, но часто разделенных для удобства. Последние четыре строки относятся к инструкциям, которые будут выполняться, если вопросы пользователей либо сделать чистыми или сделать деготь. Прежний очищает все временные файлы из текущего каталога, а последний создает деготь файл всех соответствующих программных файлов. Поскольку это был такой простой процесс компиляции, нет необходимости в установке секции. Программист (s) записывает файл Makefile и упаковывает его со всем исходным кодом и любые другие необходимые файлы (например, документации). Затем администратор устанавливает программное обеспечение, опираясь на сценарий Makefile. Это не редкость, что вы, возможно, придется изучить сценарий и внести изменения. Например,

представьте, что Makefile переехал полученные файлы в каталог / USR / местные / бен но вы хотите, чтобы переместить его в / USR / бен вместо этого, требуя, чтобы вы сделали небольшую корректировку в сценарии, или если вы хотите изменить настройки для
компилятор от -g и -Wall к чему-то еще.
При использовании средств управления пакетами, такими как оборотов в минуту и ням, почему настроить, сделать, и сделать установку важно? Они не могут быть, если вы счастливы, используя способный код доступен через репозиторий RPM веб-сайтов. Тем не менее, есть две ситуации, когда они становятся существенными. Во-первых, некоторое программное обеспечение с открытым исходным кодом не приходит в исполняемой форме и, следовательно, у вас нет выбора, кроме как скомпилировать его самостоятельно. Во-вторых, если вы или кто-то в вашей организации хотели изменить исходный код, вы бы все равно придется установить его с помощью Configure, сделать, и сделать установку команды. Мы кратко рассмотрим использование задаете в следующем разделе.

Сервисы и серверы

Термин сервер может быть использован для обозначения трех различных вещей. Первый специализированный компьютеру используется для предоставления услуг. Серверы, как правило, высокопроизводительные компьютеры с большими жесткими дисками. В некоторых случаях серверы встроены в шкафах, так что несколько или даже десятки жестких дисков могут быть вставлены. Сервер обычно представляет собой автономный компьютера. То есть, это не чей-то настольный компьютер, а также не пользователи, как правило, необходимо войти непосредственно на сервер, если они не администраторы сервера.
Второе использование термина сервера программного обеспечения, которое запускается на компьютере. Серверное программное обеспечение включает в себя службу печати, FTP-сервер, веб-сервер, сервер электронной почты, сервер баз данных и файловый сервер. Сервер программного обеспечения должен быть установлен, настроен и управление ими. могут быть вовлечены эти шаги. Например, настройки и управления веб-сервера Apache потенциально может быть работа сама по себе, а не один оставляется либо системного администратора или веб-разработчик.
Третье использование для сервера является сбор аппаратных средств,

программного обеспечения, а также данные, которые вместе составляет сервер. Например, мы могли бы обратиться к веб-серверу в качестве аппаратных средств, программного обеспечения и документов, хранящихся там.
В этом разделе мы рассмотрим несколько типов программных серверов. Мы подчеркиваем, веб-сервер Apache с взглядом на установку и минимальной конфигурации. Ниже приведен список наиболее распространенных типов серверов.

Веб сервер
Роль веб-сервера в ответ на HTTP-запрос. HTTP запросы чаще всего приходят из веб-браузеров, но другие программы и пользователи также могут подавать заявки. HTTP является протокол передачи гипертекста. Большинство запросов HTTP предназначены для HTML-документов (или перемен- муравьев HTML), но может включать в себя как раз о любом веб-ресурсе доступны. HTTP похож на FTP в том, что файл передается с помощью сервиса, но в отличие от FTP, типичный запрос сделан, нажав на ссылку в веб-браузере-и, таким образом, пользователь не должен знать имя сервера или имя или путь ресурса. Они все объединены в сущности, известной как URL. Apache сервер с открытым исходным кодом является самым популярным веб-сервер, используемый сегодня. Помимо обслуживания запросов, Apache может выполнять сценарии для создания динамических страниц, использовать механизмы безопасности, чтобы гарантировать, что запросы не являются формами атак, регистрации запросов, так что аналитики могут найти тенденции в поведении браузера общественности на веб-сайте, а также многочисленны другие задачи , Мы смотрим на Apache более подробно ниже. Internet Information Server (также называемый Internet Information Services), или IIS, является веб-сервер Microsoft, впервые доступен для операционных систем Windows XP, а также поставляется с Windows Vista и Windows 7.

Прокси-сервер
Прокси-сервер используется в организации, чтобы действовать как гигантский кэш веб-страниц, что кто-то в этой организации недавно извлеченной. Идея заключается в том, что если многие люди в организации склонны рассматривать одни и те же страницы, кэширование этих страниц локально разрешает доступ в будущем получить страницу локально, а не удаленно. Еще одна функция прокси-сервера, чтобы обеспечить

определенную степень анонимности, так как IP-адрес записывается веб-сервером в ответ на запрос, является то, что прокси-сервера, а не отдельного клиента. Прокси-серверы также могут быть использованы для блокирования определенного контента от возвращения, например, отклонения запросов выход на определенные серверы (например, Facebook) и отказаться от ответов, которые содержат определенное содержание (например, слово "порно"). Squid является наиболее часто используемый прокси-сервером Apache, хотя может также служить в качестве прокси-сервера. Кальмары, как Apache, является открытым исходным кодом. Среди типов вещей системным администратором или веб-администратор может настроить кальмаров являются количество и размер кэшей, а кальмар брандмауэра разрешить или запретить доступ различных типов.

Сервер баз данных

База данных представляет собой структурированным хранилищем данных. Система управления базами данных является программное обеспечение, которое отвечает на запросы пользователей для создания, обработки и извлечения записей из базы данных. Есть доступны многочисленные системы управления базами данных пакетов. Сервер базы данных обеспечивает доступ к базе данных по сети. Сервер базы данных будет выполнять такие задачи, как анализ данных, обработки данных, безопасности и архивирования. Сервер базы данных может или не может хранить саму базу данных. Разделив сервер из базы данных, можно использовать механизмы разных для хранения базы данных, например, сети хранения данных для поддержки балансировки нагрузки. MySQL является очень популярным, открытым исходным кодом сервера баз данных. MySQL на самом деле общее название для ряда различных баз данных. MySQL Community является сервером базы данных. Существует также прокси-сервер для базы данных MySQL, сервер кластера для взаимодействия транзакционной базы данных высокоскоростной и инструмент для поддержки строительства GUI.

Сервер FTP

FTP-сервер, как веб-сервер в том, что он принимает файлы и позволяет клиентам получать доступ к этим файлам. Тем не менее, с FTP, доступ в виде загрузки файлов и загрузки файлов. Любые файлы загруженные, сохраняются на диске в отличие от веб-сервиса, где большинство файлов загружаются непосредственно в веб-браузере клиента. FTP доступ либо

требует, чтобы пользователь иметь учетную запись (в отличие от HTTP), или требует, чтобы пользователь войти как анонимный пользователь. Анонимный пользователь имеет доступ к публичным файлам (часто в специальный каталог, называемый / паб). FTP является старый протокол HTTP, чем и в значительной степени были заменены HTTP за исключением загрузки файлов. FTP сервер доступен в Linux, даемоне. Эта услуга на основе текста. Расширение FTP является ФСНП, безопасная форма FTP. Хотя SFTP еще один безопасный способ для обработки передачи файлов, он не основан на протоколе FTP. Можно также запустить SSH, а затем использовать FTP изнутри SSH. Популярные клиентские средства FTP для выполнения FTP включают WS-FTP и FileZilla, оба из которых являются графическим интерфейсом программы, которые посылают команды на FTP-сервер. FileZilla может также работать в качестве FTP-сервера, как можно Apache.

Файловый сервер

Файл-сервер находится, в сущности, компьютер с большим жестким диском хранения файлов, что любой или многие пользователи сети могут пожелать получить доступ. Более сложные файловые серверы используются не только для хранения файлов данных, но и для хранения программного обеспечения для запуска по сети. Файловые серверы, используемые, чтобы быть очень видное место в компьютерных сетях, так как объем памяти жесткого диска был непомерно дорого. Сегодня, с 1 ТБ пространства на жестком диске стоимостью $ 100, многие организации отвергнутые использование файловых серверов. Файл-сервер по-прежнему выгодно для большинства организаций, поскольку он поддерживает совместный доступ к файлам, позволяет легко резервное копирование пользовательских файлов, позволяет для шифрования данных хране-, и позволяет осуществлять удаленный доступ к файлам. Есть много различных файловых серверов, доступных с NFS (файловая система Network) является наиболее популярным в мире Unix / Linux.

Сервер электронной почты

Сервер электронной почты предоставляет услуги электронной почты. Его работа состоит в том, чтобы принимать запросы электронной почты от клиентов и отправлять сообщения на другие серверы электронной почты, а также получать сообщения электронной почты с других серверов и предупреждает пользователя, что электронная почта приходит. Операционные системы Linux и Unix имеют встроенную службу

электронной почты, Sendmail. Клиенты могут свободно использовать любое количество различных программ для доступа к электронной почте. В операционной системе Windows, существует большое разнообразие доступных серверов электронной почты, включая Eudora, CleanMail и Microsoft Windows Server. В отличие от FTP и веб-серверов, которые имеют огра- ITED количество протоколов для обработки (в первую очередь FTP, FTPS, HTTP и HTTPS), есть еще протоколы электронной почты, что серверы должны обрабатывать: IMAP, POP3, SMTP, HTTP, MAPI и MIME.

Система доменных имен

Для удобства, люди, как правило, используют IP-псевдонимы при доступе к веб-серверы, FTP-серверы и серверы электронной почты. Алиасы легче запомнить, чем фактическом IP-адреса. Но IP-псевдонимы не могут быть использованы маршрутизаторами, поэтому требуется перевод с IP-псевдонима для IP-адреса. Это, как правило, осуществляется с помощью системы доменных имен (DNS). DNS состоит из картирование информации, а также указатели на другие DNSS, так что, если запрос не может отображается, то запрос может быть перенаправлен на другой DNS. В Linux, общая программа DNS будет связывать. Программа Dnsmasq программа форвардера часто используется на небольших Linux (и MacOS) сетей. В операционной системе Windows, Microsoft DNS является компонентом Windows Server. Cisco также предлагает DNS-сервер, Cisco Network Регистратора (CNR).

Мы закончим эту главу, глядя на то, как установить и настроить Апач web-сервер в Linux. Поскольку большая часть программного обеспечения в Linux является открытым исходным кодом, мы рассмотрим, как установить Apache с помощью Configure, сделать, и сделать шаги установки.

Первый шаг, чтобы загрузить Apache. Вы можете получить Apache из httpd.apache.org. Хотя Apache доступен как для Linux / Unix и Windows, мы рассмотрим только при установке его в Linux, так как версия для Windows имеет ряд ограничений. Из упомянутого выше сайта, необходимо выбрать Загрузить из самой последней стабильной версии (нет необходимости загружать и устанавливать бета-версию, потому что Apache получает регулярные обновления и поэтому любые новые функции будут вплетены в стабильную версию, прежде чем слишком долго) , Apache доступен как исходный код и двоичные (исполнимые) формат для Windows, но исходный код только в Linux. Выберите один из выбранных элементов Unix Source. Есть зашифрованы и зашифрованную

версии доступны. Например, при выборе HTTP-2.4.1.tar.gz выбирает Apache версии 2.4.1, битумированной и Zipped. После загрузки, вы должны распаковать и разархивировать его. Вы можете это с смогло полностью выполнить возложенную одной командой:

деготь -xzf HTTPD-2.4.1.tar.gz

Варианты x, г и е стенд для "экстракт", "Unzip" и "файл, заданный в качестве аргумента", соответственно. Результатом этой команды будет новый каталог, HTTPD-2.4.1, в текущего каталога. Если выполнить на этой Ls директории, вы увидите список, как показано на рисунке 13.4.
Каталог содержит несколько подкаталогов. Наиболее важными из этих направлений являются серверы, которые содержат весь исходный код C; что включают в себя различные файлы заголовков C; и модули, которые содержит код, необходимый для построения различных модулей, которые поставляются с Apache. Есть несколько сценариев в этом каталоге. Build - конфигурация и настройка сценариев помогут настроить Apache, изменив сценарий Makefile, чтобы соответствовать спецификациям пользователя. Сценарии Makefile.in и makefile.win используются на самом деле выполнения этапов компиляции.
Следующим шагом для системного администратора для выполнения этого шага настроить. Использование этой команды ./configure [КЛЮЧ] ... [VAR = VALUE] ... Если вы ./configure -h, вы получите помощь от того, какие опции, переменные и значения. Есть несколько вариантов, доступны способности и в значительной степени будет исключением, возможно, не использовался -h, чтобы получить файл справки и -q работать в "тихом режиме", чтобы избежать длительных распечаток сообщений.
Более общие аргументы в конфигурировании устанавливают ряд переменных окружения. Многие из них используются для либо изменить каталоги по умолчанию или указать модули, которые должны быть

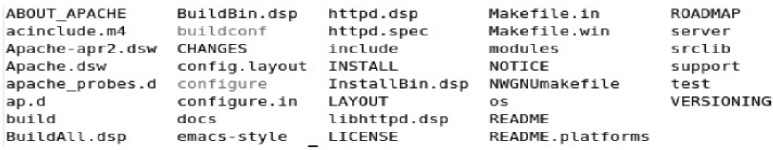


FIGURE 13.4 Listing of the Apache directory.

Рис. 13.4.
Внесение в список Apache

установлены. Вот некоторые примеры:

--bindir = DIR -replace исполняемый каталог по умолчанию с DIR
--sbindir = DIR -replace по умолчанию системного администрирования каталога исполняемый с DIR
--sysconfidir = DIR -replace каталог конфигурации по умолчанию с DIR
--datadir = DIR -replace каталог данных по умолчанию (обычно / Var / WWW / HTML) с DIR
prefix = DIR -replace все каталоги по умолчанию с DIR
--enable-нагрузки все-модули -load все доступные модули
--enable-модули = МОДУЛЬ-LIST -load все модули, перечисленные здесь
--enable-AuthN-файл -load модуль, касающийся контроля аутентификации
--enable-включают -enable стороне сервера включает в себя модуль (для запуска CGI скрипт)
--enable-прокси -enable модуль прокси-сервер
--enable-SSL -enable SSL / TLS модуль поддержки

Типичная команда в данный момент может быть:

./configure --prefix = / USR / местные / --enable-модули = ...

Где вы будете заполнять в нужных модулей вместо ... или опустить -Enable-модулей = полностью, если нет начальных модулей, которые вы хотите установить. Команда конфигурирования, скорее всего, займет несколько минут, чтобы выполнить. После завершения, ваш Makefile доступен. Введите команду Make, которая сама по себе займет еще несколько минут, чтобы закончить. Наконец, когда все сделано, типа сделать установки. Теперь будет иметь установленную версию Apache, но он еще не запущен.
Есть и другие способы установки Apache. Например, исполняемая версия доступна с помощью установочного диска CentOS. Вы можете установить это с помощью Add / Remove Software команду в меню Приложения. Это требует, чтобы у вас был установочный диск в доступе. Еще более простой подход заключается в использовании ням путем ввода ням установить HTTPD. Тем не менее, эти два подхода ограничивают вас варианты по умолчанию и пический Apache доступны в соответствующих хранилищах (один на диске и один на веб-сайте RPM), а не самой последней версии, размещенном на веб-сайте Apache.
Теперь, когда установлен Apache, мы можем посмотреть на то, как

использовать его. Для запуска Apache, необходимо запустить службу Apache. Найти двоичный каталог, в данном случае это будет / USR / местные / apache2 / бен. В этом подкаталоге является apachectl скрипт. Вы хотите, чтобы начать это путем выполнения ./apachectl запуска. Теперь Ваш сервер должен быть функциональным.

Используя описанную выше конфигурацию, все ваши конфигурационные файлы будут храниться в / USR / местные / apache2 / conf. Основной файл conf. является httpd.conf. Вы редактируете этот файл для дальнейшей настройки и адаптируете свой сервер для ваших конкретных потребностей. Этот файл не единственный файл конфигурации. Вместо этого, он является общим, чтобы разбить команды конфигурации на несколько файлов. Директива Include используется для загрузки других файлов конфигурации. Вы можете найти в файле httpd.conf, что многие из заявлений, в частности, были закомментированы.

Давайте посмотрим, что вы могли бы сделать, чтобы настроить сервер. Вы можете установить дополнительные IP-адреса или порты, которые Apache будет реагировать. Вы можете захотеть изменить расположение файлов журналов и тем, что в настоящее время регистрируется. Вы можете использовать различные модулей, которые вы указали в команде ./configure. Вы можете захотеть изменить количество детей, что Apache будет икру и продолжать работать в фоновом режиме для обработки HTTP-запросов. Кроме того, вы можете добавить различные контейнеры. Контейнеры описывают, как конкретно относиться к подкаталог файлового пространства, группы файлов, которые разделяют то же имя, или конкретный URL. Каждый раз, когда вы измените файл конфигурации, необходимо перезапустить Apache, чтобы эти изменения вступили в силу. Вы можете либо выдавать команды ./apachectl остановки и

./apachectl запуск или перезапуск ./apachectl (последняя команда будет работать только тогда, когда Apache находится в данный момент ключа работает).

Местоположение вашего "веб-пространства", которые хранятся в вашем компьютере Linux, должно быть / USR / местные / apache2 / веб. Это где вы будете размещать большинство ваших веб-документов. Тем не менее, вы можете захотеть создать подкаталоги, чтобы лучше организовать файловое пространство. Вы можете также контроль доступа Паритера отдельных подкаталогов, так что вы могли бы, например, поместить определенные ограниченные файлы под HTTPS доступа, который требует авторизованного входа в систему, с помощью контейнеров каталогов.

Файл конфигурации Apache состоит из комментариев и директив. Комментарии английские описания директив в файле наряду с полезными предложениями для внесения изменений. Директивы разбиты на три типа. Во-первых, директивы сервера влияют на весь сервер. Во-вторых, директивы контейнеров влияние, как сервер обрабатывает конкретные директорий и файлов. И, наконец, включают заявления, используются для загрузки других файлов конфигурации.
Директивы сервера влияют на весь сервер. Ниже приведены примеры семи директив сервера.

ServerName www.myserver.com Апач Пользователя
Группа Апач TimeOut 120
MaxKeepAliveRequests 100
Слушать 80
DocumentRoot "/ USR / местные / apache2 / веб"

ServerName является IP-псевдонимом, с помощью которого эта машина будет реагировать. Пользователь и группа определяет имя, под которым этот процесс будет появляться при работе в Linux (например, когда вы делаете команда пс). TimeOut обозначает количество секунд, что сервер должен пытаться общаться с клиентом до выдачи тайм-аута команды. MaxKeepAliveRequests устанавливает число запросов, одно соединение может отправить перед соединение закрывается и новое должно быть установлено соединение между клиентом и сервером. Слушайте идентифицирует номер порта (ов), что сервер будет слушать. Слушайте также разрешает IP-адреса, если сервер работает на компьютере, который имеет несколько IP-адресов. DocumentRoot хранит расположение каталога на сервере веб-страниц. По умолчанию для этого адреса, как правило,
/ Var / WWW / html; Тем не менее, мы перегрузили это во время стадии ./ configure, когда мы определили местоположение в файловой системе для всех Apache.
Директива Include позволяет указать другие конфигурационные файлы. Например,

Включите conf.d / *. Конф

будет загружать все other.conf файлы, найденные в подкаталоге conf.d в момент запуска Apache. Это позволяет отделить директивы на множество

файлов, сохраняя каждый файл короче и компактнее. Это также позволяет администратору сервера Интернета на директив группы в кате- горий и решить, какие директивы должны быть загружены, и которые не должны быть загружены, просто комментируя некоторые из заявлений, в частности. В версии CentOS Linux в, есть два .conf файлов, найденных в этом подкаталоге: один для настройки прокси-сервис и приветственный файл .conf.

Последний тип директивы фактически класс директив называемых контейнеров. Контейнер описывает местоположение, для которого директивы должны быть применены. Есть несколько типов контейнеров.

ГЛАВА 14. ПРОГРАММИРОВАНИЕ

ИТ-администраторам часто приходится писать скрипты для поддержки систем, которые они пронесли Tain. В этой главе рассматриваются задачи программирования. Для тех, кто не знаком с программированием, эта глава охватывает больше, чем просто оболочки сценариев. Во-первых, глава рассматривает типы языков программирования, которые впервые были представлены в главе 7. Далее, в главе описывается, с помощью многочисленных примеров, типы инструкций языка программирования, найденные в большинстве языков программирования: входные и выходные инструкции, операторы присваивания , заявления выбора, операторов итерации, подпрограммы. С этим введением, глава затем поворачивается к скриптинг как на языке Bash интерпретатора и DOS, сопровождается многочисленными примерами.
Цели обучения данной главы являются

• Опишите различия между языком высокого уровня и программирования низкого уровня.
• Обсудить эволюцию языков программирования с изучения новаций в области программирования.
• Ввести типы инструкций языка программирования.
• Представить через примеры программирования как в оболочке Bash и DOS языков.

В этой главе мы рассмотрим программирования и рассмотрим, как писать сценарии оболочки и в оболочке Bash и DOS.

Виды языков

Только типы программ, которые могут выполнять команды компьютера, написанные на машинном языке его к компьютеру. Машинный язык состоит из инструкций, написанных в двоичной системе, которые непосредственно ссылаются на набор команд компьютерного процессора. Написание программ на машинном языке является чрезвычайно сложной

задачей. К счастью, хотя, программисты уже давно избегать написания программ непосредственно на машинном языке из-за наличия языка переводчиков. Язык переводчик это программа, которая принимает одну программу в качестве входных данных, а также создает программу на машинном языке в качестве вывода. Каждый переводчик язык адаптирован переводить с одного конкретного языка на машинный язык данного процессора. Например, один переводчик будет использоваться, чтобы перевести программу на C для компьютера Windows, а другой будет использоваться, чтобы перевести программу на C для рабочей станции Sun. Аналогичным образом, программы переводчик различных потребуется для перевода программ, написанных на языке COBOL (Кобол), Ада, или FORTRAN.

Есть три типа языковых переводчиков, каждый из которых для другого класса языка программирования. Эти классы языков являются языками сборки, высокий уровень скомпилированные языков, и высокий уровень интерпретируемые языки. Переводчики для этих трех типов языков называются ассемблеры, компиляторы, и переводчиков, соответственно. языки Ассамблеи были разработаны в 1950-х годах в качестве средства, чтобы избежать использования машинного языка. Несмотря на то, что легче написать программу на языке ассемблера, чем в машинном языке, большинство программистов избегать языков сборки, поскольку они почти столь же сложным, как машинного языка программирования. Рассмотрим простую инструкцию C

для (я = 0; <п; я ++)
а [я] = а [я] +1;

который итерацию по п местах массива, увеличивающиеся каждый из них. В сборке (или машина) языка, эта простая инструкция будет состоять из нескольких до нескольких десятков индивидуальных инструкциями. После того, как возникли лучшие языки, использование языка ассемблера программирования был ограничен для программистов, которые разрабатывают программы, которые требуют наиболее эффективного исполняемого кода возможным (например, с ранних компьютерных игр). Сегодня почти никто не пишет на языке ассемблера программы из-за превосходства языков программирования высокого уровня.

Языки программирования высокого уровня, возникла из-за громоздкости писать в машинных или ассемблере. Чтобы проиллюстрировать разницу, см Рисунок 14.1. На этом рисунке, программа представлена на трех

языках: машинный язык (для вымышленной компьютера называется MARIE), ассемблере (опять же, в MARIE), и язык программирования C высокого уровня. Обратите внимание на то, что код C является гораздо более кратким. Это происходит потому, что многие задачи, такие как вычисление значения (например, a = Ь - 1) требует несколько шагов в машинных и сборочных языках, но только один на языке высокого уровня. Также обратите внимание, что каждая инструкция на языке ассемблера имеет эквивалентную команду на машинном языке, поэтому эти две программы имеют одинаковый размер.

Языки программирования высокого уровня, входят в один из двух типов, составленное языков и интерпретируемых языков. В обоих случаях, язык переводчик должен перевести программу из языков высокого уровня на машинный язык. В составленном языка, переводчик языка, который называется компилятор, переводит всю программу в одно время, создавая execut- способную программу, которая может быть запущена на более позднее время. В интерпретируемый язык, программный код записывается в интерпретируемой среде. Переводчик принимает совсем недавно введенную команду, переводит его в машинный язык, и выполняет его. Это позволяет программисту экспериментировать при разработке кода. Например, если программист не уверен, что конкретная команда может сделать, программист может ввести команду и протестировать его. Если он не делает то, что было необходимо, программист затем пытается снова. Интерпретирован процесс программирования довольно сильно отличается от скомпилированного

```
1111 0000000000001          Load #1         int a = 1, b = 5, c;
0010 1010100110000          Store a         while(a < b)
1111 0000000000101          Load #5         {
0010 1010100110001          Store b                 c = c + 2;
0001 1010100110000    Top:  Load a                  b = b - 1;
0100 1010100110001          Subt b          }
1000 0000000000000          Skipcond 00     printf("%d", c);
1001 0001000101010          Jump exit
0001 1010100110011          Load c
1110 0000000000010          Add #2
0010 1010100110011          Store c
0001 1010100110001          Load b
1101 0000000000001          Subt #1
0010 1010100110001          Store b
1001 0001000100010          Jump top
0001 1010100110011    Exit: Load c
0101 0000000000000          Output
```

Рисунок 14.1.
Простая программа, написанная на машинном языке (крайний левый столбец), сборка Язык (центральная колонка) и C (крайний правый столбец).

процесса программирования, поскольку это более экспериментальный характер и разработки программы может осуществляться по частям. Составитель программирование часто требует, чтобы вся программа была написана до того, как из него может быть собран и протестирован. Большинство языков высокого уровня, не упал в составленном категорию до недавнего времени. Языки сценариев интерпретируются, и все больше и больше программистов используют языки сценариев. Хотя истолковано программирования звучит как проще и лучше подход, есть большой недостаток истолковано программирования. После компиляции программы, хозяйственная работа языкового перевода закончилась с и, следовательно, выполнение программы требует только время выполнения, а не дополнительное время перевода. Для выполнения интерпретируемый программы, каждая команда сначала транслируется, а затем выполняется. Это приводит к тому времени выполнения, чтобы быть большего времени, чем над подобным скомпилированной программе. Тем не менее, многие современные интерпретируемые языки имеют необязательные компиляторы, так что после завершения, программа может быть составлена при желании.

Другой вопрос, который возникает между обобщаются и интерпретирован программирования является то, что тестирования программного обеспечения. Большинство компиляторов языка будет найти много типов ошибок в время про- грамма компилируется. Они известны как синтаксические ошибки. Как программист (в частности, инженер-программист), не хотел бы выпустить программное обеспечение, которое содержит ошибки. Один из типов ошибок, которые будут пойманы компилятором ошибка несоответствие типов. В этом типе ошибки, программа пытается сохранить тип значения в переменной, которая не должна быть в состоянии хранить этот тип, или инструкция может попытаться выполнить операцию над переменной, которая не допускает такого рода операции. Два простых примера пытаются сохранить дробное значение с десятичной запятой в целой переменной и пытается выполнить умножение на переменную строки. Эти ошибки перехватываются компилятором в то время программа COM- свалили. В интерпретируемой программы, программа не транслируется, пока она не будет выполнена, и поэтому интерпретатор будет только поймать ошибку во время выполнения (ошибка времени выполнения), когда уже слишком поздно, чтобы избежать этой проблемы. Пользователь, запустив программу интерпретирован, можно оставить задаваться вопросом, почему программа не работает неправильно.

Ошибка во время выполнения является тот, который возникает во время выполнения программы. Ошибка во время выполнения происходит потому, что компьютер может буквально не выполнить задачу под рукой. Распространенной формой ошибки времени выполнения является арифметической перегрузки. Это происходит, когда вычисление приводит к значению слишком большой, чтобы поместить в в указанном месте памяти. Другой тип ошибки времени возникает, когда пользователь вводит неуместное значение, например, введя имя, когда число просят.
Ошибка несоответствие типов, описанный выше, который может быть захвачен компилятором. Но если язык не требует компиляции, то ошибка перехватывается только тогда, когда конкретная команда выполняется, и таким образом это происходит во время выполнения.
Третий класс ошибок является логической ошибкой. Этот тип ошибки возникает потому, что логика программиста неверна. Это может быть нечто тривиальное, как вычитания двух чисел, когда программист предназначен для сложения двух чисел. Он может также возникнуть в результате сложнейшие кода, так как программист не смог отслеживать все возможные результаты.
Рассмотрим пример, иллюстрирующий, насколько сложно определить логическую ошибку из кода. Следующий фрагмент кода от C (или C ++ или Java).

INT x = 5; плавать y;
y = 1 / x;

Код сначала объявляет две переменные, целое x и точку (реальный) y плавающей. Переменная x инициализируется с номером 5. Третья инструкция вычисляет 1 / x и сохраняет результат в y. Результат должен быть значение 1/5 (0,2), который хранится y. Это не должно быть проблемой для компьютера, так как y является поплавок, который означает, что он может хранить значение с десятичной точкой (в отличие от x). Тем не менее, проблема с этой инструкцией является то, что в C, C ++ и Java, если числитель и знаменатель являются целыми числами, деление представляет собой целое подразделение. Это означает, что фактор представляет собой целое число, а также. Разделив 1 по 5 дает значение 0 с остатком 1/5. Целое фактор равен 0. Таким образом, y = 1 / x; приводит к y = 0. Так как y поплавка, значение преобразуется от 0 до 0,0. Таким образом, y становится 0,0 вместо 0,2! Логические ошибки чумы почти каждый программист.

Краткое исследование высокоуровневых языков программирования языков программирования, таких как компьютерное оборудование и операционные системы, имеют свою историю. Здесь мы кратко рассмотрим эволюцию языков программирования, concentrat- ING в первую очередь на наиболее популярных языках программирования высокого уровня за последние 50 лет.

Самые ранние языки были родными языками машины. Такой язык будет выглядеть совершенно чуждой нам, потому что инструкции будут содержать только 1S и 0s бинарных, или, возможно, инструкции и данные будут преобразованы в восьмеричном или шестнадцатеричном представлении для несколько легче читаемости. В любом случае, никто не использовал машинный язык в течение многих десятилетий, за исключением студентов, изучающих компьютерные науки или компьютерной инженерией. К середине 1950-х годов, языки сборки были введены. Как и в случае машинных языков, языков сборки очень сложной задачей.

Начиная с конца 1950-х годов, программисты начали разработку компиляторов. Компиляторы языка могли переводчики, которые переводят языки программирования высокого уровня на машинный язык.

Разработка компиляторов была рука об руку с развитием первых языков программирования высокого уровня. Самый ранний язык высокого уровня, Фортран (Фортран) была предпринята попытка разрешить программисты, которые имели дело с кал и математик научных расчетов средства, чтобы выразить программу в виде ряда математических формул (отсюда название языка) вместе с входным сигналом и операторы вывода. Большинство программистов были настроены скептически, что компилятор может производить эффективный код, но как только это было показано, что компиляторы могут генерировать машинный язык код, который был столь же эффективным, как, или более эффективными, чем, что произведенный человеком, FORTRAN стал очень популярным.

FORTRAN, будучи первым языком на высоком уровне, содержит много особенностей, обнаруженных на СБОРОЧНЫЕ языке и не содержат много функций, которые будут найдены в более поздних языках. Одним из примеров является зависимость Fortran на неявной переменной набора текста. Вместо того, чтобы требовать, чтобы программист объявлять переменные по типу, тип переменной будет основываться на имени переменной. Переменная с именем, начинающимся с любой буквой от I до N было бы целое, а любой другой переменной, была бы реальная

(число с десятичной точкой). Кроме того, FORTRAN не содержат тип символов, так что вы не могли хранить строки, только цифры. Имена переменных были ограничены шестью символами в длину. Вы не могли бы, таким образом, назвать переменную income_tax.

FORTRAN не было полезным, если-заявление, полагаясь вместо этого на использование GO TO операторах (тип безусловной инструкции ветвления). Безусловные ветви широко используются в машиностроении и сборки языка программирования. В Фортране GO TO позволяет программисту передавать управление программой из любого места в другое. Мы рассмотрим пример этого ниже в этом разделе. Из-за GO TO, а не полезно, если-заявление, FORTRAN код будет содержать логику, которая была гораздо более сложным (или запутанным), чем это было необходимо. Несмотря на свои недостатки, FORTRAN стал чрезвычайно популярным для научного программирования, потому что это был огромный шаг вперед по сравнению сборки и машинный язык.

Как развивается FORTRAN, Министерство обороны начал работать с бизнес-программистов для создания бизнес-ориентированный язык программирования, COBOL (Кобол). В то время как FORTRAN часто выглядел так же, как математической нотации, была выражена COBOL в английских предложений и абзацев, хотя слова, составлявшие приговоры должны были быть юридические заявления, COBOL, написанные в правовой синтаксис COBOL. Идея программы COBOL должен был писать небольшие процедуры, каждая из них будет свой собственный пункт. Подпрограммы будет вызывать другие подпрограммы. данных программы будет подробно описано, при условии, что данные будут поступать из базы данных, и выход будет помещен в другой базе данных. Таким образом, большая часть программирования COBOL было движение и обработка данных, в том числе простых математических расчетов и сортировке. COBOL был выпущен несколько лет после того, как FORTRAN и стал таким же успешным, как FORTRAN, хотя в случае Cobol, он был успешным в бизнес-программировании.

COBOL также не хватало многих функций, доступных в более поздних языках. Одним из главных недостатков было то, что все переменные были глобальными переменными. Программы, как правило, делятся на несколько более мелких подпрограмм, которые иногда называют также процедур, функций или методов. Каждая процедура имеет свое собственное пространство памяти и переменные. Таким образом, программа может манипулировать локальными переменными, не заботясь о том, что другие функции могут изменить их. Глобальные переменные

нарушают это идея, потому что переменная известна в одном месте программы, или одна процедура, известна по всей программе. Создание изменения переменной в одной процедуре может иметь неожиданные последствия в других подпрограмм.
COBOL страдал от других проблем. Рано COBOL не включает понятие массива. Там не было передачи параметров в начале COBOL (отчасти потому, что все переменные были глобальными переменными). Как FORTRAN, COBOL имел зависимость от безусловных инструкций ветвления. Кроме того, COBOL был ограничен по отношению к своим вычислительным способностям. COBOL сделал, однако, ввести два очень важных понятия. Первым было понятие структурированных данных, теперь упоминается как структуры данных, а вторая была возможность хранить строки символов. Структуры данных позволяют программисту определить переменные, которые состоят из различных типов данных. Например, мы можем определить запись студента, который состоит из имени студента, майор, GPA, всего кредитных часов заработал, и текущие курсы принимаются. Различные типы данных будут представлены с использованием различных типов данных: строки, вещественных чисел.
Исследователи в области искусственного интеллекта начали разрабатывать свои собственные языки в конце 1950-х и начале 1960-х годов. Исследователи призвали к языку функции недоступными в любом FORTRAN или COBOL: обработка списков символов, рекурсии и динамической памяти размещения. Несмотря на несколько языков изначально были разработаны, это был LISP-язык-что Лисп прижился. В дополнение к вышеупомянутым особенности, которые сделали LISP + граммы можно было бы записать в разрозненно. LISP был очень инновационным и продолжали использоваться в различных формах, через 1990-е годы.
Исходя из этих трех языков, язык Алгол (алгоритмический язык) была предпринята попытка объединить лучшие черты FORTRAN, COBOL и LISP на новый язык. Алгол в конечном счете, получил большую популярность в Европе, чем в Соединенных Штатах, и его использование было ограничено по сравнению с другими языками, в конечном итоге приводит к гибели Алголом в. Тем не менее, Алгол будет играть значительную роль, как будут разработаны новые языки.
В 1960-е годы, языки программирования были разработаны для многих различных применений областях катионов. Новые языки также были разработаны для замены старых языков или про- Vide объекты, которые более старые языки не включают. Некоторые известные языки в начале

1960-х годов были PL / I, язык, разработанный IBM, чтобы содержать в себе лучшие черты FORTRAN, COBOL и ALGOL в то же время введения новых понятий, таких как обработка исключений; SIMULA, язык моделирования для разработки; SNOBOL (Струнные Oriented и символическом языке), язык строка соответствия; и BCPL, ранняя версия C.

К концу 1960-х годов, программисты поняли, что эти ранние языки полагались слишком сильно на безусловных ветвлений заявлений. Во многих языках, в заявлении был назван GO TO (или Goto). Безусловный переход позволит программисту указать, что программа может перейти от текущего местоположения в другом месте в программе.

В качестве примера рассмотрим программу, которая случается быть 10 страниц в длину. На странице 1, есть четыре инструкции, за которыми следует GO TO, который ответвляется к инструкции на странице 3. На этом этапе есть пять инструкции следуют в если-то-иначе заявления. Тогда оговорка имеет оператор GO TO, что переходит к инструкции на странице 6, и п-то есть GO TO заявление, что переходит к инструкции на странице 8. На странице 6, есть три инструкции последовательности заявлений GO TO на странице 2 на стр. 8, есть пять инструкции следуют в если-то заявление. То предложение имеет заявление GO TO, что ветви на страницу 1. И так далее. Попытка разобраться в программе требует отслеживания через него. Трассировка с помощью программы с операторами GO TO приводит к путанице. След может начать походить на кучу спагетти. Таким образом, безусловные ветвящиеся предписания могут создать то, что было названо код спагетти.

Раннее FORTRAN использовал If-заявление, которое было построено на GO TO заявлений. Основной формой заявления является (арифметическое выражение) линия number1, линия number2, линия номер3. Если арифметическое выражение имеет значение отрицательное значение, перейти к строке number1, иначе, если арифметическое выражение равно нулю, перейти к строке number2, в противном случае перейти к строке number3. Смотри рисунок 14.2, который иллюстрирует спагетти код, который слишком легко производится в FORTRAN. Вторая инструкция является примером условного оператора на Fortran, который вычисляет I-J, а если отрицательный, ветви к 10-й строки, если ноль, ветви к линии 20, в противном случае переходит к строке 30.

Полагая I = 5, J = 15, и K = -1, вы можете проследить через код на рисунке 14.2, чтобы увидеть, какие инструкции выполняются и в каком порядке? Если ваш ответ нет, это не может быть из-за громоздкости трассировки

с помощью кода, который использует GO TO заявления в такой степени. Программисты, которые были развивающиеся новые языки решили, что безусловный переход был слишком неструктурированным. Их решение должно было разработать более структурированные операторы управления. Это привело к развитию условных операторов итерации (например, в то время как петли), считая петли (для петель), и вложенные если-то-иначе заявления. Два языка, которые приняли концепцию структурного программирования, были разработаны как ответвление АЛГОЛа: C и Pascal. На протяжении более 15 лет, эти два языка будут широко использоваться в программировании (C) и образования (Паскаль). Разработанная в начале 1980-х годов, Ада будет взять на себя особенности обоих языков, включать особенности, найденные в различных других языках, вводить новые функции, и стать язык, используемый исключительно правительством США в течение десятилетий.

В 1980 году новая концепция в программировании возникла, объектно-ориентированного программирования. Разработанный первоначально в одном из вариантов Симула, первый объектно-ориентированное программирование язык был Smalltalk, который был результатом диссертационного исследования студента. Идея заключалась в том, что структуры данных может моделировать объекты реального мира (будь то физические объекты, такие как автомобиль или абстрактные объекты, такие как окна операционной системы). Программа может состоять из

```
        READ (*,*) I, J, K
        If (I-J), 10, 20, 30
10      If(K) 30, 40, 50
20      I=J*K
        If(I-25) 40, 50, 60
30      J=K*3
        GO TO 20
40      K=K+1
        GO TO 10
50      WRITE (*,*) I, J, K
        GO TO 70
60      WRITE (*, *) J, K
70      END
```

Рис. 14.2
Неструктурированная программа в Фортане

объектов, которые будут взаимодействовать друг с другом. Например, если бы нужно было запрограммировать объект рабочего стола и несколько объектов окна, рабочий стол может отправлять сообщения в окно, чтобы закрыть себя, чтобы переместить себя, чтобы уменьшить себя, или изменить размер себя. Объект будет указано в определении класса, описывающего как структуру данных объекта и весь код, необходимый для обработки сообщений, которые другие объекты могли бы пройти к нему. Программирование изменяется от идеи программных подпрограммами называть друг друга программировать объекты отправки сообщений в список друг к другу.

В конце 1980-х годов, C был повышен до C ++ и LISP был повышен до Common Lisp, две новые языки являются объектно-ориентированными. позже Ада была увеличена до Ada 95. А потом, основываясь на C ++, Java был разработан. Java, в то время, был уникальный язык в том, что он был составлен как и интерпретированы. Об этом более подробно обсуждается ниже. С огромной значимости народно как C ++ и Java, почти все языки с 1990-х был объектно-ориентированным. Другая концепция введена в течение этого периода было то, что визуального программирования. В визуальном языке программирования, программы с графическим интерфейсом могут быть легко созданы с помощью метода перетаскивания и падение развивать сам GUI, а затем запрограммировать "за кадром" код для обработки операций, когда пользователь взаимодействует с компонентами GUI , Например, графический интерфейс пользователя может содержать две кнопки и текстовой строки. Программист создает графический интерфейс пользователя, просто вставив кнопки и текстовой строки на пустую панель. Затем программист должен осуществить действия, которые должны произойти, когда при нажатии кнопок или, когда текст вводится в текстовом поле.

В 1990-е годы начало приватизации Интернета. Компании было разрешено продавать частным лицам возможность получить доступ в Интернет из своих домов или офисов. С этим и появлением World Wide Web, новая концепция в программировании была пионерски внедрена. Раньше, чтобы запустить программу на вашем компьютере, вы должны получить компьютерную версию. Программист, возможно, придется найти компилятор для каждой платформы так, что программное обеспечение может быть доступна на каждой платформе. Если программист написал C ++ про- грамм, программист должен был бы составить программу для Windows, для Macintosh, Sun для рабочих станций, для Intel на базе Linux для IBM мэйнфреймы, и так далее. Мало того, это потребует от

программиста, чтобы получить несколько различных компиляторов, он может также потребовать, чтобы программист модифицировать код перед использованием каждого компилятора, так как составители ERS могли бы ожидать немного другой синтаксис.

При разработке языка программирования Java, авторы пытались что-то новое. Компилятор Java будет транслировать программу Java в промежуточный язык, который они назвали байт-код. Байт-код не будет работать на компьютере, так как это было не машина какой язык сам. Но байт-код не будет зависеть от каждой платформы. Компилятор не будет производить программный код одного байта независимо от того, какая платформа программа была предназначена для запуска на. Далее, авторы Java реализован ряд виртуальных машин Java (JVM), по одному на платформе. Ответственность виртуальная машина была бы взять, один за другим, каждую команду байт кода программы, преобразовать его в машинный язык платформы, и выполнить его. Таким образом, виртуальная машина является переводчиком. Ранее было заявлено, что интерпретация программы гораздо менее эффективна, чем составление программы, которая является истинным. Тем не менее, интерпретации байт-код может быть сделано почти так же эффективно, как работает скомпилированную программу, потому что байт-код уже частично переведен команда программы. Таким образом, эта комбинация и интерпретации программы почти столь же эффективна, как работает уже скомпилированную программу. Преимуществом такого подхода является то, что Java переносимости; программа скомпилирована компилятором Java может работать на любой платформе, которая имеет JVM. JVMs были реализованы практически во всех веб-браузеров, так что Java является языком обычно используется для реализации программ, которые запускаются в веб-браузеры (такие программы часто называются апплеты).

Стремление к платформ независимых программ продолжает увеличиваться с течением времени с ростом популярности Интернета. Сегодня .net от Microsoft ("точка сети") программирования платформы позволяет подобный подход к байт-кодов Java в этой компиляции кода в байт-код, а затем могут быть интерпретированы. .Net Языки программирования включают в себя C # (разновидность C ++ и Java), ASP (Active Server Pages), Visual Basic (визуальный языка программирования), C ++, и вариант Java под названием J ++. Платформа .Net выходит за рамки природы независимой от платформы Ява в этом коде, написанном на одном языке платформы может использоваться код, написанный на

другом одном из языков. не Таким образом, программа больше не должна быть написана на одном языке, но вместо этого может быть составлен из отдельных классов, написанных на ряде этих языках.
Неизвестно, какие новые возможности будут добавлены языки в будущем, или где будущее программирования перенесет нас относительно новых языков. Но многим в науке компьютеру, долгосрочная цель состоит в том, чтобы программировать компьютеры, чтобы понять естественные языки. Естественный язык является языком, который люди используют, чтобы взаимодействовать друг с другом. Если компьютер мог понимать английский язык, то программист не придется беспокоиться о написании кода в пределах какого-либо одного языка программирования, ни бы пользователь компьютера должен узнать синтаксис позади команд операционной системы, такие как Linux и DOS. Естественные языки, однако, изобилуют двусмысленности, поэтому программирование компьютера, чтобы понять естественный язык остается очень сложной задачей.

Типы инструкций

Программный код, независимо от того, на каком языке, не содержит те же типы инструкций. Эти типы инструкций описаны в данном разделе наряду с примерами на различных языках.

Входные и выходные Инструкции

Ввод является процессом получения значений из устройств ввода (или файлов) и сохранения значений в переменных в памяти. Выход это процесс отправки буквенные значения и значения переменных на устройства вывода (или файлов).
Некоторые языки имеют два отдельных набора ввода / вывода отчета: те, которые используют стандартный ввод (клавиатура) и выход (монитора), а также те, которые используют файлы. Другие языки требуют, чтобы оператор ввода или вывода укажите устройство источника / назначения. В C, C ++ и Java, например, существуют различные утверждения для стандартного ввода, стандартного вывода входного файла и выходной файл. В FORTRAN и Pascal, входными и выходными данными отчетности по умолчанию в стандартном входе и выход, но может быть изменено, чтобы указать файл. Кроме того, некоторые языки используют разные операторы ввода / вывода для различных типов данных. В Ada

существуют различные входные менты для Утверждения строк, для целых чисел, для чисел и так далее. Большинство языков позволяют несколько переменных параметров для ввода или вывода в одном операторе.

Рисунок 14.3 показывает инструкции ввода файла с четырех языков: FORTRAN, C, Java и Pascal. В каждом случае, кодовые входы трех данных x, y, r (целое и два действительных или чисел с плавающей точкой) -из текстовый файл input.dat. Каждый набор кода требует больше, чем просто самой команде ввода.

Код FORTRAN Первое требует открытое заявление, чтобы открыть текстовый файл для ввода. Файл получает обозначение 1, который используется во входном заявлении, READ. Оператор READ довольно прост, хотя (1, 50), обратитесь к файлу целеуказательно и в формате строки, соответственно. Оператор FORMAT описывает, как вход должен быть интерпретирован. В этом случае первые два символа рассматриваются как целое (I), а следующие шесть символов, рассматриваются в качестве плавающей запятой (реального) числа с двумя цифрами справа от десятичной точки (F6.2) с последующим другим поплавком с тем же форматом.

В случае C, первый оператор объявляет переменную к типу FILE. Затем файл открыт с FOPEN заявлением. "Г" указывает на то, что файл должен

```
   OPEN(1,FILE='input.dat', ACCESS='DIRECT', STATUS='OLD')
   READ(1,50) X, Y, Z
50 FORMAT(I2, F6.2, F6.2)


FILE *file;
file = fopen("input.dat", "r");
fscanf("%d %f %f", &x , &y, &z);


var input : text;
…
Assign(input, 'input.dat');
Reset(input);
Readln(input, x, y, z);


try {
       BufferedReader input = new BufferedReader(new FileReader("input.dat"));
       data=input.readLine( );
       StringTokenzier token = new StringTokenizer(data, " ");
       x=Integer.parseInt(token.nextToken( ));
       y=Float.parseFloat(token.nextToken( ));
       z=Float.parseFloat(token.nextToken( ));
}
catch(IOException e) { }
```

Рис. 14.3
Пример ввода файла в четырех языках программирования

быть считан из, но не записывается. Обозначение "% d% F% F" указывает на то, что вход должен рассматриваться как значение десятичного (целое число) и двух значений с плавающей точкой. И символ, используемый в C до переменных в операторе Scanf, требуется из-за того, как C передает параметры своих функций.

В языке Pascal, то текстовый файл должен быть объявлен как тип текста. Затем переменная текст присваивается имя файла и открыт. И, наконец, значения ввода. Паскаль, наименее мощный из этих четырех языков, является самым простым. На самом деле, Паскаль был создан с простотой в виду.
В Java, любой вход должен быть обработан, что известно как обработчик исключений. Это делается через блок примерочных систем (хотя Java дает вам другие механизмы для обработки этого). Далее, переменная типа BufferedReader необходимо для выполнения ввода. Входной сигнал от BufferedReader обрабатывается как одна длительная строка. Строка должна затем быть разбита на части компонента (три числа). Строка токенизатор делает это. Кроме того, результаты токенизатор представляют собой строки, которые должны быть преобразованы в числовые значения с помощью утверждений синтаксического анализа. Если исключение возникает при попытке выполнить вход, блок поймать выполняется, но здесь, блок поймать ничего не делает.

Операторы присваивания

Оператор присваивания сохраняет значение в переменной. Переменная находится на левой части оператора, а значение в правой части оператора. Разделение двух частей является оператором присваивания. Типичная форма оператора присваивания = или: = (Pascal и Ada). Сторона значение правая не должно быть буквенное значение, но часто некоторые формы выражения. Есть три общие формы выражений: арифметические, реляционные и Boolean. Арифметическое выражение содержит числовые значения, переменные и арифметические операторы, такие как * Ь - 3. В этом случае, а и Ь являются переменными, предположительно, хранящие числовые значения. Арифметические операторы +, -, *, /, и по модулю (деление с получением Остальной дер), часто обозначается с помощью%. Реляционные выражения используют операторы отношения (меньше, равно, больше чем, больше чем или равные, равно, и т.д.) для сравнения значений. Булевы выражения использовать AND, OR, NOT, XOR. Там

Таблица 14.1 Назначение данные и равенство Операторы на нескольких языках

Язык	Оператор присваивания	Оператор равенства	Оператор не равно
Ада	: =	=	/=
C / C ++ / Java, Python, Ruby	=	= = (2 знака равенства)	! =
COBOL	Присвоение	Равно (или =)	не равно (или нет =)
FORTRAN	=	.EQ.	.NE.
Pascal	: =	=	<>
Perl	=	экв (строки), = = (число)	пе (строки),! = (номера)
PL / I	=	=	<>

также может быть строковые выражения с использованием функций, таких строк как конкатенация и вызовы функций, которые возвращают значения, такие как SQRT (y), чтобы получить квадратный корень y. Вызовы функций обсуждаются в подпрограммах и подпрограммам.

Так как знак равенства (=) часто используется для присвоения, какой символ этого языка используют для проверки равенства (т.е. "делает равным Ъ")? Мы обычно используем знак равенства в математике. Если язык программирования использовать один и тот же символ для двух целей? Если да, то это не только может запутать программиста, но это будет путать компилятор, а также. Вместо этого используйте языков два различных набора символов для присваивания и равенства. Таблица 14.1 обеспечивает сравнение на нескольких языках. Он также показывает, что "не равно" оператор. Обратите внимание на то, что PL / I является единственным языком, который использует = для обоих присваивания и равенства.

Оператор присваивания в большинстве языков присваивает одну переменную значение. Некоторые языки, однако, позволяют несколько назначений в одной команде. В C, например, x = y = r = 0; устанавливает все три переменные в 0. В Python, вы можете назначить несколько переменных, разделяя их на левой стороне с запятыми, как в x, y = 5, x + 1, который присваивает x к значению 5, а затем присваивает y к значению x + 1 (или 6). Этот ярлык может уменьшить количество команд

в программе, но может сделать программу менее читабельным. С также имеет сокращенные операторы. К ним относятся использование ++ для увеличения переменной, как в X ++, которая является такой же, как x = x + 1, и
+ =, Которая позволяет записать x + = y вместо x = x + y. С также допускает назначение в пределах выражения части оператора присваивания. Утверждение a = (Ь = 5) * c; два оператора присваивания в одном. Во-первых, б устанавливается равным 5, а затем устанавливается равной Ь * С использованием нового значения Б. Если язык компилируется и переменные объявляются, то компилятор может проверить несоответствия типов. Этот тип ошибки будет возникать, если правая сторона об уступке генерирует значение, которое не совместимо с переменной на левой стороне. Например, если x является целой переменной, то оператор присваивания x = 3,1415 * y даст ошибку несоответствия типов, так как правая часть включает в себя реальную (с плавающей точкой) число.
Это справедливо независимо от того, какой тип у есть.

Заявления выбора

Условиями являются выражения, которые вычисляются как истина или ложь. Условия используются в программах, чтобы определить, что программа должна делать дальше. Они используются в двух типах заявлений, заявлений выбора и итерации (или петли) отчетности (рассматриваемые в итерационных отчетности). Идея условием является то, что значение, которое хранится в одной или нескольких переменных сравнивается. Сравнения могут быть основаны на реляционных операторов, а также может использовать логические операторы и арифметические операции. Например, условие может проверить, имеет ли x больше y или x> y. Другое условие может проверить, если x равно y или x равен г. Разные языки используют символы разных для реляционных операторов (меньше, больше чем, равно, и т.д.) и логические операторы (и, или, нет).
Оператор выбора решает, следует ли выполнять или пропускаются на основе оценки состояния ряда инструкций. Этот тип заявления часто называют если-то заявление (или просто если заявление). В заявлении если-то, если условие истинно, то часть заявления (обычно называемый затем пункт) выполняется. Если условие ложно, то пункт пропускается.
В если-то-другое заявление, если условие истинно, то предложение выполняется, и если условие ложно оговорка еще выполняется. Вы

бы использовать, если-то заявление, если вы только хотели код для выполнения, если условие истинно. Это иногда называют выбор односторонним. Вы бы использовать если-то-другое заявление, если вы хотите, чтобы выполнить код, независимо от того, в каком состоянии оказался. Это иногда называют выбор двухсторонний.
Рисунок 14.4 содержит пример кода на трех языках: Pascal, C и FORTRAN IV (ранняя версия FORTRAN). Каждый пример имеет той же логической задачи, добавляя 1 к x, если x равен y или x равен r, и вычитанием 1 из x в противном случае. Код Паскаль легчайшего понять. В C, два знака равенства составляют «равенство» оператора и двух вертикальных бары (||) составляют оператора ИЛИ. Код FORTRAN, которая опирается на GO TO отчетности, является менее интуитивным. Читайте через код и посмотреть, если вы можете понять это. Можно заметить, что в C, то слово ", затем" опущено и условие помещается внутри скобок.

Если-то-иначе оператор выбирает между двумя наборами кода, «один вещь или другой" ситуации. Что делать, если там было больше, чем просто два возможных набора кода, чтобы выбрать между ними? Большинство языков позволяют тогда пункт и другое-пункт, чтобы содержать если-то, и если-то еще заявления тогдашний. Это создает то, что называется вложенным, если-то-другое заявление. На рисунке 14.5 показан пример на языке Паскаль, демонстрируя, как определить, какая буква класса, чтобы назначить на основе числового класса студента. Мы используем

```
If x = y or x = z then
x := x + 1
Else
x := x - 1;


if ( x = = y || x = = z)
       x = x + 1;
else
       x = x - 1;


       IF ( X .EQ. Y .OR. X .EQ. Z) GO TO 10
       X = X - 1
       GO TO 20
10     X = X + 1
20 ….
```

Рис. 14.4.
Три примера операторов if–then–else (если-то-иначе)

типичный 90/80/70/60 пробой. Исходя из условия, что соответствует, код присваивает переменной буква одно из следующих значений: 'A', 'B', 'C', 'D' или 'F'.
Обратите внимание на то, что последнее утверждение не требует, если. Идея заключается в том, что если первое условие истинно, то первый пункт, то выполняется и весь вложенный оператор заканчивается; в противном случае, первый пункт выполняет еще. Этот раздел еще содержит если-то-другое заявление. Для этого второго условия, если это правда, то второй пункт, то выполняется и оператор заканчивается; в противном случае, второй пункт еще выполняет. Этот раздел еще содержит также если-то-другое заявление. Если выполнено условие, что пункт (в третье условие) истинно, то третий пункт, то выполняется и оператор заканчивается; в противном случае, третий еще пункт выполняет. Третий раздел содержит еще одно заявление. Если это четвертое условие истинно, то четвертый пункт, то выполняется и оператор заканчивается; в противном случае, конечный пункт выполняет еще.
Несколько других замечаний стоит упомянуть здесь. Углубление Приведенную выше помогает сделать код более удобным для чтения, но не влияет на точность Кодекса или характеристик, что были достигнуты. То есть, все вкладки, вставленные игнорируются компьютером. Кроме того, капитализация не имеет значения. И, наконец, вы можете заметить, что весь набор кода содержит один с запятой в конце. Одно из различий между Pascal и C, C ++ и Java является использование Semicon до конца заявлений. В языке Pascal, только один используется, чтобы закончить всю структуру.
Обратите внимание, в приведенных выше примерах, что тогда и остальное пункты состояли из отдельных операций. Например, в примере, показанном на рисунке 14.4, каждое предложение сделал одно: инкремент x или декремента x. Это так же, как часто бывает, что положение будет содержать несколько инструкций. В языке Pascal, например, вы можете иметь код следующим образом:

```
Если x> = y
Тогда x: = r;
Y: = 0;
д: = x + y;
```

Отступов и другие пробелы игнорируются, так как знает, что компилятор, когда то пункт заканчивается? То есть, есть д: = x + y; часть тогдашнего

пункта? На самом деле, это не так, и ни один не y: = 0; даже если расстояние подразумевает, что оно есть. В таких языках, как Pascal и C, компилятор только ожидает, что оговорка, чтобы иметь одну команду. Если вы хотите указать несколько инструкций, необходимо поместить их в блок. Блок обозначает набор инструкций, которые следует рассматривать как единое целое. Это позволяет вам иметь несколько инструкций в то или другое п. В Паскале, блок обозначается с помощью слов начинаются и заканчиваются, тогда как в C, блок обозначается с помощью символов {и}. Правильные версии Pascal и C вышеуказанного заявления, если-то показаны ниже. Вы можете заметить, что y: = 0 не заканчивается точкой с запятой. Паскаль правила точка с запятой несколько сложнее.

Если x> = y Тогда, если (x> = y) {Начало X = Z;
x: = r; y = 0;
Y: = 0}
Конец; д = x + y;
д: = x + y;

В некоторых языках, таких как Ada, явное ENDIF заканчивается заявление. В начале FORTRAN вместо этого, вы обозначили конец положений на основе заявлений GO TO. Мы увидим возвращение блока, когда мы получаем к итерации заявления и в подпрограммах и подпрограммам. Ниже приведен тот же код из выше, но на этот раз написана на Python. Python является уникальным языком при выравнивании значения. Обратите внимание, как нам не нужны индикаторы блока (начинаются и заканчиваются или {и}), но вместо этого, отступы указывает на то, что обе инструкции после того, как если x> = y: являются частью тогдашнего пункта. Без отступа для последнего утверждения, то это означает, что она не является частью тогдашнего пункта.

Если x> = y:
x = r y = 0
д = x + y

Давайте посмотрим на другой пример вложенного if-другое заявление, на этот раз в C. Вы увидите, что есть три различных положения, которые могут выполняться на основе значений x, y, и
г. Указанные значения x, y и r, вы должны быть в состоянии определить, какой из трех положений выполняет. Например, если x = 5, y = 3 и r = 0,

что бы выполнить пункт?

если (x> 0)
если (y> 0)
... // Пункт 1, если еще (r> 0)
... // Пункт 2
еще ... // пункт 3

Пункт 1 будет выполняться, если x> 0 и y> 0. Пункт 2 не будет выполняться, если x> 0, y <= 0 и r> 0 (если x <= 0, мы никогда не достигают второй if-оператор, и поэтому мы никогда достичь еще if- заявление, а если y> 0, то мы никогда не достичь другого, если-заявление). Когда пункт 3 выполнить? То есть, это последнее условие еще спаренный с условием (r> 0) или условие (x> 0)? Он не в паре с (y> 0), так как иначе, если заявление содержит положение еще для этого условия.
В данном примере кода, не ясно, если последний еще связан с первым, если или третий если. Это ситуация, известная как еще зависшего. Если бы мы должны были использовать блок обозначения, мы могли бы легко решить эту проблему в качестве одного из следующих действий:

если (x> 0) {если (x> 0)
если (y> 0), если (y> 0) {
... // Пункт 1 ... // пункт 1
} Еще если (r> 0)
иначе, если (r> 0) ... // пункт 2
... // Пункт 2}
еще ... // пункт 3 еще ... // пункт 3

В коде на левой стороне, последнее еще идет со вторым, если заявление (y> 0), а в коде справа, последний еще идет с первым, если заявление (x> 0). Тем не менее, в C, мы не должны использовать блок обозначения, так как, по умолчанию, любой другой пункт оборванных всегда подсоединенных с самого последнего состояния. Таким образом, пункт 3 выполняется, если x> 0 и y <= 0 и r <= 0. Если мы не хотим, по умолчанию, чтобы применить (случай, справа вверху), мы бы затем использовать блок обозначения. Различные языки имеют разные правила относительно оборванных либо еще при операторах. Кстати, если x = 5, y = 3 и r = 0, пункт 3 будет выполняться.
Существуют и другие формы нескольких операторов выбора. В Паскале,

они называются заявления случая. В C и Java, они называются операторы коммутатора. Мы не будем рассматривать их здесь, хотя вы бы видеть их в любом вводном курсе программирования.

Итерационные заявления

Есть моменты, когда набор кода должен выполняться несколько раз. Рассмотрим в качестве примера, который вы хотите написать программу расчета заработной платы, которая будет выполнять операции начисления заработной платы. Программа должна вводить данные работника, вычислить заработной платы и налогов работника, вывода результатов, а также поддерживать итоги повсюду. Вы можете подумать, чтобы написать программу в виде прямой линии кода, который выполняет входные шаги, вычисления и выходы, а затем завершается. Это было бы вычислить заплатить один работника. Вы не хотели бы, чтобы запустить программу один раз на одного работника, потому что это будет раздражать держать перезапуска программы. Кроме того, вы не смогли бы вычислить итоговые данные всех сотрудников. Вместо этого можно было бы написать "прямой линии" код и поместите его в инструкцию итерации. Оператор итерации используется, чтобы повторить набор кода. Существуют две основные формы итерации заявления. Условное, или логический, цикл повторяется код, пока условие вычисляется как истина. Цикл подсчета (часто известный как петля) перебирает несколько раз либо на основе начальной и конечной точки (например, от 1 до 10), или значения, сохраненные в списке. Причина того, что подсчет петли иногда называют для петель является то, что большинство языков используют зарезервированное слово для, чтобы указать, что это цикл подсчета. Ниже приведены примеры Паскаля кода. Во-первых, мы видим логический цикл.

```
В то время как x> y
начать
...
конец;
```

В предыдущем цикле, условие x> y. Это означает, что цикл будет продолжать выполняться, в то время как x остается больше, чем y. Обратите внимание на использование блока начать ... конец. Как и в случае если-то и если-то-иначе утверждений, если код то время цикла

(в известном как тело цикла) состоит из более чем одной команды, использование блока требуется. В приведенном выше цикле, как только у больше не меньше, чем x, цикл завершается и программа продолжается со следующей инструкции после окончания заявления. В C, тот же цикл будет выглядеть следующим образом:

```
в то время как (x> y) {
...
}
```

Для контура может быть использовано в одном из двух способов. Во-первых, цикл повторяется через ряд значений, обозначаемых в качестве исходного значения и значения, завершающего единицами называется размер шага. Например, мы могли бы указать цикл от 1 до 11 на 2s (цикл будет итерацию путем подсчета 1, 3, 5, 7, 9, 11). Во-вторых, перебирает цикл по списку. Например, если переменная хранится список значений, скажем x = {1, 10, 25, 39, 44}, то цикл будет повторять пять раз, один раз на одно значение в списке. В любом случае, цикл содержит индекс цикла. Эта переменная хранит значение, которое в настоящее время итерации. В цикле подсчета, индекс будет принимать по каждому значению 1, 3, 5, 7, 9, 11, а также в цикле списка, он будет принимать по каждому значению списка (1, 10, 25, 39, 44). Цикл ниже цикл подсчета. Индекс цикла I принимает значение 1 в течение первой итерации, 2 во второй итерации, 3 во время третьего, и так далее до тех пор, пока не достигнет
10, ее окончательный итерации.

```
для I: = 1 до 10 начинают делать
...
конец;
```

Индекс цикла можно ссылаться в теле цикла. Вы можете использовать это значение как часть вычислений. Следующий код не только перебирает от 1 до 10, но добавляет текущий индекс цикла в текущей суммы. Затем этот код суммирует значения от 1 до 10 (то есть, он вычисляет 1 + 2 + 3 + ... + 10). Это известно как цикл суммирования.

```
Сумма: = 0;
Для I: = 1 до 10 делать
Сумма: = Sum + I;
```

Обратите внимание на то, что нам не нужно, помещать тело цикла внутри начать ... конец заявления, потому что тело цикла состоит из одной команды. Тот же самый код приведен ниже в C. C для цикла выглядит странно, по сравнению с относительно простой Pascal циклом. В C, первый пункт в скобках инициализирует любой индекс цикла (ов), второй содержит подключаемым условия Aco, а третий пункт выполняет шаг увеличения или уменьшения значения индекса.

```
сумма = 0; для (я = 0; г <= 10; I = I + 1)
сумма = сумма + I;
```

C на самом деле предоставляет ряд ярлыков. Одним из которых является утверждение I ++, который может быть использован вместо I = I + 1 Другой сумма + = I, которые могут быть использованы вместо суммы = сумма + I. Ниже приведен для цикла (только оператор цикла, а не тело цикла), который будет перебирать вниз, а не вверх. для (I = 10;> 0; i--). Одно из отличий между C и Pascal является то, что C может варьировать свой размер шага (например, итерацию по 2s или 5s), в то время как Паскаль может только итерацию вверх или вниз по те.
Некоторые языки предлагают различные формы обоих условных и подсчета петель. C и Java имеют как в то время как петли и делать-то время как петли. Разница между ними в том, где проверяется условие. В то время цикла, условие проверяется перед входом в тело цикла, тогда как в цикле Do-в то время как, условие проверяется после выполнения тела цикла. Хотя это может показаться, что то же самое, цикл Do-то время проверяет условие после выполнения тела цикла, так что тело цикла выполняется, по крайней мере, один раз. Если выполнено условие, то время цикла является изначально ложно, то тело цикла не выполняется вообще. Ниже приведен пример, который сравнивает две петли в C.

```
в то время как (x> 0)
x = x / 2;

делать {
x = x / 2;
} В то время как (x> 0);
```

Предположим, в обоих из этих петель выше, что x представляет собой целое число первоначально равно -1. В первом цикле, условие изначально ложно, и поэтому тело цикла (x = x / 2) никогда не выполняется. Таким образом, остается x -1. Во втором цикле, тело цикла выполняется один раз, так как условие не проверяется до тех пор, после того, как тело цикла выполняется, так что x изменяется от -1 до 0 (значения -1/2 не могут быть сохранены в целое число, так что вместо x получает значение 0). Затем цикл завершается, потому что x не больше 0.
Новые языки часто отказываются цикл подсчета в пользу цикла итератора. Эта версия петли подсчета итерацию по списку, а не по последовательности значений. Например, если у вас есть список, 1, 5, 10, 19, то цикл будет итерацию четыре раза, один раз для каждого из четырех значений в списке. Python является примером, где цикл перебирается в список. Вы можете использоваться этим типом цикла для имитации цикла подсчета, что перебирает в диапазоне от 1 до верхнего предела, заявив диапазон (1, 100). C ++ и Java изначально была только счетную форму для цикла, но сегодня оба подсчета и итераторов петли.
Одно последнее замечание относительно условных циклов является то, что тело цикла должно содержать инструкции, которые манипулируют, по меньшей мере, одной из переменных, используемых в условиях цикла;. Рассмотрим следующий код на C.

```
в то время как (x> y)
x = x + 1;
```

Если x изначально больше, чем y, тело цикла будет выполняться. Тем не менее, так как x начинается при значении больше, чем y, последовательно добавляя к x никогда не будет делать x меньше или равно y, а результат в том, что цикл никогда не прекращается. Это известно как бесконечный цикл.
Для того, чтобы убедиться, что вы не имеете бесконечный цикл, вы должны убедиться, что есть код в теле цикла, который изменяет некоторое значение в состоянии, так что, в конце концов, условие может стать ложным. Рассмотрим следующий код C.

```
в то время как (x> 0)
E ( "% d \ n", x); // Вывод x x = - 1;
```

Этот цикл выглядит, как он будет отсчитывать от того, что x начинается с

0, вывод на печать каждого значения. Но поскольку тело цикла содержит две инструкции, мы должны поместить их внутри блока, используя символы} {. К сожалению, компилятор анализирует этот код и решает, что оператор x = - 1; существует вне (или после) петли. Таким образом, цикл в то время как только Пер- формирует выходной Printf заявление. Результатом является то, что, если x больше 0, чтобы начать с, Printf выполняет и условие проверяется снова. Поскольку Printf не изменяло x, он все еще больше 0 и Printf выполняется снова. Это повторяется снова и снова, и никогда не останавливается (до тех пор, пока пользователь не заставляет программу прервать). Причина этого бесконечного цикла является логической ошибкой, логика программиста не было правильным. Чтобы устранить эту проблему, то цикл должен выглядеть следующим образом.

```
в то время как (x> y) {
E ( "% d \ n", x); // Вывод x x = - 1;
}
```

Бесконечные циклы часто являются проблемой для программистов, даже для тех, которые имеют обширный опыт. Этот тип логической ошибки часто бывает трудно обнаружить, и программист только открытиях ERS ошибка при тестировании программы и находя, что это, кажется, "повесить". В этом случае цикл никогда не заканчивается, так что программа не продвигается к следующему шагу.
Напомним, из ранее, что Python не использует специальный блок обозначения, такие как начать ... конец и {...}. Вместо этого используется отступы. Так что в этом случае бесконечный цикл из ранее, вероятно, не произойдет, потому что отступы бы решить.

Подпрограммы и вызовы подпрограмм

Подпрограмма представляет собой набор кода отдельно от кода, который может вызвать его. Команда вызова подпрограммы вызывает подпрограмму. Подпрограмма может быть частью вашей программы, или это может быть частью другой программы или какой-либо независимой части кода, которая доступна через библиотеку подпрограмм. Большинство программистов писать программы в виде серии подпрограмм. Таким образом, проектирование, написание и отладка программы упрощается, поскольку программист должен

только сосредоточиться на одной подпрограммы в то время. Концепция известна как модульное программирование, где модули представляют собой относительно небольшие и независимые куски кода.
Ниже приводится краткий пример из двух подпрограмм, написанных на языке C. В C, подпрограммы называются функциями. Функция главной всегда является первым выполнить в программе. Здесь главный выполнит свой первый PRINTF (выход) заявление, а затем вызовите функцию Foo. Функция Foo будет выводить его собственное заявление и прекратить. Когда Foo завершается, главный возобновляется с точка сразу после вызова Foo, который является последним оператором Printf. Функция Основной затем прекращает свою работу и программа закончится.

```
недействительными Foo () {
E ( "мы в обув Да?!");
}

пустая функция( ) {
E ( ". Привет, мир Мы собираемся ввести Foo."); Foo ();
E ( "Теперь мы вернулись из Foo.");
}
```

Вывод этой программы выглядит следующим образом:

```
Привет мир. Мы собираемся ввести Foo. мы в Foo? Да!
Теперь мы вернулись из Foo.
```

Очевидно, что нет никаких причин, чтобы написать эту программу в двух функциях; мы могли бы вместо того, чтобы написать все три PRINTF заявления в главном. Тем не менее, большинство программ слишком сложны, чтобы писаться в одной функции.
В предыдущем примере, вызов функции Foo (). Круглые скобки служат средством связи между вызывающей функции (основной) и вызываемой функции (Foo). Эта коммуникация поставляется в виде переменных и их значений. Мы называем их параметры. Например, если функция будет вычислять квадратный корень из значения, мы должны сказать, что значение функции мы хотим, чтобы квадратный корень. Если функция вызывается SQRT, мы могли бы вызвать функцию, используя обозначения, такие как SQRT (5) для вычисления квадратного корня из 5, или SQRT (x) для вычисления квадратного корня из значения,

хранящегося в переменной x. Ниже приведен пример передачи параметра в функции в C. Функция вычисляет значение (обратного параметра) и выводит результат.

Аннулирование определяется как Reciprocal (флоат x) {если (x == 0)
E ("Там нет взаимных 0"); еще E ("Взаимное является% F", 1,0 / x);
}
Здесь функция передается значение с плавающей точкой. Функция будет выводить другое сообщение для каждого отдельного параметра. Например, что делает это выход, если x = 1,0? Что делать, если x = 10,0? Что делать, если x = 0?

Инструкции специального назначения

Они могут включать в себя вызовы операционной системы, операции со строками, компьютерная графика процедуры, случайное число операций и процедур обработки ошибок.
Помимо списка команд, выполняемых выше, в некоторых языках программирования также требуют явного переменную *ations*. In languages such as Pascal, C, and Java, различия требуют набора, то есть, объявляется конкретный тип значения, которое они затем остаются в ходе программы. В JavaScript, хотя переменные должны быть объявлены по имени. На рисунке 14.6, трех переменных, x, y и r, объявляются на трех языках: Pascal (вверху), C (в центре) и JavaScript (внизу). В C, INT означает целое реальное (число с десятичной точкой) Паскаля.
Некоторые языки позволяют определять собственные типы. В Ada, вы можете объявить тип, что может быть подмножеством другого типа. Например, мы могли бы определить тип, для подмножества целых чисел допустимого диапазона 0..100. Таким образом, переменная типа – это любое целое число, где переменная типа класса является диапазон от 0

```
Var x, y : integer;
Var z : real;


int x, y;
float z;


var x, y, z;
```

Рисунок 14.6
Три формы объявления переменных.

Что в имени?
Наиболее ранние языки программирования были названы с помощью сокращений, которые объясняли язык (например, FORTRAN, COBOL, ALGOL, LISP). Сейчас, языки программирования более креативны. Далее представлены некоторые из них.

- **Axum**
- **BeanShell**
- **Blue**
- **Boo**
- **Boomerang**
- **Caml**
- **Chef**
- **Cola**
- **EASY**
- **Fancy**
- **Go**
- **Groovy**
- **Joy**
- **Lua**
- **Nice**
- **Oxygene**
- **Pure**
- **Rust**
- **Scratch**
- **Shakespeare**
- **Squeak**
- **ZZ T-oop**

до 100. В Java, типы, называемые как классы.

Скриптовый Язык

Как системный администратор, важно, что вы понимаете язык программирования, потому что вы будете часто писать сценарии. Сценарий является интерпретируемой программой. Сценарии часто довольно короткие (по сравнению с приложениями или системного программного обеспечения). Но, несмотря на отсутствие необходимости писать код для крупномасштабных приложений, системный администратор должен еще понять команды языка программирования.

Виды Инструкции

Популярные языки сценариев на сегодняшний день включают оболочки языков сценариев (например, Баш script - язык ИНГ и DOS), а также более полные и сложные языки программирования, такие как Python, Ruby, Perl и PHP. Здесь мы сконцентрируемся только на Bash Shell и DOS языках. Мы специально рассмотрим особенности языка, описанные в типы инструкций, чтобы увидеть многочисленные примеры. Поскольку это не программирование текста, эти сечения нужны только для того, чтобы ввести сценарии оболочки.
Некоторые коды, которые вы можете поместить в сценарий оболочки так же, как команды, которые вы используете в командной строке. В Linux, например, сценарий может содержать компакт-диск, LS, гт, эхо, и так далее. И точно так же, команды, которые вы увидите в следующих двух подразделах также можно ввести в командной строке. Тем не менее, есть некоторые различия. Сценарий может получить параметры, что делает сценарий более мощным.

Bash оболочки языка

скрипты оболочки хранятся в текстовых файлах. Эти файлы должны быть как для чтения и выполнения. Лучше всего устанавливать такие файлы с 745 или 755 разрешения. Для выполнения сценария с именем Foo, вы должны ввести ./ Foo. Это означает, что ваш рабочий каталог хранит файл Foo. Все сценарии оболочки Bash должна начинаться со следующей строки, которая предупреждает интерпретатор Bash выполнить: #! / Bin / Баш.
Переменные в сценариях Bash обрабатываются так же, как из командной строки. Вы можете присвоить переменной значение, используя оператор присваивания вида VAR = VALUE. * Значение на правой стороне должна быть строка, буквальное число, или значение вычисляется арифметическое выражение целого числа на основе. Если строка содержит пробелы, заключите всю строку в "" знаков. Мы смотрим на целочисленных основах арифметических выражений ниже. Для того, чтобы получить доступ к значению в переменной, перед именем переменной с $. Ниже приведено несколько примеров.

```
X = 0
Y = "Привет там" NAME = Frank Z = $ X
MESSAGE = "Y $ NAME"
```

X содержит значение 0, Y строка "Привет там", и NAME строка "Франк", даже если не кавычки не были использованы в операторе присваивания. Z получает значение

* Обратите внимание, что имена переменных не должны быть написаны заглавными буквами. Я буду использовать заглавные буквы, чтобы легко отличить переменную из инструкции.

В настоящее время хранится в X, 0. ПОСЛАНИЕ получает строку значения, хранящегося в Y (Привет там), пробел, и значение, хранящееся в NAME (Frank), или "Привет там Frank".
Операция целое на основе сложно, потому что, если он имеет переменные значения, оно должно быть заключено в своеобразный синтаксис: $ (()). Например, чтобы установить Z быть X + 5, вы должны использовать Z = $ ((X + 5)), а также для увеличения X, вы должны использовать X = $ ((X + 1)). Юридические операторы для арифметических выражений являются +, -, *, / и% (% выполняет операцию по модулю, то есть, он обеспечивает остаток от деления). Операция деления (/) работает только с целыми числами и возвращает целое частное. Например, если X = 5 и Y = 2, то Z = $ ((X / Y)) приводит к Z быть установлено на 2 нет,
2.5. Операция Z = $ ((X% Y)) приводит к Z, установленным на 1, поскольку 5/2 2 с остатком 1.
Выходное утверждение эхо. Вы видели это раньше в учебнике. Эхо Оператор содержит список элементов для вывода. Этот список может содержать переменные, буквенные значения и команды Linux. Если вы поместите этот список ", выход осуществляется в буквальном смысле (то есть, значения переменных не возвращаются, вместо этого вы получите имена переменных). Например, если NAME = Франк и возраст = 53, то

Эхо $ NAME составляет $ AGE обеспечивает вывод Франк 53
но инструкция
Эхо '$ NAME составляет $ AGE'
обеспечивает вывод
$ NAME составляет $ AGE.

Вы также можете разместить свой выход в пределах "", который обеспечивает тот же результат, не имеющий кавычки. В большинстве случаев, "" может быть опущен в эхо заявлении. Тем не менее, рассмотрим следующие два эхо-заявления:

Эхо $ NAME составляет $ AGE эхо "$ NAME составляет $ AGE"

Первое эхо утверждение содержит три аргумента (параметры), в то время как второе утверждение эхо содержит только один. Кроме того, использование "" подавляет названия файла Bash 'разлет. Например, эхо * приведет к Bash первое расширение * быть все элементы в локальном каталоге, а затем эхо выведет их, так что эхо * такое же, как Ls или Ls *. Однако эхо "*" в буквальном смысле выход *, как расширение имени файла не происходит.
Вы можете использовать эхо объединить вывод буквального сообщения с выводом другой команды Linux. Например, если вы пожелали вывести "дата и время являются" вместе с текущей даты / времени, вы можете объединить эхо заявление и дату операцию. Тем не менее, чтобы сохранить эхо от вывода слова "дата" в буквальном смысле, вы должны указать, что Bash должен выполнить команду даты. Это достигается с помощью обратно кавычки. Например, оператор

эхо «дата»

буквально выход "дата". Тем не менее, заявление

эхо дата `date`

заставляет Linux выполнить команду даты, а затем вставить свой выход в эхо значение. Результат будет выглядеть следующим образом: "дата Чт 29 марта 2012 10:35:32 EDT".
Ввод осуществляется через отчет чтения. Это утверждение должно всегда быть далее следует имя переменной. Например, "чтение NAME" будет использоваться для получения ввода от пользователя, и поместить введенное значение в переменную NAME. Когда сценарий с входным значением выполняется оператором, чтение личных данных приведет к мигающему курсору на пустой строке. Поскольку пользователь может не знать, что происходит, или то, что программа ожидает, важно, чтобы предшествовать любому заявление с предлагающим заявлением. Контекстное это просто оператор вывода, что дает пользователю о том, что делать. Следующий пример запрашивает у пользователя их имя, введите значение, а затем обеспечить персонализированный выходное заявление. Обратите внимание на то, что имя переменной во входном

заявлении не предшествует $, в то время как $ используется в выходном сообщении.

эхо Введите имя чтения NAME
эхо Привет $ NAME, приятно встретиться с вами!

Пользователь может ввести что-либо в операторе чтения. Например, они могут ввести свое имя или другую строку целиком, или даже несколько. Ничто в коде выше, не препятствует этому. Если вход имеет место в нем, никакого специального механизма не требуется в коде, в отличие от оператора присваивания, такие как NAME = "Frank Zappa", который требует, чтобы строка быть помещены в кавычки. Если пользователь вводит Фрэнк Заппа для указанного выше кода, то выход будет, как и ожидалось, "Привет Фрэнк Заппа, приятно встретиться с вами!".
На данный момент, мы приложим все вместе, что уже видели. Следующий сценарий будет вычислять среднее значение из четырех значений. Обратите внимание на использование скобок в двух операторов присваивания. Напомним, что для выполнения арифметической операции, которая содержит переменные, можно использовать обозначение $ ((...)). Тем не менее, в данном случае, так как мы хотим подвести num1, num2, num3 и Num4 перед выполнением разделения или мод операцию, мы размещаем, что суммирование в
еще один слой круглые скобки. Без скобок, деление (Num4 / 4) выполняется перед дополнениями.

#! / Bin / баш
эхо Пожалуйста, введите ваш первый номер чтения num1
эхо Пожалуйста, введите ваш второй номер чтения num2
эхо Пожалуйста, введите ваш третий номер чтения num3
эхо Пожалуйста, введите ваш номер четвёртого чтения Num4
фактор = $ (((num1 + пит2 + num3 + Num4) / 4)) остаток = $ (((num1 + пит2 + num3 + Num4)% 4))
эхо-сигнала в среднем составляет $ фактор с остатком $ остатка / 4

Условный оператор в Bash имеет очень специфический синтаксис. Из-за этого, он часто бросает вызов для вводных скриптеров, чтобы получить их сценарии запуска правильно. Убедитесь, что вы будете придерживаться синтаксиса. Форма заявления является ", если [состояние]; то действие (s); фи ". Полупроницаемую колоны должны быть расположены там,

где они показаны здесь. Кроме того, должно быть промежутки между различными компонентами, если заявление, как показано на рисунке. И, наконец, слово "фи" должно закончить, если есть заявление. Действия являются отдельными высказываниями, разделенных точкой с запятой. Например, вы можете иметь операторы присваивания, входные операторы, операторы вывода, и даже если вложенными-заявления.
Условный оператор требует условие. Есть два распространенных форм условия:

• значение переменной сравнения
• filetest имя файла

В первом случае, имя переменной предшествует $. Сравнения являются одним из == или! = Для сравнения строк, или один из -eq, -ne, -gt, -lt, -ge, -lt для цифровых сравнений (равно, не равно, больше чем, меньше чем, больше или равно, меньше или равно). Значение может быть буквальным значением, другая переменная (чье имя снова предшествует $), или арифметическое выражение.
Второй случай является условием filetest с последующим именем файла. Filetest является одним из - d (пункт является каталогом), - e (файл существует), - f (файл существует и является обычным файлом), -h (пункт является символической ссылкой), -r (файл существует и является читаемый), - w (файл существует и доступен для записи), и - x (файл существует и является исполняемым). Filetest может предшествовать символ! чтобы указать, что условие должно быть сведено на нет. То есть, тест - r foo.txt спрашивает, если foo.txt читается, в то время как! -r foo. txt спрашивает, если foo.txt не читается. Некоторые примеры условия следуют. Обратите внимание на пробелы между каждой части условия. Без этих пробелов, ваш сценарий будет генерировать ошибку и не выполнить.

[$ NAME == "Frank"] [$ X -gt Y $]
[$ X -eq 0]
[$ STUDENT1! = $ STUDENT2] [-x $ FILENAME]
[! -r $ ФАЙЛ2]
Полный пример приведен ниже. В этом случае-то заявление, то переменная $ NAME сверяется со значением "Франк", и если они равны, то инструкция и выводит сообщение "Привет Франк" и добавляет 1 к переменной X. ФИ в конечных состояниях что это заканчивается if-

оператор.

если [$ NAME == "Frank"]; то эхо Hi Frank; $ X = $ ((X + 1)); Fi

В некоторых случаях, это состояние требует более одного сравнения, в сочетании с логическим или OR. Например, чтобы проверить, чтобы увидеть, если значение переменной лежит между двумя другими значениями, вы должны проверить переменную против обоих значений. Если мы хотим, чтобы увидеть, если значение хранится в переменной X и находится в диапазоне от 0 до 100, мы испытали бы $ X -ge 0 и $ X -le 100. Синтаксис становится еще более свойственно в этом случае. Такое условное называется компаунд условным и требует дополнительного набора [] вокруг всего состояния. Символы, используемые для И и ИЛИ те же, что используются в C и Java, && для AND и || для OR. Чтобы проверить X от 0 до 100, мы будем использовать следующее: [[$ X -ge 0 && $ X -le 100]].
Спецификатор еще можно добавить в отчет if-, и в этом случае слово "еще" появляется после того, как действие (ы) в то пункте, за которым следует действие (я) пункт еще. "Фи" появляется только после того, как в пункте другого. Ниже приведены три примера. Обратите внимание на то, что второй пример на самом деле не имеет никакого смысла, потому что нет никаких с плавающей точкой вычисления в Linux, поэтому операцию, которую $ ((1 / X)) не обеспечит истинную обратную.

```
если [$ X $ -gt Y]; то Y = 0; еще X = 0;
Fi
если [$ X -ne 0]; то Y = $ ((1 / X)); эхо Обратную $ X составляет $ Y; иначе
эхо не может вычислить обратную 0;
Fi
если [$ NAME == "Frank"]; затем эхо "Hi Frank"; X = $ ((X + 1)); еще эхо
Кто ?; X = 0;
Fi
```

Обратите внимание, что вы не должны поместить пункт или другое заявление фи на отдельных строках, они написаны таким образом, здесь в целях удобства чтения.

Существует также вложенная если-то-другое заявление, где "еще, если" пишется как Elif. Структура выглядит следующим образом:

если [состояние]; тогда
действие (s); Элиф [состояние], то
действие (s); Элиф [состояние], то
... еще
Fi
действие (s);
действие (s);

Если обратиться к рис 14.5, есть другое заявление на языке Паскаль. Эквивалентное утверждение в Bash показано на рисунке 14.7.
Есть два типа циклов, доступных на языке Bash сценариев. Для цикла является цикл итератор, который будет перебирать список. Список может состоять из одной из нескольких вещей. Во-первых, он может быть нумерованный список значений. Во-вторых, это может быть список, который был поставлен в скрипт в виде списка параметров (это обсуждается далее в этом разделе). В-третьих, это может быть список, который генерирует подстановку, то есть расширение имени файла.
Основная форма Bash для петли:
для переменной в списке; сделать заявление (ы); сделанный
Здесь переменная является индекс цикла. Индекс цикла принимает по каждому значению в списке, по одному за раз. Индекс цикла, то можно ссылаться в высказываниях в вашем теле цикла. Например, ибо я в 1 2 3 4 5; делать эхо $ I; сделанный будет выводить каждый из 1, 2, 3, 4, и 5 на отдельных строках, по одному на итерации цикла.
Цикл в основном используется для двух целей. Во-первых, перебирать элементы в директории, воспользовавшись универсализации. Во-вторых, перебирать параметры для того, чтобы обработать их. Опять же, мы будем держать взгляд на параметры, пока позже в этом разделе. Следующий цикл является примером использования глобирование. В этом случае

```
if [ $grade -ge 90 ];
        then letter=A;
elif [ $grade -ge 80 ];
        then letter=B;
elif [ $grade -ge 70 ];
        then letter=C;
elif [ $grade -ge 60 ];
        then letter=D;
        else letter=F;
fi
```

Рисунок 14.7
Вложенные если-то-иначе в Bash

цикл просто выполняет длинный список по каждому пункту в текущем каталоге. Это может быть легко осуществлено с использованием Ls -l *. *.

for file in *.*; do ls –l $file; done
Глобирование в цикле становится более полезным, когда мы размещаем условный оператор , в теле цикла, где оператор тестирует файл, используя одно из условий filetest. Далее, например, проверяет каждый текстовый файл в каталоге и выводит те, которые являются исполняемыми.
дfor file in *.txt; do
if [-x $file];
then echo $file;
fi
done
Углубление не является необходимым, но добавляет читаемости. Этот цикл перебирает все файлы .txt в текущем каталоге и отображает имена файлов тех файлов, которые исполняемые.
Другой тип цикла – это условный цикл. Этот цикл аналогичен петли C.
while [condition]; do statement(s); done
Напомним, что в то время цикл может быть бесконечным. Поэтому крайне важно, чтобы тело цикла содержало код, который будет изменять состояние от верного к ложному в какой-то момент. Один здравый цикл использования, в то время, чтобы обеспечить взаимодействие с пользователем, который управляет числом итераций цикла. Например, мы могли бы выполнить некоторый процесс в списке имен файлов. Мы могли бы попросить пользователя ввести имя файла. Для того, чтобы контролировать, продолжает ли цикл или нет, мы можем попросить пользователя ввести специальное ключевое слово, чтобы выйти, например, "бросить курить". В этом случае, "бросить курить" будет называться значение дозорного. Следующий сценарий неоднократно запрашивает у пользователя имена файлов, выводит те файлы, которые для чтения и записи, а также выход из цикла после того, как пользователь вводит "бросить курить". Если-то заявление подсчитывает количество раз условие истинно для суммарного выхода в конце кода.

#!/bin/bash
COUNT = 0
echo Введите имя файла, выйти, чтобы выйти читать FILENAME
в то время как [$ FILENAME! = Бросить курить]; делать

```
если [[-r $ FileName && -w $ FILENAME]];
затем -l $ левая шкала FILENAME; COUNT = $ ((COUNT + 1));
 сделанный

Fi
echo  Введите имя файла, выйти, чтобы выйти читать FILENAME

echo  Там были $ COUNT файлы, которые были для чтения и записи
```

Обратите внимание, в предыдущем примере, что две команды появились как перед циклом и в конце цикла. Почему вы думаете, что эхо и читать заявления должны были появляться в обоих местах? Подумайте, что произойдет, если мы опустили два заявления от перед циклом. Когда условие цикла достигается, $ FILENAME не возвращает никакого значения, поэтому цикл никогда не вошел. Таким образом, мы должны обеспечить начальное значение имени файла. Если мы опустили эхо и прочитать внутри цикла, то любое имя файла, который вы ввели изначально остается в переменной FILENAME. Таким образом, условие петля в то время всегда является истинным, и мы имеем бесконечный цикл!
Вы, как правило, можете использовать то время цикла, которое вы хотите. Выполнить задачу по ряду данных, вводимых пользователем, ожидая, пока пользователь, введет значение, которое указывает выходящее условие (например, слово бросить курить), или когда вы хотите сделать серию вычислений, которые должны остановить, когда вы достигли предела. Первый случай показан в предыдущем коде. На следующем примере, вычисляет полномочия 2 меньше, чем 1000, и выводит первый найденный быть больше или равен 1000.

```
#/bin/bash
VALUE=1
в то время как [$ VALUE -lt 1000]; делать
VALUE = $ ((VALUE * 2));
сделанный
echo   первой степени двух более чем 1000 составляет $ VALUE
```

Bash скрипты могут быть переданными параметрами. Они вводятся в командной строке после имени сценария. Сценарий выполняется с использованием обозначения ./script <Enter>. Параметры могут быть

добавлены до <войти>, как и в ./script 5 10 15 <Enter>. Эти параметры могут быть использованы в сценарии, как если бы они были переменными, инициализируются со значениями, предоставленные пользователем. Есть три различных способа параметров задания. Во-первых, $ # хранит число параметров при условии. Это может быть использовано в если заявление, чтобы определить, является ли пользователь предоставил надлежащее количество параметров. Во-вторых, каждый отдельный параметр доступен, используя $ 1, $ 2, $ 3, и так далее. Если вы ждете два параметра, вы бы получить доступ к ним, как $ 1 и $ 2. И, наконец, $ @ возвращает весь список параметров. Вы должны использовать $ @ в цикле в виде списка. Следующий скрипт выводит больший из двух параметров. Если пользователь не указывает точно два параметра, то получает сообщение об ошибке.

```
 #!/bin/bash
если [$ # -ne 2];
то  echo  этого сценария требует два параметра; elif   [$ 1 -gt $ 2];
то echo  Ваш первый параметр больше; еще echo   Ваш второй параметр больше;
Fi
```

Следующий сценарий ожидает, что список имен файлов. Если такой список не предусмотрено, то есть, если $ # равно 0, он выводит сообщение об ошибке. В противном случае она перебирает список и проверяет каждый файл для исполняемого статуса и обеспечивает длинный список всех тех файлов, которые являются исполняемыми. Обратите внимание на то, что этот сценарий имеет другое заявление, в котором условие еще имеет цикла и цикл имеет заявление. Из-за двух заявлений в сценарии, есть необходимость в двух утверждений Fi.

```
#/!bin/bash
если [$ # -eq 0]; тогда
Не  echo  Нет указанных параметров, не может продолжаться;
еще
за файл в $ @; делать
если [-x $ FILE]; затем -l $ левая шкала файла;

Fi
сделанный
Fi
```

Давайте закончим со сценарием, который будет ожидать список номеров, предоставленных в качестве параметров. Сценарий будет вычислять, и выводить среднее из списка. Предполагая, что скрипт называется СРЕДНЕМ, вы можете вызвать его из командной строки, набрав ./avg 381 56 10 18 266 531

```
#/!bin/bash
если [$ # -eq 0]; затем эхо параметров, не может вычислить среднее значение; еще
SUM  = 0;
за NUMBER  в $ @; сделать Sum = $ ((SUM + номер));
сделали AVERAGE  = $ ((SUM / $ #));
echo  в среднем значений в $ # составляет $ AVERAGE ;
Fi
```

Обратите внимание, в этом примере, что все введенные цифры были указаны в момент, когда пользователь выполнил сценарий. Что делать, если мы хотим, чтобы ввести значения с клавиатуры (с помощью чтения команда)? Как бы она отличается? Эта проблема остается упражнение в разделе обзора. Тем не менее, вот несколько подсказок. Вы должны использовать то время цикла. Допустим, все числа, введенные будет положительным. Условие цикла, в то время как будет [$ ЧИСЛО - gt 0], так что, как только пользовательский ввод 0 или отрицательное число, в то время как цикл будет прекращен. Вы должны добавить как эхо заявление (запросить у пользователя) и прочитайте заявление до и в цикле в то время, чтобы получить первый вход, и, чтобы получить каждый последующий ввод.
Один последний комментарий на Bash сценариев оболочки. Использование точки с запятой может быть очень запутанной. В общем, вам не нужно, если все ваши инструкции размещены на отдельных строках. Это требует перемещение таких слов, как "тогда", "делать" и "другое" на отдельные линии. Приведенный выше сценарий также можно переписать следующим образом:

```
#!/bin/bash
если [$ # -eq 0], а затем
еще
Не echo  Никакие параметры, не может вычислить среднее
```

```
SUM = 0
для заменяемого номера в $ @
делать
SUM = $ ((SUM + NUMBER))
сделанный
Средний = $ ((SUM / $ #))
echo  в среднем значений в $ # составляет $ AVERAGE  Fi
```

В качестве вводного Scripter оболочки, хотя, это всегда безопасно, чтобы добавить точку с запятой.

MS-DOS

В этом разделе мы рассмотрим команды и синтаксис, специфичные для MS-DOS. Вы должны прочитать оболочки Bash языка, прежде чем читать этот раздел, как некоторые из понятий, таких как петли, не будет повторяться здесь. Вместо этого предполагается, что вы уже знаете, концепциями, так что мы можем ограничить наше обсуждение на команды, синтаксис и примеры.

Как и на языке Bash, DOS позволяет переменные, которые могут хранить строки и целые числа. Есть три типа переменных: параметры, предоставляемые из линии команд, когда выполняется скрипт, переменные окружения (определяется операционной системой или пользователем), а индекс для цикла переменных. Параметры обозначаются с помощью% N, где n означает число параметра. То есть, первый параметр% 1, второй 2%, до до% 9 для девятого параметра. Несмотря на то, что вы не ограничены девяти параметров, используя больше параметров сложно (это не так просто, как ссылки на% 10,% 11, и так далее). Переменные окружения появляются с нотациями% имя% (за исключением числовых значений, смотрите примеры, которые следуют). Для петли переменные появляются с именем нотации%. Чтобы назначить переменная значение, оператор присваивания использует набор слов, как и в "набор x = привет". Для того, чтобы присваивать вспомогательной переменной значение, хранящееся в другой переменной, используйте обозначение "набор X =% Y%". Эти знаки процента вокруг имени переменной интерпретируют переменную стоимости; в противном случае, интерпретатор буквально использует имя, например, "набор x = y" будет установлено x, чтобы сохранить значение "y", а не значение, сохраненное в переменной y. Тем не менее, это отличается, если мы имеем дело с арифметическими значениями, как описано в следующем абзаце.

По умолчанию значения в переменных обрабатываются как строки, даже если значения содержат цифры.
Рассмотрим следующие две команды
a = 5
Множество Ь =% в% + 1

похоже, чтобы сохранить значение 6 в переменной Ь. Но вместо того, инструкции хранить строку 5 + 1 в б. Это происходит потому, что элементы на правой стороне от знака равенства рассматриваются как строки по умолчанию, и поэтому обозначение% А% + 1 считается операция строки, путем объединения значения, хранящегося в строкой "+1". Чтобы изменить это поведение по умолчанию, вы должны указать / а в множестве заявление, чтобы рассматривать выражение в арифметической операции, а не объединения строк. Таким образом, второе утверждение следует "установить / АВ =% в% + 1", который бы правильно установить б до 6. Для арифметических утверждений, как это, вы можете опустить знаки процента из разных переменных, так что "установить / АВ = а +1 "будет сделать то же самое.
Input использует команду набора, а также, с двумя небольшими вариациями. Во-первых, параметр / Р используется для указания того, что множество оператор будет предлагать пользователю. Во-вторых, правая сторона знака равенства больше не будет иметь значение или выражение, но вместо этого будет побуждать сообщение, которое будет выводиться перед входом. Если опустить правую команду, пользователь видит только мигающий курсор. В следующем примере показано, как использовать ввод.

```
 set /p first=Enter your first name
set /p middle=Enter your middle initial
set /p last=Enter your last name
set full=%first% %middle% %last%
```

Обратите внимание на то, что пробелы вставляются между каждой переменной в последнем множестве заявление так, что объединение трех строк имеет пространства между ними, как и в Frank Zappa V.
Выходные заявления, содержащиеся в DOS использовать эхо-команду, подобную Linux. Выходные заявления будет сочетать в себе буквенные значения и переменные. Для того, чтобы различать, хотите ли вы буквенное значение, скажем, Hello, или значение, которое хранится

в переменной, вы должны поместить имя переменной внутри знаки процента. Если предположить, что имя переменной хранит Фрэнк Заппа, заявление "Эхо привет имя" буквально выход "привет имя", тогда как "эхо привет% имя%" будет выводить "привет Frank Zappa".
Сценарий DOS настроен для вывода каждый исполняемый оператор, как он выполняет. Для того, чтобы "отключить эту функцию", вам нужно добавить @ECHO прочь. Для того, чтобы отправить вывод в файл, а не на экране, добавьте "> имя файла" в эхо-заявление для записи в файл и ">>" имя файла для добавления к файлу. Например, если предположить, что имя снова магазины Фрэнка Заппы. Три следующих выходных заявления показывают, как вывести приветственное сообщение на экран, в файл greetings.txt и добавить в файл greetings.txt.
echo привет% имя%
echo привет имя%%> greetings.txt эхо привет% Имя% >> greetings.txt

Условный оператор допускает три типа условий. Вы можете проверить уровень ошибки как показателей, представленных в последней выполненной программе, сравните две переменные / значения, и проверьте, если файл существует. Для каждого из этих типов условий, вы можете предшествовать условие со словом нет. Сравнение являются фас (=), NEQ (не равно), LSS (<), Leq (<=), ГТП (>) и GEQ (> =) для числовых сравнений и = = для строк. Условный оператор имеет необязательный другое положение. Весь если-заявление (в том числе дополнительного пункта другое) должен появиться на одной строке сценария, таким образом, он ограничивает количество действий, которые то пункт или другое пункт может выполнить. Если вы используете дополнительный другой, поместите оба затем пункт и пункт еще внутри (). Слово "затем" опущено, как в C. Вот некоторые примеры if-заявлений.

Если код ошибки 0 эхо предыдущая программа выполняется корректно
Если% x% Гтп 5 (эхо X больше, чем 5) остальное (эхо X не больше 5)
Если% x% LSS 0 множество / a x = 0-% x%
Если нет% имя% = = Франк (эхо, я не знаю, что вы)
При наличии C: \ Users \ foxr \ foo.txt дель C: \ Users \ foxr \ foo.txt

DOS содержит цикл, но не во время работы цикла. Для цикла является цикл итератора, как цикл Linux. Это только перебирает список значений. Формат выглядит следующим образом:

для опции % Var в (список) делать команду

В следующем примере будет суммировать значения в списке. Он не использует параметры в течение цикла.

for %i in (1 2 3) do set /a sum=sum+%i

Помимо предоставления списка перебрать, можно указать такие команды, как директории, так что для цикла перебирает результата от команды реж. Для этого необходимо включить опцию / п в для цикла. Например, следующий вывод будет в искомые позиции при выполнении команды DIR, по одному за раз.

for /f %file in (‘dir’) do echo %file

К сожалению, комбинируя для цикла с помощью команды ('реж') могут не работать, как ожидалось, поскольку команда реж обеспечивает ценность нескольких столбцов 'элементов, и эхо только выражений поднимается на первом элементе (столбец) каждого линия (которая является датой последней модификации). Команда реж / б дает только сами имена элементов. Чтобы изменить выше инструкции, необходимо указать для F% файла / в ('реж / б') делать эхо % файла
Как и с Linux, DOS команды могут быть помещены в сценариях оболочки DOS. Это включает в себя такие команды, как del (удалить), move, copy, ren (переименовании), dir, type (похожий на Linux’ cat), print (отправить на принтер), RmDir, MKDIR, CD, и CLS (прозрачный экран), Так что вы можете писать сценарии, например, для проверки файлов в файловой системе, и удалить, переименовать или переместить их, если они существуют или имеют определенные атрибуты.

Дальнейшее чтение

• Блюм, Р. и Бреснахан, С. Linux командной строки и Shell Scripting Библии. Нью-Джерси: Wiley и Sons, 2011.
• Dietel, П. и Dietel, Н. С ++: Как программировать. Верхний Saddle River, Нью-Джерси: Prentice Hall, 2011.
• Стекло, Р. В начале: Личные Воспоминания Software Пионеров. Нью-Джерси: Wiley и Sons, 1997.
• Kelley, А. и Pohl, И. Книга о С: Программирование на языке С. Reading,

MA: Addison Wesley, 1998.
• Керниган Б. и Ритчи Д. Язык программирования C. Englewood Cliffs, NJ: Prentice Hall, 1988.
• Лян, Ю. Введение в Java-программирование. Boston, MA: Pearson, 2013.
• Митчелл, J. Концепции в языках программирования. Великобритания: Кембридж, Cambridge University Press, 2003.
• Ньюхем, C. Изучение оболочки Bash: Unix Shell по программированию. Массачусетс: O'Reilly, 2005.
• Редди, Р. и Циглера, C. FORTRAN 77 с 90: Приложения для ученых и инженеров, Сент-Пол, штат Миннесота: West Publishing, 1994.
• Robbins, A. и Биб, H. Classic Shell Scripting. Массачусетс: O'Reilly, 2005.
• Себеста, Р. Концепции языков программирования. Boston, MA: Pearson, 2012.
• Такер, А. и Нунан, Р. Языки программирования: принципы и парадигм. Нью-Йорк: McGraw Hill, 2002.
• VENIT, C. и Субраманьян, П. Прожектор на Структурное программирование с Turbo Pascal. Иган, MN: West Publishing, 1992.
• Уэббер, А. Современные Языки программирования: Практическое введение. Wilsonville, OR: Франклин, Бидл и Associates, 2003.
• Weitzer, M. и Ерш, Л. Понимание и использование MS-DOS / PC DOS. Сент-Пол, Миннесота: West Publishing, 1986.

ГЛАВА 15. ИНФОРМАЦИЯ

Учебник в первую очередь подчеркнул технологию. В этой главе основное внимание обращается к информационной стороне. В частности, в главе рассматриваются типы информации, организации хранения и использования вместе с программным обеспечением, которое поддерживает информацию о переработка. Затем в этой главе рассматривается информационная безопасность и уверенность: способность организации обеспечивать, чтобы их информация защищена. С этой целью глава описывает многочисленные формы атак, которые организация должна защитить, наряду с технологическими методами информационной безопасности. В главе рассматриваются аппаратные решения, такие как хранение RAID и программных решений, включая шифрование. Глава заканчивается взгляд на законодательство, созданный, чтобы помочь защитить информационные технологии и неприкосновенность частной жизни граждан.

Цели обучения данной главы являются

• Обсудите различные формы, что информация приходит с введением в иерархии Информация-Знание-Опыт .

• Опишите использование баз данных, систем управления базами данных и хранилищ данных.

• Ввести области информационной безопасности и обеспечения и процессов, используемых при анализе риска.

• Обсудите угрозы информации и решений в области охраны информации.

• Описать технологии шифрования.

• Ввести законодательство на территории США, облегчающая использование информации и защищает конфиденциальность индивидами.

До этого момента учебника, мы сосредоточились в основном на технологической стороне информационной технологии (ИТ). Мы рассматривали компьютерное оборудование, программное обеспечение, операционные системы, программирование, и некоторые из основных теорий позади их использования и функций. В этой главе мы посмотрим на другую сторону ИТ, информационную .

Что такое информация?

Информация часто определяется как обработанные данные. То есть, необработанные данные являются входной и информация является выходом какого-либо процесса. На самом деле, в более общем плане, мы должны думать о информации как любая форма интерпретированных данных. Эти данные могут быть организованы, манипулируют, фильтруют, сортируют, и используется для вычисления. Затем эта информация представлена человеку в поддержку принятия решений. Мы могли бы просматривать данные и информацию в виде спектра, который колеблется от исходных данных, что соответствуют накоплениям, но неорганизованные выводы, к интеллектуальной собственности, которая включает в себя такие человеческие производства артефактов, как формулы и рецепты, книги и другие формы литературы, планов и конструкции, а также компьютерные программы и стратегии.
С точки зрения ИТ, однако, хотя информация часто является конечным результатом ИТ-процессов, мы должны рассмотреть более конкретно следующие вопросы:

• Как информация должна храниться
• Каким образом (или если) эта информация должна быть передана
• Какие процессы, которые мы хотим использовать на данных, чтобы превратить его в информации
• Как эта информация должна быть просмотрена / визуализирована
• Какие требования к обеспечению точности данных и информации являются
• Какие требования к обеспечению доступности и безопасности данных и информации сама информация не является желаемым конечным результатом обработки информации. Исследователи в области информатики разработали "DIKW иерархии". Эта иерархия определяет переход данных к информации к знаниям к мудрости. Каждый уровень иерархии определяется в терминах предыдущего уровня.
Данные непосредственно получены человека (или компьютера). Данные, как правило, думают как сигналы. Для человека, это могут быть сигналы, полученные с помощью органов чувств (зрение, звук, запах, вкус, осязание). Для компьютера, данными могут быть значения, введенные пользователем при запуске приложения, хотя в современной вычислительной технике , данные также могут быть получены с помощью камеры, микрофона, считывателя штрих-кода, датчика или планшета,

чтобы назвать только некоторые из них. Входные данные, как правило, не могут быть использованы до тех пор, пока не были преобразованы в соответствующую форму. Для компьютера, который будет двоичным представлением.

В иерархии, информация следующий уровень. Информация часто определяется в иерархии, как будто она была выведена из данных. Термин вывод (или вывод) означает, что один или несколько процессов, были применены к данным, чтобы преобразовать его в более удобной форме. Подразумевается в любом процессе, применяемом в том, что полученная информация является более структурированной и более актуальной, чем данные, сам по себе. Идея заключается в том, что данные, как правило, слишком низкий уровень для людей, чтобы использовать напрямую. Список номеров, например, сам по себе не может быть полезным при принятии решений, в то время как тот же список номеров обрабатываются с помощью статистического анализа, может обеспечить для нас среднее, медиана, стандартное отклонение, и дисперсии, которые могли бы тогда сказать нам что-то более значительное.

Знание является более расплывчатое понятие, чем любой информации или данных. Знание иногда определяется как информация, которая была введена в использовании. В других случаях, знание представляет собой синтез нескольких различных источников информации. Один из способов думать о том, что знания это информация, размещенная в контексте, возможно, является результатом опыта, полученного с использованием информации. Кроме того, мы могли бы думать о знании, как совершенствуются информации таким образом, что пользователь знаний способен вызывать только соответствующие части информации, когда это необходимо. В качестве примера, можно научиться решать алгебраические задачи путем изучения алгебраических законов. Законы представляют информацию, в то время как практика выбора надлежащего применения закона в данной ситуации является знание.

И, наконец, мудрость обеспечивает социальную установку к знаниям. Некоторые относятся к мудрости, как понимание "почему" - это "почему все случилось так, как они делали" или "почему люди делают то, что они делают". Мудрость может прийти только при наличии как знания и опыт. Знание, с другой стороны, можно извлечь из опыта других людей, которые имеют его. Мудрость является конечной целью для человека в том, что он улучшает способность человека к функции в современном мире и принимать обоснованные решения, которые принимают во внимание убеждения и ценности.

В некоторых случаях термин понимание добавляется к иерархии. Как правило, понимания, если она установлена, будет помещен между знанием и мудростью. Другие полагают, что понимание является частью мудрости (или требуется для уровней как знание и мудрость).
В следующем разделе мы сосредоточимся на данных и один из наиболее распространенных форм хранения, базы данных. Мы также рассмотрим хранилищ данных и приложений, используемых на домах Data фурнитура. Затем мы сосредоточимся на двух основных проблемных областях в области ИТ: Обеспечение информации и информационной безопасности. Среди наших решений различных проблем будет управление рисками, планирование бедствий, компьютерной безопасности, и шифрование. Мы закончим эту главу с рассмотрения соответствующего законодательства, которое имеет дело с использованием информационных технологий.

Базы данных

Информация имеет множество различных форм. Это может быть информация, записанная в текстах. Это может быть устное общение между людьми или группами. Она также может быть мысли, которые еще должны быть переданы другим лицам. Тем не менее, в этом, информация хранится в компьютере, и поэтому должны быть представлены таким образом, что компьютер может хранить и обрабатывать. Помимо требующей бинарной формы хранения, мы также хотим, чтобы организовать информацию в удобной форме. То есть, мы хотим, чтобы информация, хранящаяся смоделировать некоторый реальный эквивалент. Как правило, информация организована в виде записей и хранятся в базах данных.
База данных представляет собой организованную совокупность данных. Мы создаем, манипулировать, и получить доступ к данным в базе данных с помощью той или иной форме системы управления базами данных (СУБД). СУБД включают Microsoft Access, SQL и его вариации (MySQL, Microsoft SQL, PostgreSQL и т.д.), и Oracle, чтобы назвать несколько.
Типичный принцип организации базы данных является то, что отношения. Отношение описывает отношения между отдельными записями, где каждая запись делится на значения атрибутов. Например, отношение может состоять из записей студентов. Для каждого студента, есть первое имя, фамилия, студент ID, крупный, незначительный (по желанию), и средний балл (средний балл).
Само отношение часто представляется в виде таблицы (таблица

15.1). Строки таблицы представляют собой записи: Физические лица, представляющие интерес, являются ли они люди, предметы или понятия (например, запись может описать клиента, автомобиль или курс колледжа). Формально записи (строки) называются кортежами. Все записи соотношения одни и те же атрибуты. Они называются полями, и они составляют столбцы отношения. Как правило, поле будет хранить тип данных, таких как число, строка, дата, да / нет (правда ложной или /) значения, а также в пределах типа могут быть подвиды, например, число может быть длинное целое, короткое целое, вещественное число, или сумма в долларах.

В таблице 15.1, мы видим, студенческие академические отношение. В связи с этим, студенческие записи содержат шесть полей: удостоверения, фамилия, имя, мажор, минор, и ГПД. Каждая из записей является строка (текст, который может объединить буквы, знаки пунктуации и цифры) для ГПД, который является действительным числом, кроме. Student ID может быть несколько, но мы, как правило, использовать цифры для данных, которые могут быть использованы в некоторых арифметических операциях . Так как мы не будем добавлять или многозарядных идентификаторы ING студентов, мы будем хранить их в виде строк.

Поля отношения, возможно, ограничений, наложенных на значения, которые могут быть сохранены. GPA, например, должен находиться в пределах диапазона 0.000-4.000. Мы могли бы ограничить значения под мажор и минор, чтобы соответствовать списку юридических специальностей и несовершеннолетних за университет. Обратите внимание, для одной записи в отношении в таблице 15.1, что не существует никакого значения для Минор. Мы могли бы потребовать, чтобы все записи имеют майор, но не Minor.

Учитывая отношение, мы могли бы пожелать, чтобы запросить базу данных для некоторой информации, такой как "показать мне все студенты, которые CIT майоров", "показать мне все студенты, которые CIT майоры с GPA> = 3.0", "показать мне все студенты, являются CIT майоров или CSC главным с ГПД
> = 3,0 ". Эти типы запросов известны как ограничения. Результатом такого запроса является список тех записей в отношении, которые соответствуют заданным критериям. То есть, мы ограничиваем отношение к подгруппе на основе заданных критериев. Мы можем также обратиться к этому типу операции, как фильтрации или поиска.

Запросы возвращают данные из базы данных, хотя они не должны быть наложены ограничения. То есть, они не должны ограничивать записи,

возвращаемые. Проекция другая форма запроса, который возвращает все записи из соотношения, но только выбирать атрибуты или поля. Например, мы могли бы спроектировать соотношение из таблицы 15.1, чтобы обеспечить только первые имена и фамилии.
Еще одна операция по отношению является сортировать записи, основанные на некотором поле (ях). Например, записи в таблице 15.1 сортируются в порядке возрастания по фамилии, затем по имени (таким образом, Ян Андервуд предшествует Рут Андервуд).

Таблица 15.1 Пример базы данных связей

Студент ID	Имя	Фамилия	Кафедра	Незнасительность	GPA
11151631	George	Duke	MUS	PHY	3.676
10857134	Mike	Keneally	MUS	CHE	2.131
19756311	Ian	Underwood	MUS	HIS	3.801
18566131	Ruth	Underwood	MUS	MAT	3.516
18371513	Frank	Zappa	MUS		2.571

Мы могли бы также выполнять операции вставки (добавления записи), обновления операций (изменить один или несколько атрибутов одного или нескольких записей), а также удалять операции (удалить одну или несколько записей). Используя некоторые критерии фильтрации с обновлением или удалить, мы можем изменить или удалить все записи, которые соответствуют определенным критериям. Например, будучи щедрым учителем, я мог бы решить, что все CIT крупные компании должны иметь более высокий средний балл, поэтому операция обновления может указать, что ГПА = ГПД * 1.1, где Major = "CIT". То есть, для каждой записи, основная является "CIT", обновить поле GPA быть ГПД * 1.1.
База данных будет состоять из группы отношений, многие из которых имеют перекрывающиеся части. Таким образом, информация может быть взята из нескольких отношений одновременно. Например, мы могли бы иметь другой студент отношение, которое состоит из студента контактной информации. Таким образом, соотношение представлено в таблице 15.1 можно было бы назвать студента схоластической информацию, в то время как студент контактная информация может выглядеть так, как показано в таблице 15.2.
Учитывая два соотношения, теперь мы можем использовать другой запрос к базе данных, который называется соединением. Объединение

отзывает информацию из нескольких отношений. Например, мы можем объединить два соотношения из таблиц 15.1 и 15.2, чтобы получить все студенческие записи, предоставляя их удостоверение личности студента, и фамилии, майоров, несовершеннолетние, ГПД адреса и номера телефонов.

Мы можем объединить присоединения с ограничениями и / или выступов. Например, объединение и проекция может предоставить номера телефонов и имена и фамилии всех студентов. Объединение и ограничение может предоставить все записи студентов, которые CIT майоров, которые живут в Огайо. Объединение всех трех операций могли бы, например, выход первого и фамилии всех CIT специальностей, которые живут в OH.

Есть дополнительные операции, доступные в базе данных. Они включают в себя набор операций объединения, пересечения и разности. Союз объединяет все записи из отношений, указанных. Пересечения извлекает только те записи, которые существуют в обоих (или всех) отношений указанных. Разница возвращает те записи, которые не являются в обоих (или всех) отношений указанных.

Обратите внимание на то, что данные в различных соотношениях могут иметь значения повторяющихся атрибутов. Например, есть три студента из Огайо два студента с фамилией Андервуд, и пять студентов, специальности Информационные технологии . Для базы данных, чтобы работать, по крайней мере, один из атрибутов (поля) должен содержать уникальные значения. Это поле известно как уникальный идентификатор или первичный ключ. В случае обоих отношений в таблицах 15.1 и 15.2, уникальный идентификатор представляет собой студенческий билет. СУБД будет использовать уникальный идентификатор для сопоставления записей в разных соотношениях при использовании операции соединения. Например, если наш запрос запрашивает номера телефонов студентов с ГПД $\geq$ 3.0, СУБД должны сначала искать схоластические

Таблица 15.2 Другая база данных отношений

Student ID	Address	City	State	Zip	Phone
10857134	8511 N. Pine St	Erlanger	KY	41011	(859) 555-1234
11151631	315 Sycamore Dr	Cincinnati	OH	45215	(513) 555-2341
18371513	32 East 21st Apt C	Columbus	OH	43212	(614) 555-5511
18566131	191 Canyon Lane	Los Angeles	CA	91315	(413) 555-1111
19756311	32 East 21st Apt C	Columbus	OH	43212	(614) 555-5511

информацию для записей, чьи ГПД соответствует критериям, а затем вывести телефонные номера из контактной информации. Для выявления студентов между отношениями, используется уникальный идентификатор. Такжеобратите внимание, что записи в таблице 15.2 упорядочены по студенческому билету (в отличие от фамилии, как записи упорядочены в отношении из таблицы 15.1).

Реляционная база данных только один формат для организации баз данных, хотя это, безусловно, является наиболее распространенной формой. С 1980-х годов, исследования исследовали другие формы баз данных. Некоторые из наиболее интересных форматов перечислены здесь.

• Активная база данных реагирует на события, которые возникают внутри или внешне по отношению к базе данных, в отличие от статической реляционной базы данных, который реагирует только на запросы и другие операции с базами данных.

• Облако базы данных, как описано в главе 12, облако представляет собой сетевое хранилище и обработки объекта, поэтому база данных облако просто база данных, которая существует (хранится) в облаке и, таким образом, доступно в удаленном режиме.

• Распределенная база-база данных, которая не хранится исключительно в одном месте. Эта форма базы данных будет перекрывать облако и сетевые базы данных, и вполне вероятно, параллельную базу данных.

• базы данных документ базы данных, записи которых документы и который выполняет поиск информации, основанную на возможно менее структурированных запросов (например, ключевого слова квебекский Райса, как введенные в поисковой системе). базы данных документов часто встречаются в библиотеках, и поисковые системы, такие как Google может рассматриваться в качестве базы данных документов.

• Встроенная база данных - база данных и СУБД, являются частью большого приложения, которое использует базу данных. Одним из примеров является медицинской диагностической экспертной системы, которая содержит несколько компонентов: естественный язык интерфейса, база знания, рассуждающий, и базу данных записей пациентов.

• Иерархическая база-база данных, в которой данные моделируются родитель-потомок. Записи разбиты так, что одна часть записи находится в одном месте в базе данных, а другая часть находится в другой отрасли. Например, база данных сотрудник может, на самом высоком уровне, список всех сотрудников. Ребенок отношение может затем содержать для

данной информации работника о положении этого работника (например, уровень заработной платы, обязанности) и другого дочернего отношения могут включать информацию о проектах, которые работника.
• базы данных гипермедиа база данных содержит записи, связанные между собой гипер ссылками СМИ. World Wide Web можно рассматривать как базу данных гипермедии, где "records " документы. Это не традиционная база данных, в том, что записи не организованы в какой-либо конкретной моде, а отношения не являются организованными для сбора таких записей.
• многомерные базы данных отношение базы данных обычно считается в виде таблицы: двумерного представления. Многомерные базы данных хранит отношения, которые состоят из более чем двух измерений. Такое отношение могло бы организовывать данные таким образом, что третье измерение представляет собой изменение времени. Например, мы могли бы иметь группу.
Отношения, как у таблицы 15.1, где каждое отношение представляет студентов, обучающихся в конкретном семестре. Хотя мы могли бы объединить все эти отношения вместе и добавить поле семестр, трехмерная соотношение дает нам более разумным способом, чтобы просмотреть данные, поскольку она содержит более разумную организацию.
• Сетевая база данных, это не то же самое, как распределенной базы данных; вместо этого, термин сеть передает набор данных, которые связаны друг с другом, как ориентированный граф (используя математическое понятие графа). База данных сети несколько Сходные в иерархической базе данных.
• объектно-ориентированных баз данных, а не определять объекты, как записи, помещенные в варьировавших отношения, данные объекта собираются в единое целое. Кроме того, следуя из объектно-ориентированного программирования, объект инкапсулируется с операциями, определенными для доступа и управления данные, хранящиеся в объекте. По сути, это сочетание баз данных и объектно-ориентированного программирования.
• Параллельно база-это база данных работает на с несколькими процессорами параллельно. Для того, чтобы гарантировать, что данные не повреждены (например, изменении на один процессор в то время как другой процессор использует одни и те же данные), синхронизация должна быть реализована на данных.
• Пространственная или временная база данных, база данных, которая

хранит пространственную информацию, временную информацию, или обоих. Рассмотрим, например, медицинские документы, которые содержат записи о пациентах как последовательности событий. Специализированные запросы могут быть использованы для просмотра последовательности. Возможно, события 1 происходят до события 2? Или же они перекрывают либо соответствовать одному и тому же периоду? Пространственная база данных включает в себя возможность запроса о двух- и трехмерных взаимодействий.

База данных представляет собой набор записей, организованных в некотором роде. Реляционная база данных использует отношения. Как указано выше, другие организации включают в себя объектно-ориентированный, иерархическая, гипермедиа, сеть, пространственное и временное. База данных может состоять из нескольких записей или миллионы. Чем больше набор данных, тем более критической организации четко понимать и представлял. Как базы данных растут в их размерах и сложности, мы склонны рассматривать базу данных не просто набор данных, а набор баз данных. Сбор старых организованных баз данных называется хранилищем данных. Типичный хранилища данных использует процесс ETL.

• Извлечение данных данные поступают из различных источников, эти источники должны быть идентифицированы, поняты и постучал.
• Преобразование данных при условии, что данные из разных источников, вполне вероятно, что данные не организованы с использованием той же набор атрибутов. Таким образом, данные должны быть преобразованы, чтобы соответствовать модели (ям) хранилища данных. Это будет включать в себя данные отношений или объектов (в зависимости от того или формат является предпочтительным), признающие уникальный идентификатор (ы), и выбрав соответствующие атрибуты (поля).

• данных в oadL хранилище, то есть данные, преобразованные данные должны быть сохранены в базе данных с использованием соответствующего формата представления.

Для бизнеса, данные для хранилища данных может исходить от любого числа источников.
К ним относятся:

• Системы планирования ресурсов предприятия
• системы управления цепочками поставок
• Маркетинг и связи с общественностью отчеты
• рекорды продаж
• Приобретение и инвентарного учета
• Прямой ввод от клиентов (например, вход из веб-формы или обследования)
• записи человеческих отношений
• Планирование бюджета
• Текущие отчеты по проектам
• Общие данные бухгалтерского учета

После того, как собраны и преобразованы, данные сохраняются в хранилище данных. Теперь, данные могут быть использованы. Есть любое количество методов, которые могут быть применены, а результаты обработаны и могут быть сохранены обратно на склад для дальнейшего продвижения знаний организации, как описано ниже. Эти методы иногда называют - витрины с данными средствами, с помощью которых пользователи получают данные или информацию из склада.

Хранилища данных операций в диапазоне от простых операций управления базами данных до более сложных алгоритмов анализа и статистики. С точки зрения традиционных СУБД, пользователь может изучить вашу покупку, инвентарь, продаж и бюджетные данные для генерирования прогнозируемых потребностей в рабочей силе. Или, продажи и отчеты по связям с общественностью может помочь определить, какие продукты следует подчеркнуть, за счет будущих маркетинговых стратегий.

В OLAP (Online Analytical Processing), данные обрабатываются через набор программного обеспечения для анализа инструментов. Вы можете представить себе, как OLAP, сидя на верхней части СУБД, так что пользователь может извлекать данные из базы данных и анализировать данные без необходимости разделения операций СУБД от более сложных операций анализа. Операции OLAP включают нарезку, перетасовки, бурение вниз (или вверх), сворачивание и поворот данных.

Получение продольных данных создает подмножество данных путем уменьшения данных из нескольких измерений в одном измерении. Например, если мы думаем о данных в нашей базе данных как в трех измерениях, Получение продольных данных создаст одномерное представление данных. Получение поперечных данных такая же, как

Получение продольных данных за исключением того, что результат может быть в нескольких измерениях, но до сих пор получает подмножество данных. Например, может ограничить трехмерный сбор данных в меньший трехмерный сбор данных, путем отбрасывания некоторых записей и полей.

Категоризация и снижение уровня детализации данных просто смещает представление данных. Снижение уровня детализации данных обеспечивает больше деталей, сверление предоставляет сводные данные. Сворачивание похоже на категоризацию в том, что оно обобщает данные, но при этом, оно разрушает данные из нескольких элементов (возможно более одного измерения) в одно значение. В качестве примера, все HR записи могут быть свернуты в один нулевой точки, такие как количество нынешних сотрудников, или одного вектора, который представляет число сотрудников в каждой позиции (например, управление, техническая, поддержка).

Наконец, ось вращения данных для просмотра данных с другой точки зрения. Рассмотрим базу данных, содержащую информацию о продажах продукции по годам и по стране. Если типичный вид данных является продуктом, мы могли бы вместо этого хотите, чтобы повернуть все данные так, что наш взгляд первый показывает нам каждый год. Стержень затем реорганизует данные с новой точки зрения. Мы также могли бы

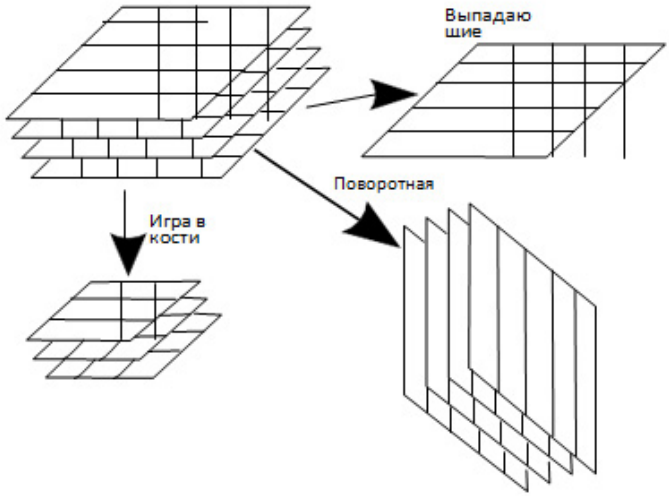


Рис. 15.1
Изображение техники анализа OLAP (Аналитический процесс в режиме онлайн)

повернуть эти данные по стране продажи.
Рисунок 15.1 иллюстрирует, отвлеченно, идеи, лежащие в категоризации , перетасовки и поворотными. Здесь мы имеем набор данных. Возможно, каждый слой (или плоскости) из исходных данных представляет собой базу данных отношение со стороны другой год. Например, верхний слой может быть записи о клиентах, начиная с 2012 года, следующий слой может быть записи о клиентах, начиная с 2011 года, а нижние два слоя с 2010 и 2009, соответственно. Срез может быть изучение одного полного слоя, скажем, что в 2010 году в кости может быть подмножеством в нескольких измерениях, например, ограничивая данные в годы 2012, 2011 и 2010, записи тех клиентов из Огайо, и поля фамилии, сумма, потраченная, а общее число посещений. Ось может реорганизовать данные так, что, а не первое нарушение год данные по годам, данные сначала разбиваются на состояние государства, а затем с каждым годом.
Анализ OLAP существует с 1960-х годов. Сегодня, однако, OLAP становится более важным, чем когда-либо, потому что хранилище данных является слишком большим, чтобы получить полезные ответы от простых операций с базами данных.
Еще один набор инструментов, которые могут быть применены в хранилище данных, является интеллектуальный анализ данных. В отличие от OLAP, интеллектуального анализа данных является сравнительно недавно идея, начиная с 1990-х годов как ответвление исследований CIAL разведки. В интеллектуальном анализе данных, идея заключается в том, чтобы использовать целый ряд различных статистических операций по сбору данных и посмотреть, если результаты имеют смысл. В отличие от OLAP, который приводится в движение с помощью человека, попытки интеллектуального анализа данных несколько случайных исследований этих данных. Там нет никакого способа узнать заранее, если результаты будут иметь смысл. Есть целый ряд различных алгоритмов, применяемых в интеллектуальном анализе данных. Здесь мы рассмотрим некоторые из наиболее распространенных методов.
В кластеризацию, данные группируются вдоль некоторых, но не во всех размеров (полей). При выборе полей с умом, вы можете обнаружить, что данные слипаются в несколько разных групп. Если вы можете определить значение каждого сгустка, вы узнали кое-что о ваших данных. На рисунке 15.2, вы можете увидеть, что данные были сгруппированы с использованием двух измерениях. Например, если эти данные состоят из записей о пациентах, мы могли бы организовали их по возрасту и весу. На основании близости, мы определили три кластера, с двумя

окраинных данными. Один из окраинных данных может принадлежать к группе 2, другой может принадлежать к группе 3, или, возможно, две данные принадлежат к их собственной группе. Возможно, кластер 2 указывает на тех пациентов, основываясь на возрасте и весе, которые более восприимчивы к определенному заболеванию.

Для идентификации кластеров автоматически, компьютер определяет расстояние между отдельными записями. Он находит центр среди групп, а затем добавляет другие элементы данных по одному путем определения, какой кластер он приходит ближе всего к.

При формировании кластеров, мы можем обнаружить, что наш первоначальный выбор полей не дают нам никаких полезных группировок. Например, если мы выбрали рост и вес, а не вес и возраст, мы, возможно, не выявили каких-либо полезных характеристик. Представьте себе, что мы хотим, чтобы идентифицировать пациентов, которые могут быть особенно восприимчивы к диабету типа 2. Поля, такие как возраст, рост, и уровень доходов, вероятно, будет бесполезным. Тем не менее, вес (или, по крайней мере, индекс массы тела), этнической принадлежности и степени стресс на рабочем месте могли бы найти более значимые кластеры.

Это приводит к проблеме. Принимая во внимание возраст, рост, вес, и даже доход достаточно легко построить график, как мы график стресс

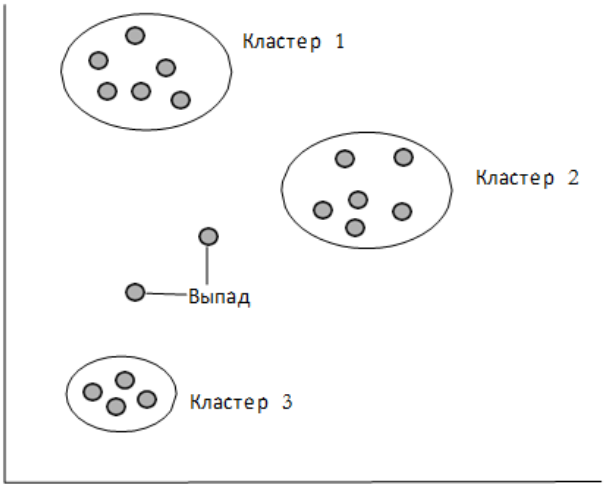


Рисунок 15.2
Результаты кластеризации данных.

на работе и этнической принадлежности? Мы должны найти способ преобразования данных из категорий в числовые значения.

Кластеризация является очень распространенным методом анализа данных, но, как подразумевается здесь, есть проблемы. Нам, возможно, потребуется попытаться сгруппировать по множеству различных сочетаний полей, прежде чем мы найдем какие-либо полезные кластеры. По мере того как число полей увеличивается, число комбинаций возрастает в геометрической прогрессии. Например, с 10 полями, есть 1024 различных комбинаций, которые мы могли бы попробовать, но с 20 полями, число комбинаций возрастает до более чем 1 миллион!
Разновидностью кластеризации заключается в определении ближайших соседей. Ближайшим соседом является элемент данных, который находится ближе всего в близости вычисленной некоторым математическим уравнением (например, евклидово расстояние формулы).[1] Алгоритм ближайшего соседа может быть использован в кластеризацию для записи позиции и идентификации кластеров с помощью тех, чье расстояние меньше, чем некоторые заданное количество. Алгоритм вычисляет расстояния вдоль полей. Выбор, который еще один вычислительно сложную задачу. Например, если имеется 10 полей и к = 6, мы могли бы попытаться использовать все возможные комбинации из шести полей из 10, чтобы определить ближайшего соседа.
Правило обучения Ассоциация выполняет поиск данных для отношений, которые могут представлять интерес, и обеспечивает их. Например, при анализе 10000 квитанции продуктового магазина, мы находим, что 4000 из поступлений показывают, что кто-то купил арахисовое масло, и 3600 из них 4000 расписок показывают, что человек также купил хлеб. Это становится правилом:

Если клиент покупает арахисовое масло, то он покупает хлеб

Правило также имеет частоту 90% (3600 из 4000). Частота описывает, как часто правило было правдой. Частота, однако, не сказал нам, насколько полезным может быть правилом. Учтите, что мы находим 12 человек (из 10000) купили суши и все эти 12 человек также купили белое вино. Несмотря на то частота очень высока (100%), то правило не очень полезно.

1 Евклидово расстояние - формула между двумя данными, 1 и 2, представляет собой данные имеет три значения атрибутов, а, Ь и с.

Интеллектуальный анализ данных для спасения

Интеллектуальный анализ данных в настоящее время является инструментом бизнеса в том, что он может помочь руководству принимать решения и прогнозы. Тем не менее, добыча данных было установлено, чтобы быть полезным далеко за пределы прибыли. Вот несколько интересных случаях использования интеллектуального анализа данных.

Система обнаружения вторжений Минессота анализирует большие объемы данных, относящихся к сетевого трафика, чтобы найти аномалии, которые могут быть интрузии различных типов в компьютерных системах. Эксперименты по более чем 40000 компьютеров в Университете Миннесоты обнаружили многочисленные обкатки попытки и червей.

В биоинформатики, интеллектуальный анализ данных регулярно используется, чтобы помочь просеять через буквально миллиарды кусочков данных, относящихся к человеческому геному. Недавние заявления привели к улучшению диагнозов заболеваний и оптимизации лечения и лучшего моделирования взаимодействия генов. Кроме того, BioSTORM сочетает в себе интеллектуальный анализ данных и веб-информацию для отслеживания возможной пандемии и биотеррористических инцидентов.

Национальное охранное агентство использовало интеллектуального анализа данных на более чем приблизительно 1,9 трлн телефонных записей ищет связей между известными террористами и гражданами. другие методы интеллектуального анализа данных были использованы для создания Темного Web, группа инструментов и ссылки на веб-сайтах, подозреваемых в террористических организаций.

Правоохранительные органы используют интеллектуальный анализ данных для выявления возможных преступлений горячих точках в своих городах, чтобы предсказать, где будущие преступления, которые могут произойти, и, чтобы разыскать преступников от предыдущих преступлений.

Поисковые системы и рекомендательные сайты регулярно используют интеллектуальный анализ данных, чтобы найти ссылки, представляющие интерес, основываясь на ваших входных запросов. Google и Amazon являются пионерами в добыче данных.

Мы можем использовать правило ассоциации, чтобы помочь нам принимать решения, например, в области маркетинга и продаж. Мы могли бы, например, принять решение о переводе масло арахиса в хлеб проход, и мы могли бы решить, поставить на продажу, чтобы продвигать два продукта, говоря "купить хлеб и получить банку арахисового масла на 25%". Результатом обучения ассоциативных правил, как сгустке сеяния, может или не может иметь значение. Правило суши, вероятно, не убедить нас делать ничего особенного о суши и вина. Или, представьте другое правило, которое говорит нам, что люди, которые покупают вино не как правило, также покупают пиво. Есть ли значение в знании этого корреляцию? Преимущество обучения ассоциативных правил является то, что мы можем предоставить данные расписку инструмент интеллектуального анализа данных и дайте ему найти правила для нас. Мы могли бы сказать инструмент, чтобы возвращать только правила, частота которых превышает 80%. Тогда, это до руководства, чтобы решить, как использовать правила.

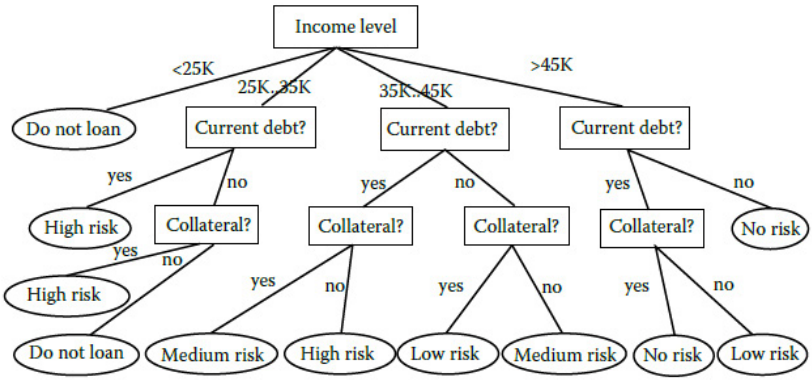


Рисунок 15.3
Пример дерева принятия решений.

Перевод:
income level – уровень дохода
Do not loan – нет кредита
Current debt – текущий долг
High risk – высокий риск
Collateral – ценные бумаги
No risk – нет риска
Medium risk – средний риск

Другой продукт интеллектуального анализа данных является дерево решений. Дерево решений представляет собой древовидную структуру, которая содержит поля базы данных в качестве узлов, таких как возраст, пол и доход. Ветви дерева представляют различные возможные значения полей. Например, если узел имеет подвиды, к примеру две ветви, одна для мужчин и одна для женщин (третья ветвь возможна, если точка привязки для пола не известна или недоступна). Для поля, таких как возраст, а не предоставление ветвь для каждого возможного значения, значения сгруппированы вместе. Так, например, может быть отделение для подростков в возрасте (от 0 до 12 лет), филиал в подростковом возрасте, отделение для молодых взрослых (например, от 20 до 32), отделение для среднего возраста (например, 33 к 58), а также отделение для пенсионного возраста (например, 59 лет и старше). Листовые узлы дерева решений представляют собой те решения, которые мы хотим делать. Дерево решений затем представляет собой процесс принятия решения, следуя соответствующие ветви дерева.

С учетом банковских решений предыдущих кредитов, мы хотим автоматизировать этот процесс. В этом случае, мы строим дерево решений из записей клиентов, учитывая их уровень дохода, имеют ли они текущий долг или нет, и если у них есть залог. Для нашего дерева, эти три поля представляют наши узлы (указанные на рисунке с коробками). Отрасли аннотируются с возможными значениями для каждого поля. Мы сломались уровень дохода на четыре возможных значений, "<25K", "25K ... 35K", "35K ... 45K", и "> 45K". Остальные два поля, текущее и сопутствующее , имеют да / нет ответов. Узлы листьев представляют собой кредитные решения, принятые в прошлом. Эти узлы обозначены овалами. Например, человек, который не имеет текущей задолженности, и доход в размере более $ 45 000 не имеет никакого риска, в то время как человек, чей доход меньше 25 000 $ не следует давать кредит ни при каких обстоятельствах. Человек, чей доход составляет от $ 25000 до $ 35000 и не имеет задолженности, по-прежнему потребует залог, чтобы получить кредит, но даже в этом случае считается высоким риском.
Деревья решений могут быть автоматически генерированы из базы данных. Тем не менее, как это подразумевается в предыдущем примере, если поле имеет больше, чем просто несколько значений (например, возраст, доход), то данные должны сначала быть сегментирован. Это может быть сделано с помощью человека или алгоритма интеллектуального анализа данных. В любом случае, алгоритмы генерации деревьев

решений также интенсивными вычислениями, потому что алгоритм должен решить, какие поля используются в дереве решений (пол и возраст не были использованы в дереве на рисунке 15.3) и для того, что поля применяются. С учетом базы данных отношение 10 полей, существует 1024 различных комбинаций, которые могут быть использованы поля, не говоря уже о миллионы потенциальных упорядочений полей (сверху вниз, в дереве).

Сегодня склады данных являются весьма распространенным способом организовать и использовать данные для крупных организаций. Хранилище данных используется корпорациями, финансовыми институтами, университетами, больниц и медицинских учреждений, телекоммуникационных компаний, авиационной отрасли, разведывательного сообщества и многих правительственных учреждений. Наряду с облачных вычислений (введен в главе 12), хранилищ данных означает сдвиг в масштабе ИТ в поддержку того, что многие теперь называют "Big Data".

Это обсуждение баз данных и хранилищ данных не хватает одной важной характеристикой. Несмотря на то, баз данных и хранилищ данных хранятся с использованием информационных технологий, они на самом деле классифицируются как формы информационных систем. В чем разница? Информационная система представляет собой систему, которая хранит и обеспечивает обработку информации (и данные). Тем не менее, идея информационной системы является нейтральной к технологии. Хотя это правда, что почти все информационные системы сегодня хранятся в компьютере и использовать программное обеспечение для обработки информации, он не должен быть таким образом. Информационная система может быть картотекой системы библиотеки, хранящейся на 3 × 5 карточках или базу данных инвентаризации компании, хранится на бумаге в картотеке. На самом деле, термин информационная система существовала намного дольше, чем компьютеры. Помимо баз данных и хранилищ данных, люди часто рассматривают экспертные системы и системы поддержки принятия решений, географических информационных систем и корпоративных систем как одна из форм информационной системы. Термин информационная технология описывает технологическую инфраструктуру, которая сама может поддерживать информационные системы, но может также поддерживать многие другие типы систем, таких как телекоммуникационные системы, сенсорные сети, и распределенной обработки с именем, но некоторые из них, которые отличаются от информационных систем. Мы кратко

рассмотрим информационные системы как вариант карьеры в главе 16.

Информация и безопасность

Обеспечение информации и информационной безопасности часто описываются вместе как информации обеспечения безопасности (IAS). В этом разделе мы рассмотрим концепции, лежащие в МСФО. Мы оставляем некоторые детали информационной безопасности в разделе Угрозы и решения. Модель IAS объединяет в себе три составляющие IT-коммуникаций, оборудования и программное обеспечение , используемой тремя компонентами. Сосредоточение внимания на том, как информацию используется, МСФО запрещает три цели, как правило, называют ЦРУ: конфиденциальность, целостность и доступность.
Конфиденциальность требует, чтобы данные сохраняются в безопасности, так что они не случайно при условии, неуполномоченным лицам и не могут быть получены посторонними людьми. Конфиденциальность выходит за пределы механизмов безопасности, поскольку она касается вопросов политики, которая является авторизованным пользователем данных, как данные должны быть сохранены и сохранены, и политику конфиденциальности. В качестве примера, мы обсуждали ранее в этой главе необходимость уникального идентификатора. Организация может иметь политику, записи о сотрудниках будет использовать номера социального страхования в качестве уникальных идентификаторов. Тем не менее, такая политика нарушает свободу работника, чтобы защитить их номер социального страхования (который должен только быть доступны для целей начисления заработной платы).
Кроме того, механизмы безопасности должны идти далеко за пределы тех, которые защищают от несанкционированного доступа через компьютерную сеть. Например, конфиденциальность должна гарантировать, что либо конфиденциальные данные не доступны на портативных компьютерах, или что эти портативные компьютеры не должны быть доступны для посторонних лиц. Кража ноутбука не должны нарушать конфиденциальность организации.
Честность требует, чтобы данные являются правильными. Для этого требуется, как минимум, трех различных усилий. Во-первых, сбор данных должен включать в себя компонент, который обеспечивает точность собранных данных. Во-вторых, данные должны быть введены в систему точно. Но самое главное, модификация данные должны быть отслежены. То есть после того, как данные в базе данных, изменения в

эти данные должны оставить записи или проследить, чтобы указать, кто внес изменения, когда было сделано изменение, и что изменение было. Это позволяет аудит данных и возможность катиться данные обратно в случае необходимости. Кроме того, если точка привязки используется одним человеком, ИГД должен быть безопасным от изменения другой. Отсутствие надлежащей синхронизации данных может легко привести к повреждению данных, как уже обсуждалось ранее в этом тексте.
И, наконец, наличие требует, чтобы информация доступна, когда это необходимо. Это кажется очевидным требованием, но конфликты в некоторой степени связаны как с конфиденциальностью, так и с интегрированной опасностью . Мы предполагаем, что большинство организаций сделает их доступной информации в сетевой моде, так что пользователи могут получить данные с компьютеров, расположенных по всей организации, в том числе удаленного доступа из мест за пределами площадки. Для того, чтобы сохранить конфиденциальность, надлежащие меры безопасности должны быть предприняты, в том числе защищенных форм авторизации, шифрования / дешифрования и контроля надлежащего доступа (например, права доступа к файлам). Кроме того, если один пользователь получить доступ к некоторым данным, для поддержания целостности, другие пользователи закрыты из доступа к этим данным, ограничивая тем самым их доступность. В целях содействия целостности, своевременное резервное копирование данных должны быть сделал, но, как правило, в течение времени, что файлы копируемых, они не доступны. Дополнительная сложность для доступности является любой компьютер, время простоя, вызванное при выполнении апгрейд компьютера, перебои с подачей электроэнергии, сбоев оборудования или атак отказа в обслуживании.

2000 год

Целостность данных является достойной целью, но как насчет целостности программного обеспечения? Там были многочисленные программные ляпы, но, вероятно, никто не более хорошо известна, чем сама Y2K. Что такое Y2K, и почему это произошло?
в 1960-х и 1970-х годах, оно было общим для программистов, чтобы попытаться сохранить как можно больше места в памяти, как это возможно в рамках своих программ, поскольку компьютеры были очень ограниченные основные объемы памяти. один из способов сэкономить пространство было ограничить пространство, используемое для хранения в год. а не хранить четырехзначный данным числом, они, как

правило, хранят лет в качестве двузначных чисел. так, например, 31 января 1969 году стал храниться как 01, 31 и 69. никто не понял, что это не будет проблемой, примерно до 1997 года.
С 2000 года приближается, программисты начали понимать, что любая логика, которая занимается сравнением текущего года в каком-то целевом годе, может внезапно быть неправильным. Рассмотрим следующий фрагмент кода, который будет определять, был ли человек достаточно стар, чтобы голосовать:

Если (CURRENTYEAR - год рождения> = 18) ...

Теперь давайте посмотрим, что произойдет, если вы родились в 1993 году и в настоящее время 2013 г. Использование четырехзначных лет, мы имеем:

Если (2013 - 1993> = 18) ...

Это правда. так что код работает правильно. но теперь давайте посмотрим, что происходит с двумя цифрами:

Если (13 - 93> = 18) ...

Это неверно, даже если вы достаточно взрослые, чтобы голосовать, вы не можете!
Y2K была значительную угрозу для нашего общества, поскольку она потенциально может помешать людям получать свои чеки социального обеспечения, вызвать банковское программное обеспечение для вычисления неправильный интерес, и даже предотвратить студентов от окончания! но никто не знал истинных масштабов проблемы, и мы не знали бы, пока ракетные шахты января 2000 года внезапно не запускать свои ракеты? Будет ли энергокомпания автоматически выключать электроэнергию для своих клиентов? Будет ли неисправность медицинского оборудования? Будет ли самолеты падать из неба или оборудование управления воздушным движением закрытой?
поэтому программисты пришлось изменить весь код, чтобы изменить двухзначные до четырех лет. В одних только Соединенных Штатах, эти усилия стоят больше чем $ 100 миллиардов!
Два дополнительных областей, которые, запрещенные некоторые, но не все, в МСФО являются подлинность и безотказности. Это особенно

важно в электронной коммерции ситуациях. Подлинность гарантирует, что данные, информация, или ресурс является подлинной. Это позволяет действительная сделка иметь место (будь то лично или через Интернет). Аутентичность чаще всего осуществляется сертификатами безопасности или другой формы цифровой подписи. Мы рассмотрим их более подробно в Угрозы и решения. Неотрекаемость является юридическим обязательством и следует по договору между двумя сторонами. В электронной коммерции, это означало бы, что заказчик обязан платить за сделки и бизнес, обязан выполнить обслуживания или предоставить объект или услугу. Поскольку эти две области, влияют только на предприятия, они не признаются в качестве принципов ядра МСФО всеми.

МСФО связана в первую очередь с защитой ИТ. IAS объединяет ряд практиках, которые определяют информационные ресурсы организации, уязвимость этих активов, угрозы, которые могут повредить эти активы, а также политики в области защиты активов. Конечный результат этого анализа, известного как стратегического анализа риска, является политика безопасности, которые переводятся в механизмы для обеспечения информационной безопасности. Мы сосредоточимся на процессе управления рисками. Для выполнения управления рисками, организация должна сначала иметь цели. Эти цели используются в несколько этапов управления рисками. Цели часто заявляются только на очень высоком уровне, такие как "организация будет предоставлять высококачественные услуги членам сообщества с целостностью и отзывчивость". Цели могут быть приоритетными, некоторые цели могут вступать в противоречие с другими. Например, цель быть отзывчивым к клиентам может вступать в противоречие с целью обеспечения целостности данных, требует быстрого реагирования, в то время как в последнем случае требуется, чтобы данные тщательно перед использованием.

В настоящее время организация должна выполнить оценку риска. Первым шагом в оценке риска заключается в определении информационных активов организации. Эти физические активы (например, компьютеры, компьютерные сети, люди), интеллектуальная собственность (идеи, продукты), а также информация (концентрированную и обработанные данные). Помимо выявления активов, они могут быть классифицированы по типу (например, аппаратное обеспечение, процесс, персонал) и приоритетность их важности для организации. В случае информации, классификации безопасности иногда применяются. Бизнес может использовать такие категории, как "общественность", "чувствительный",

"частный" и "конфиденциально", чтобы описать данные, которые они собрали на своих клиентов. Общественная информация может включать в себя имена и адреса (так как эта информация доступна через телефонную книгу). Чувствительная информация может включать в себя номера телефонов и адреса электронной почты. Несмотря на то, что это не публичная информация, это информация, которая не будет рассматриваться как угроза неприкосновенности частной жизни человека, если в противном ERS, должны были узнать об этом. Частная информация это информация, которая может представлять собой угрозу, если передаваться третьим лицам, таких как социальное обеспечение и номера кредитных карт или здоровья и образования информации. Эта информация часто защищена от разглашения федеральным законодательством. И, наконец, конфиденциальная информация содержит информацию о том, что организация будет держать в секрете, такую как патентуемую информацию и бизнес-планов. Правительство выходит за рамки этих четырех категориям с группой классифицированных тегов, такие как конфиденциальной, секретной и совершенно секретно.

Следующим шагом в оценке риска заключается в выявлении уязвимостей каждого актива. Уязвимости будет варьироваться между активами, но многие активы будут делить уязвимости, основанные на их категорированного типа. Например, представьте себе, в оценке риска, что три вида активов являются менеджеры, технический персонал, а также административного персонала. Все эти люди, и некоторые из их уязвимости будет таким же. Любое лицо, может быть переманенной в другую организацию на более высокооплачиваемую зарплату, и любое лицо может быть уязвимо для атак социальной инженерии (охватываемых угроз и решений). Аналогичным образом, большинство аппаратных элементы будут разделять определенные уязвимости, такие как уязвимые для отключения электроэнергии и ущерб от скачков напряжения, но не все аппаратные средства будут иметь уязвимости от несанкционированного доступа, потому что некоторые аппаратные средства не могут быть объединены в сеть.

После того, как уязвимости выявлены, оценка рисков продолжается с угрозами. То есть, учитывая уязвимость, каковы возможные способы, которыми уязвимость может быть использована злоумышленниками? Как было указано в предыдущем абзаце, человек может быть уязвимым для инженерной атаки социальной.

Тем не менее, такое нападение на кого-то, имеющего доступ к секретным файлам, является гораздо большей угрозой, чем нападение на кого-то

без такого доступа. Аналогичным образом, несанкционированный доступ к файловому серверу хранения конфиденциальных данных является гораздо большей угрозой, чем несанкционированного доступа к принтеру или портативный компьютер, который используется только для презентационной графики.

С оценкой риска завершено, организация движется вперед, определяя, как обрабатывать каждый риск. Риски, как и цели, должны быть приоритетными. Здесь, организационные цели могут помочь определить приоритеты рисков. Например, одна из организационных целей может быть «вербовать самых лучших и талантливых сотрудников", но цель "высокое качество обслуживания" имеет большее значение. Таким образом, риски, которые ставят под угрозу "услуга" цели будет иметь более высокий приоритет, чем риски, которые угрожают «работник» цели.

В приоритизации рисков, управление рисками может определить, что некоторые риски являются приемлемыми. Это может, например, быть приемлемым риск того, что сотрудники будут переманиваться к другим организациям. Тем не менее, некоторые риски могут быть идентифицированы, которые могут критически повредить организации, такие как атаки на веб-портал организации (например, отказ в обслуживании), что приведет к потере бизнеса в течение некоторого промежутка времени. Риски, в свою очередь, требует создания политики, которая приведет к снижению угрозы. Такая политика должна быть, затем принята. Политика может включать в себя конкретные области организации такие найма практики в области людских ресурсов, методов управления для управления среднего звена, а также технические решения, примененные персоналом ИТ.

После того, как план управления рисками будет реализован, то это должно контролироваться. Например, после того, как 6 месяцев прошло, различный персонал может проверить результаты. Такие санкции или самообучения организации может привести к выявлению областей плана, которые не работают должным образом. Таким образом, процесс управления рисками является итеративным и на постоянной основе. Время между экзаменов будет основываться на нескольких критериях. Первоначально итерации может длиться всего несколько месяцев, пока план управления рисками не работает хорошо. Итерации затем может длиться месяцев до нескольких лет в зависимости от того, как часто могут возникать новые активы, уязвимости и угрозы. План управления рисками может включать в себя широкий спектр ответов. Ответы могут

отличаться в зависимости от физических действий к ИТ-операций для создания новых процедур. Физические действия могут включать в себя, например, установка камер, датчиков и пожарной сигнализации; механизмы блокировки на компьютерах; и найма дополнительного персонала для монитора или карауле. ИТ операции обсуждаются в следующем разделе, такие как межсетевые экраны. Новые процедуры могут включать найма процессы, политики в области использования ИТ и контроля доступа, управления информацией
активы, техническое обслуживание и графики обновление.

Еще одним компонентом плана управления рисками представляет собой план аварийного восстановления. План аварийного восстановления план действий для организации в ответ на бедствие, которое воздействует на организацию до такой степени, что она потеряла функциональность. Есть несколько различных типов стихийных бедствий, а также план восстановления может охватывать все (или несколько) видов стихийных бедствий, или может быть уникальный план восстановления для каждого вида стихийных бедствий. Катастрофа может быть естественной, основанной на суровых погодных условий, таких как смерчи, ураганы, землетрясения и наводнения. Стихийные бедствия могут быть техногенные, но случайно, такие как пожар, коллапса инфраструктуры (например, пол, который коллапсирует в здании), отключения электроэнергии, которая длится дни или химического разлива. Катастрофы могут быть также антропогенными, но целенаправленными, такие как террористический акт или акт войны. Ниже приведены примеры двух последних категорий. Прекращение подачи электроэнергии, которые обрушились на восточное побережье в 2008 году, было признано случайным, но отключение электричества длилось несколько дней для некоторых общин. И, конечно же, 11 сентября 2001 года, террористы разбился четыре самолета в Нью-Йорке, Вашингтоне, округ Колумбия, и Пенсильвании, кото- рое вызывает большую путаницу и приводит к отключений авиакомпании в течение нескольких дней.

План аварийного восстановления должен обратить внимание на три вещи. Во-первых, профилактические меры необходимы в попытке избежать катастрофы. Например, пожарная сигнализация и спринклерная система поможет предотвратить пожар от распространения и повреждения ИТ и инфраструктуры. Надлежащие процедуры при литературе руководство утечки химических веществ могут включать в себя, как справиться с эвакуацией в том числе, как быстро закрыть все ИТ-ресурсы или как передавать возможности ИТ-обработки на другой сайт. Простые

профилактические меры должны включать в себя использование источников бесперебойного питания и сетевых детекторов , чтобы гарантировать, что компьютеры могут быть закрыты должным образом, когда питание вышел, и для защиты от скачков напряжения электричества, которые могли бы повредить компьютеры. Во-вторых, должна быть некоторая мысль, как обнаружить ситуации бедствия в прогрессе. Очевидно, что торнадо или ураган будет заметен, но скачок напряжения не может быть легко обнаружен. И, наконец, план аварийного восстановления должен решить, как оправиться от самой катастрофы.

Сосредоточение внимания на ИТ, лучше подходит для подготовки и обработки их бедствия. Во-первых, регулярные и своевременные резервные копии всех данных должны быть сделаны. Кроме того, резервная копия данных должна быть проведена за пределами площадки. Избыточность следует использовать в дисковом хранилище в максимально возможной степени. Этот вопрос обсуждается более подробно в следующем разделе. всегда следует использовать простые меры, такие как бесперебойного питания и сетевые фильтры на стороне оборудования, а также сильных политик паролей и программного обеспечения противовирусной. Мониторинг оборудования (например, с камерами) и регистрации персонала, которые имеют доступ к оборудованию, которое хранит конфиденциальные данные и также являются полезными действиями. И, конечно же, огнестойкий в виде пожарной сигнализации, огнетушителей, планов пожарной эвакуации, и даже пожаротушения пены все возможно.

Подготовка только одной части плана аварийного восстановления. Восстановление другая половина. Действия включают восстановление резервной копии данных, имеющих свободное ИТ-оборудования (или возможность быстро заменить поврежденное или уничтоженное оборудование), и имеющие службы резервного копирования готовы пойти лишь некоторые из возможных вариантов. Во многих крупных организациях, ИТ распределены по более чем одном сайте, так что катастрофа на одном участке не повреждает всю организацию. Во время стихийного бедствия, другой сайт (ы) забрать груз. Восстановление требует чего исходный сайт резервного копирования и передачи данных и процессов обратно на этот сайт.

План управления рисками часто вместе взятые руководством. Но это очень важно для любой организации, которая опирается на ИТ-персонал, чтобы включить его в процессе. Это, скорее всего, так что управление не будет иметь понимание всех уязвимостей и угроз самой технологии.

Кроме того, по мере возникновения новых угроз довольно часто, это важно, чтобы получить своевременный вклад. Не удивляйтесь, если вас просят быть вовлеченным в управление рисками во времена вашей карьеры.

Проблемы и решения

Информационная безопасность должна защищать в своей основе данные / информацию организации. Тем не менее, окружающие данные имеют несколько слоев. Это операционная система, программные приложения, компьютер и его ресурсы, сети и пользователей. Каждый из этих слоев имеет различные угрозы, и каждый из них будет иметь свои собственные формы безопасности. Поскольку некоторые угрозы происходят на разных уровнях, мы будем рассматривать угрозы, а не слои. Тем не менее, информационная безопасность не будет полным без защиты каждого слоя.

Социальная инженерия представляет собой угрозу, которая предназначается для пользователей. Идея заключается в том, что пользователь является слабым звеном в том, что он или она, может быть, обманута, и часто довольно легко. Простой пример социальной инженерной атаки как это работает. Вы получаете телефонный звонок на дому в один вечер. Голос идентифицирует себя как ИТ и говорит, что из-за сбоя в работе сервера, они нуждаются пароль, чтобы восстановить свой аккаунт. Без вашего пароля, все ваши данные могут быть потеряны. Вы говорите им свой пароль. Теперь они могут проникнуть в ваш счет, потому что они не являются ИТ, но на самом деле кто-то со злым умыслом.

Социальная инженерия используется, чтобы успешно получить пароли людей, номера банковских счетов, номера кредитных карт, номера социального страхования, значения PIN (персональный идентификационный коли- BER), и другую конфиденциальную информацию. Социальная инженерия может быть гораздо более тонким, чем телефонный звонок. В социальном окружении, вы гораздо более вероятно, разглашать информацию о том, что умный хакер может затем использовать для взлома пароля . Например, зная, что вы любите своих домашних кошек, кто-то может попытаться получить имена ваших кошек, чтобы увидеть, если вы используете какой-либо из них в качестве пароля. Многие люди будут использовать любимого имя свое с последующей цифрой для ввода пароля. Вариация на социальной инженерии, чтобы

обмануть пользователя, фальсифицируя информацию в электронном виде.
Фишинг включает в себя электронную почту для людей, чтобы перенаправить их на веб-сайт, чтобы выполнить какую-то операцию. Веб-сайт, однако, это не то, что кажется. Например, кто-то может создавать веб-сайт, чтобы сделать его похожим на сайт кредитной карты компании. Теперь электронная почта отправляется на некоторые из клиентов кредитной карточки компании, сообщив им, что они должны войти в свои учетные записи или же счета будут закрыты. Ссылка заключена в электронной почте, однако, направляет их к спам-сайту . Пользователь нажимает на ссылку и попадает на фальшивый веб-сайт. Там, пользователь вводит информацию (безопасный пароль, номер кредитной карты и т.д.), но, не зная, эта информация становится доступной к неправильному человеку.
Другой класс угрозы нападения компьютерная система сама нацелена ли атака на сети, прикладное программное обеспечение или операционную систему. Этот класс включает в себя атаки протокола, программные средства, эксплойты вторжений и атак изнутри. В атаке протокола, один пытается получить доступ к компьютерной системе, эксплуатируя слабость или недостаток в протоколе. Есть, например, известные проблемы безопасности в TCP / IP (протокол управления передачей / межсетевой протокол). Один из подходов называется TCP угон, в котором злоумышленник подделывает хост-компьютер в сети работу, используя IP-адрес хост-компьютера, по существу, резки, что хост выключен из своей сети. Многие формы атак протокола используются в качестве формы разведки с целью получения информации о компьютерной сети, в качестве прелюдии к реальной атаки. (Протокол Internet Control Message) атака ICMP может использовать программу пинг, чтобы узнать IP-адреса различных хостов в компьютерной сети. После того, как злоумышленник открыл IP-адреса, другие формы нападения может быть запущен. Атака вид атаки, сочетает в себе IP-спуфинга и ICMP (пинг) атака, при которой злоумышленник фальсифицирует IP-адрес другого устройства, чтобы по всей видимости, является частью сети. Таким образом, злоумышленник может обойти некоторые из безопасности
механизмы, которые могли бы победить нормальную атаку ICMP.
Программное обеспечение эксплойты варьируются в зависимости от программного обеспечения, о котором идет речь. Два очень популярных форм эксплоитов SQL инъекций и переполнение буфера. В инъекции SQL, злоумышленник выдает команду SQL на веб-сервере как часть

URL. Веб-сервер, который может принимать запросы как часть URL-адреса, не ожидая SQL команды. Запрос в URL следует за "?", И включает в себя поле и значение, например, www.mysite.com/products. php?productid = 1. В этом случае, страница products.php веб, скорее всего, обращается к базе данных для получения въездной про- ductid = 1. SQL Injection может последовать запрос для работы на этой базе данных. Например, модифицированный URL www.mysite.com/products.php?product = 1; DROP TABLE продукты будут выдавать команду SQL DROP TABLE продукты, которые бы удалить связь из базы данных. Если не защищен от веб-серверов и может передать команду SQL в базу данных. Эта команда SQL потенциально может сделать что-нибудь к базе данных от возвращения защищенных записей удаление записей для изменения значений в записях.

Переполнение буфера, пожалуй, одна из самых старых форм программного обеспечения и эксплойтов хорошо известно, так что разработчики программного обеспечения должны быть в состоянии защититься от этого, когда они пишут программное обеспечение. Тем не менее, это не всегда так, и многие части программного обеспечения по-прежнему восприимчивы к этой атаке. Буфер это просто переменная (как правило, массив), который хранит подборку информационных материалов значений. Буфер имеет ограниченный размер. Если программное обеспечение не гарантирует, что вставки в буфер ограничены его размера, то можно вставить в буфер достаточное количество так, чтобы места памяти после массива заполнены, а также. Поскольку память хранит данные и код, можно было бы попытаться переполнение буфера с вредоносным кодом.

Белая шляпа Versus черной шляпе

Термин хакер передает три разных значения:

- кто-то, кто взломает код, то есть, программист
- компьютер хобби
- кто-то пытается проникнуть в компьютерные системы

Исторически сложилось так, что хакер – это человек, который создает программное обеспечение как средство протеста. Например, ранние хакеры часто ломались в компьютерные работающие телефонные системы, чтобы размещать бесплатные междугородние звонки. Их иногда называют фрикеры.

Для того чтобы дифференцировать более традиционное использование хакера, программиста, с использованием кем-то, кто пытается

проникнуть в компьютерные системы, термин взломщик был придуман. Взломщик пытается взломать компьютерную безопасность. но даже с этим определением, мы должны проводить различие между теми, кто делает это для вредоносных целей из тех, кто делает это либо как вызов или как попытку обнаружить недостатки безопасности, с тем, что безопасность может быть улучшена.

Первый случай теперь называют черной шляпе хакера (или просто черной шляпе). такой человек нарушает безопасность в целях совершения преступления или терроризм.

Последний случай отдельных отключающих систем безопасности без злого умысла называют белой шляпой хакера (или просто белой шляпе). Тем не менее, даже для белой шляпы, действие взлома компьютерной безопасности по-прежнему неэтично.

Как в случае точки, организация крекеров, которые называют себя анонимными является нарушением компьютерных систем различных организации в качестве средства протеста. Они напали на Ватикан в знак протеста против отсутствия католической церкви в ответ на заявления о жестоком обращении с детьми. Они атаковали сайты в ведении Департамента юстиции, Американская ассоциация звукозаписывающих компаний, и Американская ассоциация кинокомпаний в знак протеста против антипиратский законодательного органа, и они напали на сайты PayPal, MasterCard и Visa, когда эти компании заморозили активы Wikileaks. хотя рекламировали, что эти нападения являются формой

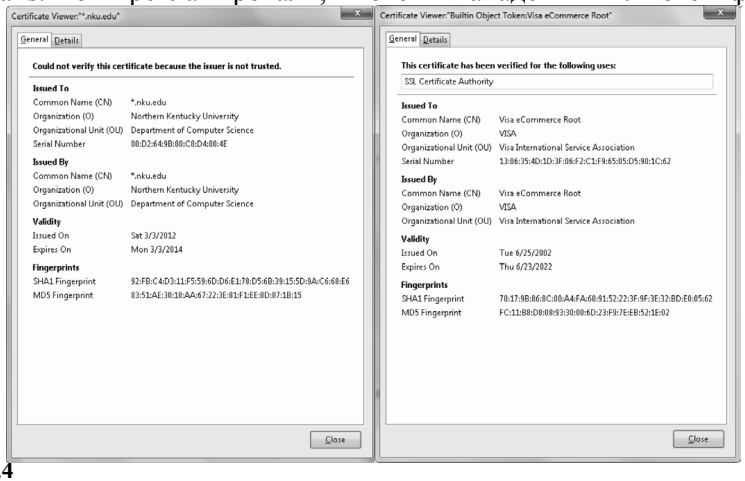


Рис. 15.4

Два сертификата, самозаверенный (слева) и заверенный (справа).

протеста, они также могут рассматриваться как бдительных операций, нарушающих многочисленные международные законы.

Сохраняется в памяти, процессор потенциально может выполнить этот код и, таким образом, выполнять операции, вставленные злоумышленником. Проникновения и другие формы активных атак обычно вращаются вокруг первого, увеличивая несанкционированного доступа в компьютерную систему. Для того, чтобы получить вход, злоумышленник должен попытаться найти брешь в системе безопасности в операционной системе или сети, или получить доступ с помощью учетной записи кого-то другого. Для этого злоумышленник должен знать имя учетной записи пользователя и пароль. Как уже говорилось выше, существуют методы социальной инженерии и фишинг способы получения ПАРОЛЯ слова. Другие способы получения паролей включают написание программы, которая постоянно пытается войти в учетную запись пользователя, пытаясь узнать каждое слово словаря. Другой подход заключается просто следить за человеком, чтобы узнать пароль человека, возможно, наблюдая за человеком ввести его. Или, если человек, как известно, пишут пароли вниз, то вы можете посмотреть вокруг своего компьютера для ввода пароля, если у вас есть доступ в офис этого человека. Можно даже найти пароль, который написан на пост - это к сведению, застрял на мониторе компьютера!

Другим средством получения пароля через анализ пакетов. Здесь злоумышленник исследует сообщение трафик, поступающий с компьютера, например, сообщения электронной почты. Вполне возможно (хотя мы надеемся, что маловероятно), что пользователь может отправить по электронной почте пароль кому-либо или к нему / себя.

Помимо угадывания паролей людей, есть и другие слабые места в операционных системах, которые могут использоваться, чтобы получить вход в систему. Операционная система Unix имела обыкновенные недостатки с программой телнет, которые могли бы позволить кому-то войти в систему Unix без использования пароля вообще. Оказавшись внутри, нарушитель можно затем развязать атаку. Активная атака может потенциально сделать что-нибудь от удаления файлов данных, копирование файлов данных, а также изменять файлы данных оставив после вредоносного кода какого-либо рода или создание учетной записи обратно- двери (скрытый учетной записи, которая позволяет злоумышленнику войти в любое время) ,

Еще более простой подход к прорыву безопасности ИТ-системы, через

внутренние работы. Если вы знаете кого-то, кто имеет авторизованный доступ и любой из них может быть подкупил или влияние, то вполне возможно, что злоумышленник может удалять, копировать или изменять файлы, вставить ее неправильному посуду, или иным образом узнать о характере компьютерной системы через человека , Это, пожалуй, одно из самых слабых звеньев в любой компьютерной системе, потому что люди получают доступ частично потому, что они доверяют. Это доверие, если будет нарушено, может вызвать более значительные проблемы, чем любой формы вторжения.

Вредоносных программ является одним из худших типов атак, совершенных на индивидуальных пользователей. Оригинал форма вредоносных программ был назван троянский конь. Троянский конь симулирует быть одна часть программного обеспечения, но это на самом деле другой. Представьте себе, что вы загрузили приложение, которое вы думаете, будет очень полезным для вас. Тем не менее, программное обеспечение, делая вид, что это приложение, фактически выполняет вредоносные операции в файловой системе. Разновидностью троянский конь является компьютерный вирус. Основные отличия заключаются в том, что вирус прячется внутри другого, умеет, файл, и обладает способностью копировать себя так, что он может копировать себя с одного компьютера на другой через дискеты (назад, когда мы использовали их), флэш-диск, или вложения электронной почты. Тем не менее другие формы вредоносных программ являются сетевые черви, которые атакуют компьютерные сети (смотри, например, Моррис Интернет-червь обсуждается в главе 12) и программ-шпионов. Spyware часто загружается, при доступе к веб-сайтам неизвестным пользователю. Шпионское ПО может следить за вашим поведением просмотра или отчитаться на сайт конфиденциальной информации, например, номер кредитной карты, который вы ввели в веб-форму. Еще одна форма вредоносной программы будет захватить некоторые из вашего программного обеспечения. Например, он может перенаправить информацию DNS, чтобы перейти к другому DNS, который вместо того, отвечать на запросы с правильными IP-адресами обеспечивает фиктивные адреса, которые всегда принимают ваш веб-браузер в неправильном месте (местах).

Одна окончательная форма атаки, которая является общим сегодня, в частности, к веб-сайтам, является отказ в обслуживании нападения. В атаки отказа в обслуживании, один или несколько нападавших пытается затопить сервер с таким количеством входящих сообщений, что сервер не в состоянии справиться с нормальной бизнесе. Один из самых

простых способов, чтобы выполнить атаку отказа в обслуживании, чтобы отправить тысячи или миллионы (или более) HTTP-запросов. Тем не менее, это только увеличивает трафик; это не обязательно ограничивать сервер от ответа на все запросы в течение долгого времени. Атака UDP может заменить содержимое пакета UDP (определенной в главе 12) с другим содержимым, вставленным злоумышленником. Новое содержание может потребовать, чтобы сервер выполнять некоторые трудоемкую операцию. Выполняя UDP наводнения, большие серверы по существу могут быть закрыты. Другие формы отказа в обслуживании используют хендшейкинг протокол TCP / IP, который известен как TCP SYN и атака переполнения TCP ACK.

Приведенные выше рассуждения ни в коем случае не полный список типов атак, которые были опробованы. И, конечно же, новые типы атак прослыть каждого года. Что нам нужно, чтобы способствовать информационной безопасности, механизмы защиты, чтобы ограничить эти угрозы до приемлемых рисков. Растворы привезены из нескольких различных подходов.

Во-первых, пользователи этой организации должны быть образованными. Узнавая о социальном фишинге и форм шпионажа, пользователи могут узнать, как защитить свои пароли. Кроме того, политика должна гарантировать, что пользователи используют только надежные пароли, а также изменить свои пароли часто.

В некоторых случаях организации используют другой подход, чем пароль, который иногда называют "то, что вы знаете". Вместо этого, два других подхода ", что у вас есть" и "кто ты". В первом случае процесс доступа включает в себя владение какой-то ключ. Наиболее распространенной формой ключа является ключ-карта (магнитная карта). Возможно, это может быть использовано вместе с паролем, так что вы должны физически обладать ключом и знать пароль для входа в систему. В последнем случае, "кто ты" представляет собой какой-то физический аспект, который не может быть воспроизведен. Биометрия используется здесь; будь то в виде отпечатков пальцев, голос идентификации матча, или даже для губ печати, метрика не может быть повторена.

Далее, нам необходимо обеспечить, чтобы пользователь получает доступ только к ресурсам, которые пользователь должен иметь доступ. Это включает в себя контроль доступа. Часть ИТ-политики или правления плана рисков человека, должен включать в себя механизм, посредством которого пользователь получает права доступа. Эти права должны включать в себя файлы, которые пользователь должен получить доступ,

пока не включая любые файлы, которые пользователь не должен получать доступ. Права доступа обычно выполняются в операционной системе с помощью какого-либо механизма списка управления доступом (см. главу 6 для обсуждения по этому вопросу). В операционной системе Windows и Linux, контроль доступа ограничивается доступ к файлам и каталогов для каждого пользователя, используя либо списки (Windows, Linux Security Enhanced) или 9-битный rwxrwxrwx схему (Linux). Многие современные СУБД используют на основе ролей управления доступом. В такой системе, роли определены, которые включают в себя список методов доступа. Например, руководитель может быть предоставлен полный доступ ко всем данным, в то время как менеджеры получают доступ к данным своего отдела. Аналитик данные могут быть предоставлены только для чтения доступ к определенным отношениям. После того, как роли определены, индивидуумы назначены роли. Один человек может иметь различные роли, которые позволяют человеку иметь подборку информационных прав доступа.

Большинство других форм атак целевой компьютерной системы. Защита будет представлять собой сочетание технологий, которые обеспечивают защиту от несанкционированного доступа и других форм вторжения, атак отказа в обслуживании, программных эксплойтов и вредоносных программ. Решения включают в себя межсетевой экран, чтобы предотвратить определенные типы сообщений от входа или выхода из сети, антивирусного программного обеспечения, чтобы искать вредоносное ПО, а также программное обеспечение для обнаружения вторжений. Разрешение программного обеспечения эксплойтов требуют выявления первых действий, а затем модификацию исходного кода программного обеспечения. Если программное обеспечение является коммерческим продуктом, или с открытым исходным кодом, как правило, после эксплуатирования найден, обновление выходит, чтобы зафиксировать его в течение нескольких дней до нескольких недель. Атак отказа в обслуживании и вторжений являются более сложные задачи, а ИТ-персонал должен постоянно быть на страже против них.

Два типа угроз возникают при общении защищенной информации через Интернет, это связано ли заполнив веб-форму или отправить информацию по электронной почте. Во-первых, интернет-коммуникации обрабатываются с обычным текстом. Если кто-то может перехватить сообщение, рекомендации защищенной информации, открыта для чтения. Во-вторых, отправитель должен быть уверен, что получатель, кто они говорят, что они, то есть, получатель должен пройти проверку

подлинности. Для решения первой задачи, шифрование может быть использовано. Чтобы решить вторую проблему, мы могли бы использовать третью сторону для проверки подлинности получателя. Для обработки обоих этих решений, мы переходим к цифровых подписей и сертификатов.

Идея сертификата или цифровой подписи является то, что пользователь (организация, веб-сайт) создает знак, чтобы указать, кто они есть. Это просто данные, и могут включать в себя, например, электронной почты подпись, изображение или имя. Далее, данные помещаются вместе в файл. Технология шифрования (рассматривается в следующем разделе) используется для генерации ключа. Ключ и данные передаются в центр сертификации. Это компания, чья работа состоит в том, чтобы гарантировать, что пользователь, который они утверждают, и поместить их печать одобрения на файл данных. Результатом этого является подписанный сертификат или цифровой подписи. Подписанный документ является одновременно доказательством подлинности и шифрования информация, необходимая для отправителя для отправки защищенной информации без вызывающей беспокойство этой информации может быть перехвачен и доступ к ним.

Если пользователь получает сертификат или подпись, которая устарела, не подписаны, или чьи данные не соответствуют заявленной личности пользователя, то получатель получает предупреждение о том, что лицо не может быть, кого они утверждают. Рисунок 15.4 показывает два сертификата. Один слева от университета и само подписан. Вы могли заметить, в верхней части этого сертификата заявление "не удалось проверить этот сертификат, так как эмитент не является доверенным". Это указывает на то, что сайт, который использует сертификат, не потрудился получить подписанный сертификат. Любой пользователь, который получает сертификат, предупрежден, что делать бизнес с этим сайтом может быть рискованно, потому что они не авторизованы. К счастью, этот сертификат не используется для получения защищенной информации, так что пользователь не будет ничего существенного рисковать. Сертификат на правой стороне фигуры от Visa и подписывается SSL Certificate Authority, таким образом, сайт можно доверять.

Информационная безопасность пытается обеспечить конфиденциальность, целостность и уточнить наличие информации, когда она хранится, обрабатывается и передается (по сети). Подходы, перечисленные выше, в первую очередь нацелены на передачу информации по сети и обработки информации. Мы должны использовать

другое решение для обеспечения надлежащего хранения: избыточность. Избыточность предоставляет средства, так что данные доступны, даже когда устройства хранения повреждены или в автономном режиме. Есть два общих подхода к содействию избыточность. Первый через резервные копии файловой системы. Во-вторых, за счет избыточного хранения, в частности RAID.

Резервные копии восходит к самым ранним компьютерным системам. Резервная копия является лишь копией сбора данных. Резервные копии, используемые для хранения на магнитной ленте. Резервные ленты будут храниться в отдельных помещениях, чтобы гарантировать, что если что-нибудь повреждения компьютерной системы, ленты должны быть защищены. Таким образом, эти данные будут легкодоступны для восстановления в случае необходимости. Как ни странно, многие из лент, используемых для датирования хранения данных, обратно в 1950-е годы уже давно деградировали, и данные были потеряны.

Дискеты и оптические диски в конечном итоге заменить ленту для целей резервного копирования. Тем не менее, с низкой стоимостью хранения на жестком диске сегодня, многие пользователи будут производить резервное копирование файловой системы на внешний жесткий диск, или использовать облако хранения. Резервные копии могут быть либо всей файловой системы, или те части файловой системы, которые были изменены с момента последнего резервного копирования. В последнем случае, это известно как инкрементного резервного копирования. Операционные системы могут быть настроены для выполнения резервного копирования автоматически. В рамках плана управления рисками, требуется политика резервного копирования.

RAID означает избыточный массив независимых дисков (в качестве альтернативы, "Я" также может стоять за недорого массива). Идея RAID состоит в создании единого устройства хранения данных (обычно в виде шкафа), который содержит несколько скоординированных жестких дисков.

Дисковые файлы

Дисковые файлы разделены на блоки, как обычно, а блок сам по себе делится на полосы. Блок-диск затем хранится на одном или более дисковых поверхностях в зависимости от размера полосы. Большая полоса будет использовать несколько дисков одновременно, тогда как небольшие полосы будут использовать множество дисков одновременно.

Таким образом, размер полосы определяет, как независимые диски являются. Например, если блок-диск распространяется через любой поверхности диска, а затем доступ к диску блока намного быстрее, потому что все диски доступны одновременно, и, таким образом, каждая поверхность требует меньше времени, чтобы получить доступ к биты, хранящиеся там. С другой стороны, если RAID кабинета должны были использовать сказать восемь отделенный дисков скорости и полоса распространяется через всего два дисков, то RAID шкаф может потенциально использоваться четырьмя пользователями одновременно.
Несмотря на то, "независимый" аспект RAID помогает сократить время доступа к диску, реальная причина для использования RAID является "избыточность". При наличии нескольких накопителей доступны, данные могут быть скопированы, так что, если диск сектора должны были пойти плохо на одном диске, информация может быть повторена в другом месте в шкафу. То есть, избыточные данные могут быть использованы для восстановления данных, потерянных из-за плохих секторов. Избыточность, как правило, занимает от одной из трех форм. Во-первых, есть зеркало формат, в котором один набор дисков дублируется через другой набор дисков. В таком случае, половина дискового пространства используется для обеспечения избыточности. Хотя это обеспечивает наибольшее количество избыточной информации, оно также является самым дорогим, потому что вы теряете половину общего дискового пространства, выступающего в качестве резервного. Два других подхода использовать биты четности (рассматривается в главе 3) и коды Хэмминга для обеспечения избыточности. Оба бита четности и коды Хэмминга обеспечивают степень избыточности, так что восстановление, возможно с минимальным ущербом, но не от крупного ущерба. Преимущество этих форм избыточности является то, что объем памяти, необходимой для избыточности информацией гораздо меньше, чем половина дискового пространства, что делает эти формы дешевле, чем в зеркальном растворе.
RAID технология в настоящее время существует с 1987 года и является обычным делом для крупных организаций. Менее вероятно, что отдельные пользователи будут вкладывать средства в RAID шкафы, потому что они являются более дорогостоящими. Сам RAID состоит из различных уровней. Каждый уровень RAID предлагает различных преимуществ, недостатки и затраты. Уровни RAID идентифицируются по номерам, где большее число не обязательно означает более высокий уровень RAID. Вот краткий взгляд на уровни RAID.

RAID 0 не обеспечивает избыточность, только независимый доступ с помощью нескольких дисков и полос. RAID-1, с другой стороны, это формат, в котором зеркало ровно половина дисков используются в качестве зеркала. Таким образом, RAID 0 и RAID 1 предложение полностью противоположные спектры и избыточности стоимости: RAID 0, который использует все пространство на диске для файлов, является самым дешевым, но не предлагает избыточности. RAID-1, которая использует половину дискового пространства для резервирования, является самым дорогим, но предлагает большую избыточность. RAID 2 использует коды Хэмминга для обеспечения избыточности и использует возможные наименьший размер полосок. RAID 3 использует биты четности для обеспечения избыточности и самых маленьких размеров возможных полос. Кодов Хемминга сложны и отнимает много времени, чтобы вычислить. Кроме того, RAID-2 требует большого количества дисков, чтобы быть успешным. По этим причинам, RAID-2 не используется.
RAID 4, как RAID 3, использует биты четности. Но в отличие от RAID 3, RAID 4 использует большие размеры полосок. В RAID 4, все биты четности хранятся на том же самом диске. Из-за этого, RAID-4 не допускает независимый доступ. Например, если два процесса пытались получить доступ дисковода в то же время, несмотря на то, что два доступы могут быть двумя отдельными полосами на двух различных дисках, оба доступы будут сталкиваться, когда также спускается в один диск, содержащий информацию о четности. По этой причине, RAID 4 не используется. RAID-5 такая же, как RAID 4 за исключением того, что информация четности распределяются по дискам. Таким образом, двух процессов потенциально может получить доступ к двум различным дисковым блокам одновременно, если предположить, что блоки и их информацию четности были на различных дисках. RAID-6 является такой же, как RAID 5 за исключением того, что информация о четности дублируются и распределены между дисками для дополнительного резервирования и независимости.
На рисунке 15.5, RAID 4, 5 и 6 показаны. В RAID 4, все данные контроля четности на один и только один диск. Этот диск станет узким местом для независимых заходов. В RAID 5, данные четности равномерно распределены среди всех дисков, так что вполне вероятно, что несколько дисковых операций могут происходить одновременно. В RAID 6, данные четности дублируются и распределены таким образом, что не только увеличивается избыточность, но вероятность одновременного доступа увеличивается. Вы заметите, что RAID 6 имеет, по меньшей мере, один

дополнительный диск и так дороже, чем любой из RAID 4 или RAID 5, но обеспечивает большую избыточность и, возможно, улучшенный доступ.
Совсем недавно, гибридные уровни RAID вышли, сочетающие уровни 0 и 1, известные как RAID 0 + 1 или RAID 10, и те, которые объединяют 3 и 5, называется RAID 53. С помощью резервного копирования и / или RAID шкафов, мы можем обеспечить избыточность наших файлов и, следовательно, обеспечить доступность и целостность хранения. Тем не менее, это не помогает решить проблему конфиденциальности. Для того, чтобы поддержать это требование, мы хотим добавить еще одну технологию для нашего решения: шифрование. Посредством шифрования, мы можем не только гарантировать, что данные проводятся конфиденциально, но мы также можем передать его через открытую среду передачи данных, таких как Интернет, не опасаясь его перехвата и понял. Поскольку шифрование является сложной темой, мы представим его в следующем разделе.

Криптография

Криптография - наука размещения информации в код, так что он может быть надежно передана. Шифрование представляет собой процесс преобразования исходного сообщения в код, и дешифрование есть процесс преобразования зашифрованного сообщения обратно в

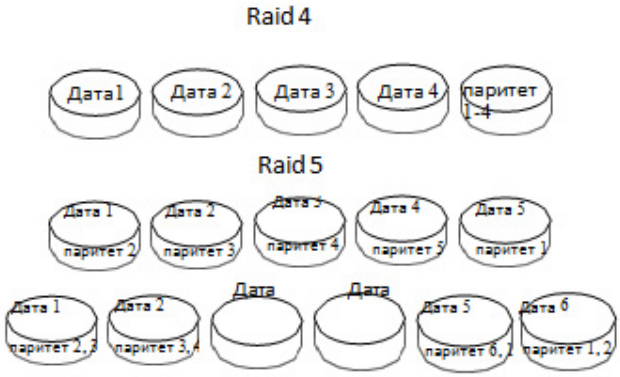


Рисунок 15.5
Данные о паритете, хранящихся в RAID 4, 5 и 6.

Исходное сообщение. Ключ к шифрования, чтобы убедиться, что зашифрованные сообщения, даже если перехватили, не могут быть поняты.

Использование кодов существует в течение многих столетий. В прошлом кодах, или то, что иногда называют классическим шифрованием, в основном были выполнены карандашом и бумагой. Сообщение было написано в коде, послал с курьером, а получатель будет декодировать его. Если курьер был перехвачен (или не заслуживающими доверия), то сообщение не может быть прочитан, потому что это было в коде.

Простой код просто заменяет буквы алфавита с другими буквами. Например, код поворота перемещает каждую букву вниз в алфавите на некоторое расстояние. При вращении + 1, его 'a' становится 'Ъ', а 'Ъ' становится 'c', и так далее, и 'Z' обтекает к 'a'. Во время Второй мировой войны, как немецкие и союзные ученые построили вычислительных устройств в попытке сломать коды других сторон. На сегодняшний день надежного шифрования используются, так как любая другая форма шифрования может быть легко нарушена современных компьютеров.

Есть две основные формы криптографии, используемых сегодня: симметричный ключ и асимметричное ключ. Обе формы шифрования основаны на идее о том, что существует ключ, математическая функция, которая будет манипулировать сообщение в незашифрованном виде и в зашифрованном виде, и из зашифрованного вида обратно в незашифрованный вид. В симметричном ключе шифрования, и тот же ключ используется для шифрования и дешифрования. Эта форма шифрования позиции полезна, только если сообщение (или файл) не предназначен для других, или если получатели могут быть дан ключ. Так, например, вы можете использовать эту форму шифрования для сохранения файла таким образом, что никто другой не может просматривать содержимое файла. Вы бы не разделять ключ с другими, потому что позволит другим пользователям просматривать этот файл.

Рассмотрим случай, когда электронной коммерции в Интернете. Вы хотите передать номер кредитной карты с вашего веб-браузера на веб-сервер компании, так что они могут взимать плату за покупку (ну, вы не можете "хочет", чтобы передать число, но вы хотите, чтобы сделать покупку). Компании, с которыми вы делаете бизнес, хотят, чтобы вы им доверять, поэтому они используют алгоритм шифрования для шифрования номера кредитной карты, прежде чем она будет передана. Чтобы сделать это, ваш веб-браузер должен иметь ключ. Тем не менее, если они обеспечивают ваш веб-браузер с помощью ключа, вы можете

также получить ключ. Если у вас есть ключ, вы могли бы потенциально перехватить другие сообщения на их серверах и разорвать шифрование, получая таким образом номера кредитных карт других клиентов.
Таким образом, для электронной коммерции, мы не хотим использовать симметричный ключ шифрования. На его месте, мы переходим к асимметричным ключом шифрования. В этой форме шифрования, исходная организация генерирует ключ. Это закрытый ключ. Теперь, используя секретный ключ, они могут создать открытый ключ. Открытый ключ может быть свободно передан кому-либо. Учитывая публичный ключ, вы можете зашифровать сообщение. Тем не менее, вы не можете использовать открытый ключ для расшифровки сообщения. Организация сохраняет закрытый ключ для себя, чтобы они могли дешифровать входящие сообщения. Мы обычно называем симметричным ключом шифрования в качестве секретного ключа шифровании позиции, поскольку она использует только один ключ, частный (или секретный) ключ. Шифрование Асимметричного ключа часто называют шифрование с открытым ключом, с помощью двух ключей, закрытый ключ и открытый ключ. Обе эти формы шифрования показаны на рисунке 15.6.

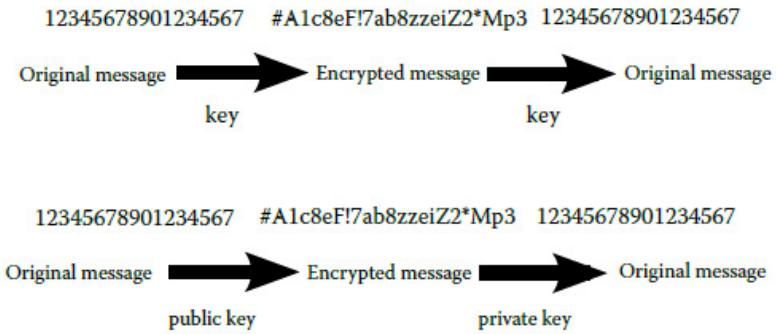


Рисунок 15.6
закрытого ключа шифрования (вверху) в зависимости от шифрования с открытым ключом (внизу).

Перевод:
original message – исходное сообщение
Encrypted message – зашифрованное сообщение
Key –ключ
Public key – открытый ключ
Private key – закрытый ключ

Оба шифрования открытого и закрытого ключей используются обозначения блока. Блок является лишь последовательность из некоторого количества символов в случае компьютеров, мы будем использовать последовательность битов. Итак, мы возьмем некоторые п битов сообщения, которые мы хотим передать и преобразовать N битов, при помощи ключа. Мы делаем это для каждого п-битовой последовательности блока, собираю новое сообщение. Получатель так же разлагает полученное сообщение в п-битовых блоков, каждый блок дешифрования. В качестве простого примера, давайте предположим, что сообщение является "Hello". В ASCII, это сообщение будет состоять из пяти 8-битовых последовательностей (мы добавим от 0 до 7-битного ASCII символа, чтобы каждый символ занимает 1 байт). Это дает нам: 01001000 01100101 01101100 01101100 01101111. Теперь мы берем эти 40 бит и разделить их на п-битовых последовательностей. Для нашего примера, мы будем использовать п = 13. Это дает нам вместо четырех последовательности битов: 0100100001100 1010110110001 1011000110111 1. Поскольку последняя последовательность только один бит, то мы будем его подушечка 0s, чтобы стать 1000000000000.

Ключ будет рассматривать каждый блок в виде двоичного числа. Он будет применять некоторые математические операции (ей) двоичного числа, чтобы превратить его в другое число. Один из подходов состоит в использовании списка целых чисел, например, 1, 2, 5, 9, 19, 41, 88, 181, 370, 743, 1501, 3003, 6081. Поскольку существует 13 целые числа, мы добавим каждый из этих целых чисел вместе, если соответствующий бит в 13-битовой последовательности является 1. например, для 0100100001100, мы добавим 2, 19, 743, и 1501 вместе (потому что только 1 бит в блоке второй , пятый, 10 и 11, и 2, 19, 743 и 1501 являются второй, пятый, десятый, и 11 в последовательности). Сумма 2265. Итак, наш ключ превратил 0100100001100 в 2265, который мы конвертируем обратно в двоичный файл для передачи.

Обратите внимание на то, что наша последовательность из 13 целых чисел не только в порядке возрастания, но и для любого числа, оно больше, чем сумма всех чисел, которые предшествуют его. Например, 88> 4119 + + 9 + 5 + 2 + 1. Это очень важно, чтобы быть в состоянии легко расшифровать наш номер. Для расшифровки номер требуется только найти наибольшее число в ключе меньше или равно числу, вычитанием это значение, а затем повторять до тех пор, пока не достигнем 0. Таким образом, чтобы расшифровать 2265, мы делаем следующее:

Наибольшее значение <= 2265: 1501 2265 - 1501 = 764.
Наибольшее значение <= 764: 743. 764 - 743 = 21.
Наибольшее значение <= 21: 19. 21 - 19 = 2.
Наибольшее значение <= 2: 2. 2 - 2 = 0. Done.
Таким образом, мы расшифровали 2265 в 1501, 743, 19 и 2, или двоичное значение 0100100001100. Для шифрования с открытым ключом, мы должны применить некоторую дополнительную математику. Во-первых, мы используем первоначальный список номеров (наш секретный ключ) для создания открытого ключа. Открытых ключей чисел не будет иметь такое же свойство все более присадка. Например, открытый ключ может быть 642, 1284, 899, 1156, 643, 901, 1032, 652, 1818, 940, 2266, 552, 723. Номера в нашем открытом ключе, наши частные номера позиций, превратившись, используют два дополнительных значения, некоторые из больших простых чисел.
Мы можем передать наш открытый ключ кому-либо. Учитывая открытый ключ, тот же процесс используется для вычисления суммы заданному двоичному блоку. Для нашего предыдущего бинарного блока (0100100001100), сложить соответствующие номера в открытом ключе, а не закрытым ключом. Эти цифры, опять же, соответствующие второму, пятому, 10 и 11-го числа, являются 1284, 643, 940 и 2266. Эта сумма 5133. Вместо передачи 2265, мы передаем 5133. С помощью открытого ключа, 5133 очень трудно для расшифровки, потому что наши публичные ключевые цифры не имеют то же свойство аддитивности. На самом деле, так как цифры не в таком порядке, чтобы расшифровать число требует перебора всех возможных комбинаций значений в нашем открытом ключе. Есть 213 различных комбинаций из этих 13 чисел. Например, мы могли бы попробовать первый номер ключа общественности само по себе, 642. Это не равно 5133, так что теперь мы можем попробовать первые два числа, 642 + 1284. Это не равно 5133, так мы в следующий раз попробовать первый и третьи чисел, 642 + 899. Это еще не 5133, так, что теперь мы пытаемся первые три цифры, 642 + 1284 + 899. в конечном счете, мы, возможно, придется попробовать все 213 различных комбинаций, прежде чем мы попали на правильное решение.
Вы можете вспомнить из главы 3, что 213 равно 8192. Для человека, пробуя все эти комбинации, чтобы расшифровать один номер, 5133, займет довольно много времени. Компьютер, однако, будет иметь возможность попробовать все 8192 комбинаций в гораздо меньше, чем на секунду. По этой причине, наши сильные технологии шифрования сегодня используют блоки, по крайней мере, 200 бит вместо чего-то,

как короткий на 13 бит. С 200 бит, это потребовало бы тестирования, по меньшей мере, 2200 различных комбинаций для того, чтобы расшифровать наш номер. Это значение, 2200, огромное количество. Это больше, чем 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000, 000,000,000, 000,000,000!

Если наше зашифрованное значение так трудно расшифровать, каким образом получатель его расшифровать? Помните, что получатель имеет закрытый ключ. Мы должны преобразовать полученный ряд первых путем применения двух простых чисел. Это будет конвертировать 5133 в 2265. И теперь мы используем наш закрытый ключ, чтобы легко и быстро расшифровать номер. Независимо от размеров блока 13 или 200 ключевые виды использования, расшифровка занимает мало времени на все.

Не было объяснено выше, является использование простых чисел для создания открытого ключа и преобразовать принятые номера. Применим математическую операцию, известную как хэш-функция. Хэш-функция отображает большой ряд значений в меньшем диапазоне. Хэш-функция уже давно используются для создания хэш-таблицы, форма хранения данных в памяти для быстрого поиска, а также используются для создания кэшей, фильтров и информации коррекции ошибок (контрольных сумм). Идея заключается в том, чтобы принять исходное значение и заменить его с целым числом. Это уже сделано для нас, используя шифрование с открытым ключом, но если мы должны были преобразовать сказать имя человека, мы могли бы сложить значения ASCII символы в имени. Далее, мы используем оператор по модулю разделить целое число от простого числа. Напомним, по модулю (по модулю) выполняет деление и дает нам целый остаток. Например, если наше простое число 1301, то 5133 мод 1301 = 1230. Результат от нашей работы мод будет новым числом от 0 и простое число минус один (в данном случае, число от 0 до 1300).

Есть много популярных стратегий шифрования, используемые сегодня. Здесь мы коснемся некоторых из них. Ниже перечислены все ключевые алгоритмы шифрования частные.

• Data Encryption Standard (DES) -developed в 1970-е годы, этот алгоритм использует 56-битные размеры блоков. Он был использован Национальным агентством по безопасности, но в настоящее время считается небезопасным.

• Advanced Encryption Standard (AES) последующие от DES, он использует замещение-перестановку сети, которая может быть легко реализована в программных и аппаратных средств для быстрого шифрования и

дешифрования. Он использует 128-битный размер блока, но ключи могут взять на себя размеры от 128 бит до 256 бит с шагом 32-разрядных.
• Triple DES-хотя DES считается небезопасным, поскольку он может быть разорван в считанные часы, Triple DES использует три ключа DES последовательно для шифрования и дешифрования соответствующее сообщение. Например, если ключи K1, K2, K3 и, шифрования сообщения осуществляется путем применения K3 (K2-1 (K1 (сообщение)), где K2-1 означает использовать K2 для дешифрования сообщения. Дешифрирование затем выполняется с помощью K3-1 (K2 (K1-1 (зашифрованное сообщение))). Каждая клавиша представляет собой 56 бит в размерах и работает на 64-битных блоков. Тем не менее, применение трех клавиш, как показано здесь значительно увеличивает сложность кода , что делает его гораздо оказывается сложнее сломать. несмотря на вызов, то результат будет примерно эквивалентен 112-битного ключа, который также может быть нарушена с современными компьютерами.
• Message-Digest Algorithm (MDN) наиболее последняя версия является MD5. Этот алгоритм шифрования 128-бит, использует хэш-функцию, применяющий 32-значный шестнадцатеричный хэш-номер. Однако, между ограниченным размером ключа и дефектов, обнаруженных начиная с 1996 года, MD5 больше не используется и был заменен SHA-1.
• ША-0 и SHA-1 подобные алгоритмы, которые основаны на 160-битовых блоков. Как и MD5, они дайджеста сообщения алгоритмы, применяемые хэш-функции. И хотя они не имеют недостатки, найденные в MD5, они по-прежнему имеют свои слабые стороны. Сегодня, SHA-2 используется федеральным правительством, повышение на SHA-1 путем объединения четырех различных хэш-функций, размеры которых выше 224 бит за штуку. SHA-3 находился в стадии внедрения в 2012 году.
Два из наиболее часто используемых ключевых алгоритмов шифрования публичных являются RSA и DSA (алгоритм цифровой подписи). RSA, названный в честь трех человек, которые написали алгоритм в 1978 году, использует два больших простых чисел для генерации открытого ключа из закрытого ключа. Эти два простых числа, вместе с закрытым ключом, должны быть надежно защищены. АДС является более сложным, чем алгоритм RSA. АДС является стандартный алгоритм, используемый правительством США для шифрования с открытым ключом.
Эти алгоритмы используются в ряде различных программных и протоколов. Здесь мы кратко рассмотрим на них.

• EPW (Wired Equivalent Privacy) был выпущен в 1999 году в качестве

средства, чтобы позволить зашифрованного беспроводной эквивалент передачи к тому, что можно найти в проводной сети. Используя двадцать шесть 10-разрядных значений в качестве ключа, было установлено, имеют многочисленные недостатки, и был заменен сначала WPA, а затем WPA2.

• WPA и WPA2 (Wi-Fi Protected Access) протоколы используются для шифрования коммуникации между ресурсами в беспроводной сети. WPA2 заменен WPA около 2004. WPA2 основан на алгоритме AES. WPA и WPA2 были предназначены для работы с беспроводным оборудованием, произведенных до выпусков протоколов, так что протоколы имеют обратную совместимость с более старым оборудованием. Есть известные недостатки безопасности как с WPA и WPA2, особенно в использовании функцию под названием Wi-Fi Protected Setup. Избегая этой функции, как WPA и WPA2 являются гораздо более безопасным.

• SSH является безопасная оболочка в Linux, замена небезопасных протоколов Telnet, Rlogin и RSH связи. Он использует шифрование с открытым ключом. Несколько версий существуют, как правило, называют SSH (или SSH-1), SSH-2, и OpenSSH. SSH использует порт 22, тогда как Telnet использует порт 23. SSH также используется для обеспечения безопасной передачи файлов с SCP (защищенный от копирования). Альтернативная форма SSH является SFTP, безопасный протокол передачи файлов. Это отличается от использования FTP внутри SSH сессии.

• SSL и TLS являются Secure Socket Layers и Transport Layer Security, соответственно. TLS в значительной степени заменяет SSL. Эти протоколы связи используются для улучшения транспортного уровня TCP / IP, предоставляя возможности шифрования / дешифрования, которые отсутствуют в TCP / IP. Связь может быть сделано с помощью секретного ключа шифрования или шифрование с открытым ключом. В последнем случае, ключи предоставляются в течение / IP рукопожатия TCP и через сертификат, отправляемый сервером. SSL и TLS используются в ряде различных протоколов, включая HTTPS, SMTP (электронная почта), Session Initiation Protocol (SIP), используемый в передачи голоса по IP и VPN связи.

• HTTPS является безопасной формой HTTP через порт 431 вместо порта 80. HTTPS требует передачу сертификата от сервера к клиенту (вернуться к рисунку 15.4). Сертификат используется двумя способами. Во-первых, он содержит открытый ключ, чтобы клиент мог шифровать сообщения. Во-вторых, если он подписан органом сертификации, он

проверяет подлинность, что сервер это заявляет. Сертификат устарелый, самосогласован и подписанный сертификат, или сертификат не на всех будет запрашивать веб-браузер клиента выдавать предупреждение.

Законы

Мы рассмотрим эту главу, рассматривая несколько важных законодательных актов, действующих со стороны правительства США, стоит отметить из-за их влияния на информационной безопасности. Это отнюдь не полный список, а также веб-сайты, которые содержат более полный перечень законов, приводится в разделе читайте дальше. Этот список не включает в себя законодательство других стран, но вы найдете аналогичные законы в Канаде и Соединенных штатах

• Закон о приватности 1974-самой ранней части законодательного органа, которая относится к сбору и использованию персональных данных, этот акт ограничивает действия федерального правительства. Закон гласит, что:
Ни одно ведомство не должно раскрывать любую запись, которая содержится в системе записи с помощью любых средств связи любого лица или другому учреждению, за исключением того, в соответствии с письменным запросом на, или с предварительного письменного согласия, индивидуума, которому звукозаписывающие относится .... [2]
Закон также устанавливает, что федеральные ведомства должны публично предоставлять данные, которые они собирают, причину для сбора данных, а также возможность для любого человека, чтобы обратить внимание на данные, собранные на них, чтобы обеспечить ее целостность. Есть исключения к закону, такие как Бюро переписи населения и правоохранительных органов.
• Закон о семье обучения правам и конфиденциальность 1974 (FERPA) -Дает студентам право на доступ к их собственные образовательные документы, вносить изменения в эти записи, должны быть сведения про отсутствие или неточностях, и предотвращать школы от распространения записей студентов без предварительного разрешения. Это важный акт, поскольку он препятствует способности делиться информацией о студенте (например, классов) с родителями учащихся, если явно не разрешено студентов. Кроме того, она требует авторизации студента,

2 Вы можете найти полный закон о веб-сайте Министерства юстиции на http://www.justice.gov/opcl/privstat.htm.

чтобы освободить стенограммы, чтобы, например, аспирантур или потенциальных работодателей.

• Закон о внешней разведке наблюдения 1978-установленных стандартов и процедур для использования электронного наблюдения, когда наблюдение происходит в Соединенных Штатах.

• Электронные коммуникации Закона о конфиденциальности 1986-подобного Закона об оперативно-розыскной внешней разведки, этот устанавливает правила и требования для выполнения электронных прослушку через компьютерные сети (среди других форм прослушивания телефонных разговоров).

• Компьютер Matching и конфиденциальности Закона о защите от 1988-вносит изменения в Закон о неприкосновенности частной жизни 1974 года путем ограничения использования других программ сопоставления базы данных и для соответствия данных между различными базами данных. Без этого акта, одно агентство потенциально может получить доступ к данным из других учреждений для создания профиля на конкретного индивидуума.

• Водители конфиденциальности Закона о защите от 1994 года, запрещает государствам от продажи данных, собираемых в процессе регистрации водителей с водительских прав (например, адреса, социальные номера ценных документов, рост, вес, цвет глаз, фотографии).

• Закон об ответственности 1996 года медицинского страхования Портативность и (HIPPA) необходимость национального стандарта для электронных записей здравоохранения среди других стратегий (например, ограничения, что лечащий врач может разместить на сотрудника, который имеет условие страховки). Подобно FERPA, HIPPA запрещает агентствам распространения информации медико-санитарной помощи лица без согласия этого лица. Это одна из причин, почему идти к врачу сегодня требует прежде заполнения документов решения вопросов конфиденциальности.

• Закон Digital Millennium Copyright 1998 (DMCA) два во всем мире договора об авторском праве, что делает его незаконным, нарушать авторские права на распространение оцифрованного материала над компьютером. Этот закон включает в себя положение, которые делают его незаконным размещать на веб-сайтах или загрузить с веб-сайтов, защищенных авторским правом материал, первоначально существовал ли, что материалов, в текст, звук, видео или программного кода. Кроме того, закон снимает ответственность с интернет-провайдеров, которые предоставляют доступ к Интернету для преступника и возбуждено

уголовное дело по этому акту. В результате этого закона, YouTube не несет ответственности за информацию, размещенную защищенной авторским правом на сайте YouTube различными пользователями; Однако, в результате, YouTube часто заказывает удаление защищенных авторским правом материалов по просьбе владельца авторских прав.

• Цифровая подпись и Закон об электронных Authentication 1998 года (SEAL) -permits использование удостоверенных цифровых подписей в финансовых операциях и требует соблюдения цифровых механизмов подписи. Он также ставит стандарты на криптографических алгоритмам, используемых в цифровых подписей. За этим последовало спустя 2 года с электронными подписями в национальной торговле и Global.

• Безопасность Нарушения Уведомления Закона с 2002 года, большинство американских штатов приняли законы, которые требуют какой-либо компании (в том числе некоммерческих организаций и государственных учреждений) уведомлять об этом всем сторонам, чьи записи данных могут быть взломаны, потеряны или украдены, в том случае, если такие данные были как в незашифрованном виде и компромиссном. Например, если файлы были сохранены на переносном компьютере, и компьютер пропал без вести, кто-то, кто потенциально имел запись хранения в файлах на этом компьютере, должны быть уведомлены о действии.

• Реформа разведки и Закон о предотвращении терроризма 2004-требует, насколько это возможно, что спецслужбы обмениваются информацией собраны и обработаны Пер- содержащих потенциальных террористических угроз. Этот акт был прямым ответом на неудачи сообщества разведки, чтобы предотвратить 9/11 террористических атак.

В дополнение к указанным выше актов и законов, правительство США ввел ограничения на экспорт криптографических технологий. Например, физическое лицо или компания не в состоянии продать или обменять определенные криптографические алгоритмы, которые слишком трудно разбиться

Правительство США. Правительство также приняло законодательное собрание, чтобы обновить многие законы, чтобы привести их в компьютерный век. Например, кража компьютера теперь считается кража, а не какой-либо другой категории. Существуют также законы, которые охватывают цензуру и непристойности в отношении к Интернету и электронной почте.

ГЛАВА 16.
КАРЬЕРА В ОБЛАСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ИТ-карьера

Учебник заканчивается там, где он начался: путем изучения ИТ-карьеры и карьеры, связанные с ИТ-сферой. В этой главе различные области ИТ рассматриваются с акцентом на конкретные роли, которые требуют каждую область. Среди деталей совместно являются формами сертификатов, что ИТ-человек может стремиться и недавние зарплаты. Как студенты ИТ, возможно, не дали много мыслей к своей карьере, когда они начинают колледж, текст описывает различные темы, которые могут продлить интерес читателя. К ним относятся такие темы, как ИТ-этики, а также социальные проблемы в цифровом мире, таких как цифровой разрыв, интернет-цензуры, и интернет-зависимости. Глава завершается обсуждением различных форм непрерывного образования для ИТ-человека.

Цели обучения данной главы являются

• Обсудить карьеры в области ИТ.

• Описать формы непрерывного образования в области ИТ.

• Внедрение социальных проблем, связанных с ИТ.

• Обсуждение этических норм поведения в сфере информационных технологий.

Вы заинтересованы в ИТ из-за денег? Возможно. Скорее всего, вы выросли как пользователь технологии. Ваш интерес, возможно, начался из-за социальных средств массовой информации или от работы или школьных компьютерных игр. Или, возможно, вы заинтересовались из-за друзей и семьи. Какими бы ни были интересы, это привело вас в технологии, вы, скорее всего в этом заинтересованы, потому что вы влюбились в использование технологии. Вы, вероятно, аналитически опрошены решать проблемы. Это общий путь, который ведет к ИТ-карьеры. К счастью для вас, карьера в области ИТ не только легко доступна, но может привести к высокооплачиваемую и удовлетворяющую карьеру!

Здесь мы ориентируемся на ИТ-карьеры. Позже в этой главе мы рассмотрим некоторые аспекты ИТ-карьеры, о которых вы, возможно, не задумывались. Помимо технологии, помимо необходимости постоянно

обновлять свои знания и улучшить свои навыки, есть вопросы, которые вы можете столкнуться: правовые, этические, социальные.
Карьера в ИТ высоко оплачиваемая, а перспективы роста числа рабочих мест огромен в течение многих лет. Таблица 16.1 показывает прогнозы роста числа рабочих мест и заработной платы информацию для ряда различных связанных с компьютером карьеры. Обратите внимание на то, что позиции компьютерного программиста, ожидается сокращение поля разработки программного обеспечения и переходит от спроса на программистов к спросу на разработчиков программного обеспечения.

Таблица 16.1 США Компьютер Карьера Работа Рост (Цифры в тысячах)

Заглавие	**2008**	**2018 (Est.)**	**Inc.**	**Inc. %**	**средняя 2008 зарплата**
Программисты	426.7	414.4	–12.3	–2.87	$69,620
Компьютерное программное обеспечение инженеров, приложений	518.4	689.9	175.1	34.01	$85,430
Компьютерное программное обеспечение инженеров системы	394.8	515.0	120.2	30.44	$92,430
Программного обеспечения	532.2	640.3	108.1	20.31	$75,500
Компьютерные системы аналитики	120.4	144.7	24.4	20.26	$69,740
Администраторы баз данных	339.5	418.4	78.9	23.23	$66,310
Сетевые системы и компьютерные системы	292.0	447.8	155.8	53.36	$71,100

Из карьеры, перечисленных в таблице 16.1, последние три карьеры, которые ИТ могли бы рассмотреть. Администраторы баз данных, сетевых администраторов и системных администраторов все проецируется на высокие темпы роста до 2018 года, и нет никаких оснований ожидать, что эта тенденция не будет продолжаться дальше в будущее. Другие карьеры, не перечисленные здесь, включают веб-программы, веб-сайты администратора, сетевой архитектор и специалист по компьютерной безопасности.
Все эти карьеры требуют высокие степени технического мастерства. Эти навыки включают в себя понимание компьютерных систем, операционных систем, программного обеспечения и аппаратных средств.

Кроме того, ожидается, что ИТ-персонал будет решать проблемы. Они также должны быть самостоятельной закуски и на протяжении всей жизни. Кроме того, очень важно для всех ИТ-специалистов, чтобы быть опытным в письменной и устной коммуникации, совместной работы группы и руководства, имеют твердое понимание ИТ-этики, и желание продолжать развивать свои навыки и образование. Фундамент, в деловой практике может помочь дальнейшей карьеры работы ИТ-специалиста.
Сегодня большинство организаций, желающих нанять ИТ-персонала, предпочитают выпускников 4-го курса ITED университетских программ. Выпускники этих программ получили гораздо более широкое обучение, чем катион обучения тех, кто только сосредоточен на технических навыках (2 года технического, профессионально-технического, или местного колледжа). Тем не менее, 4-летнее IT обучение все еще несколько редкое, поэтому организации глядя на работу ИТ-персонала, будет иногда смотреть на смежные дисциплины), как компьютерные науки, деловое администрирование или управление информационными системами (ИС) и информатика. Студенты, степени не соответствуют ИТ-профилю и могут все еще быть в состоянии желаемыми, если они приняли соответствующие курсовые в компьютерных системах, сетях, базах данных, и так далее, или подобрали подходящий опыт.
Широта ответственности в этих карьерах часто зависит от размера организации. В больших организациях, вы можете обнаружить лица, исключительно ответственностью находится в области ИТ-специальности, например, сетевой безопасности или администрирования веб-сервера. В небольших организациях, это не редкость для одного человека, чтобы нести ответственность за несколько ИТ-ролей. Здесь мы ориентируемся на каждом из типов позиций, что ИТ-специалист может иметь и роли и обязанности каждой из этих позиций. Кроме того, мы будем рассматривать формы сертификации (если таковые имеются) и зарплатные ожидания.

Сетевое администрирование

Сетевые администраторы отвечают за установку и техническое обслуживание оборудования и программного обеспечения, которые составляют компьютерную сеть. Сетевое оборудование включает в себя специализированные сетевые устройства, такие как коммутаторы, маршрутизаторы, концентраторы, межсетевые экраны, адаптивных устройств безопасности, точки беспроводного доступа и передачи

голоса по IP (VoIP) телефонов. Сетевое программное обеспечение включает в себя различные операционные системы, установленные на этих аппаратных устройств, протоколы сообщаются по сети, а также приложения. Это включает в себя протоколы маршрутизации, списки доступа, протоколы агрегирование портов, охватывающих протоколы дерева, виртуальные локальные сети, виртуальные частные сети, IPv4, IPv6 и системы обнаружения вторжений. Сетевые администраторы могут быть ответственны за установку, настройку и поддержание такого программного обеспечения. Сетевые администраторы могут также потребовать защиты сети; эта задача может быть выполнена в сочетании с другими ИТ-специалистов, таких как системные администраторы.

Если сеть замедляется или выключается, администратор сети может иногда найти себя в роли поиска и устранения неисправностей высокого давления. Администратор должен использовать свои навыки и знания для диагностики проблем, выявления потенциальных решений, и удержи выбранного решения. Решение должно наиболее эффективно решить данную проблему с точки зрения удобства пользователя, эффективного доступа и разумной цене. Кроме того, вопросы и решения должны быть хорошо документированы.

Высшие позиции администрирования сети уровня, который иногда называют как сетевые или архитекторами сетевых инженеров, часто отвечают за проектирование сетей, отвечающих максимальной мощности и емкости потребности организации. Эта работа включает в выборе соответствующего сетевого оборудования и планирование распределения сетевых адресов в масштабе всей организации. Сетевые администраторы требуют все базовые знания стека TCP / IP, а также другие сетевые протоколы, которые следуют модели открытых систем Interconnect (OSI). Тем не менее, сетевые администраторы часто сосредотачиваются на специальности, такие как сетевая безопасность, беспроводные управления, VoIP, управление интернет-провайдеров, а также надежность сети и планирование мощностей.

Из этого списка обязанностей, то очевидно, что сетевые администраторы требуют значительного количества подготовки и опыта, чтобы быть успешным в своей роли. Из-за изменяющих характер технологии, сетевые администраторы должны постоянно изучать новые технологии через торговые журналы, конференций, публикаций и непрерывного образования. Различные карьерные сертификации, в этих областях специальности, являются ценным дополнением к возобновлению административной сети. Среди наиболее востребованных сертификаций

являются CCNA (Cisco Certified Network Associate), CCNP (Cisco Certified Network Professional), Novell CNE (Certified Novell инженер), CWNA (Certified Wireless Network Administrator) и CompTIA Сеть +. Зарплаты для сетевых администраторов (по состоянию на 2008 год) в диапазоне от $ 41000 до столь же высокой как $ 104000 и средняя зарплата $ 66310. Управление сетями компании в данный момент ключа платит около $ 3000 больше, чем работать для телекоммуникационных операторов, которые платит существенно лучше, чем аналогичные рабочие места, работающих в школах любого уровня.

Администрирование системы

В то время как сетевые администраторы отвечают за выполнение сетевых аппаратных средств и программного обеспечения, системные администраторы отвечают за установку и поддержание ресурсов в этой сети. Роль системного администратора охватывает аппаратные средства, программное обеспечение и конфигурацию системы. Компьютеры могут включать в себя мэйнфреймы, миникомпьюторы ERS, а также персональные компьютеры, а также сетевые серверы. Администраторы системы также должны быть знакомы с несколькими операционными системами, как вполне вероятно, что у организации есть компьютеры нескольких платформ.

Обязанности разнообразны и варьируются в зависимости от организации. Системный администратор будет, как минимум, устанавливать и настраивать операционную систему на автоматах организаций, в установке нового оборудования, осуществлять модернизацию программного обеспечения и технического обслуживания, обрабатывать администрирование пользователей, выполнять резервное копирование и восстановление, а также выполнять систему мониторинга, настройки и устранения неполадок. Администратор системы может также быть связаны с разработкой приложений и развертывания, системы безопасности и надежности, и должны работать с сетевыми файловыми системами. Системный администратор, скорее всего, придется выполнять некоторые программы с помощью сценариев оболочки, так что повторяющиеся задачи можно автоматизировать. Большее количество программ может потребоваться в зависимости от размера и сотрудников организации. Разработка политики и планирование ИТ-инфраструктуры могут быть также задачи, возложенные на системных администраторов.

Роль системного администратора могут перекрываться или полностью подводить требования администратора сети в зависимости от размера организации и необходимости. Небольшие организации будут нанимать меньше ИТ-персонала, и, таким образом, обязанности могут быть более разнообразными. С другой стороны, небольшие организации, вероятно, имеют потребности меньшие из-за небольших сетей, меньше ресурсов компьютера, и, возможно, меньше ситуаций, которые требуют внимания системного администратора. Квалифицированный системный администратор будет обращаться к любому числу доступных веб-форумах, которые как и системные администраторы, обсуждают текущие проблемы и ищут решения.

Как и в случае с сетевым администратором, системный администратор потребует большой подготовки и опыта, чтобы гарантировать, что компьютерные системы работают эффективно, эффективно и надежно. Системный администратор должен играть активную роль в его или ее дальнейшего развития. Как администратор сети, есть несколько шансов для сертификации, а также торговые журналы, конференции и публикации. Возможности Сертификационные включают MCSA Microsoft (Microsoft Certified Systems Administrator), MCSE (Microsoft Certified Solutions Expert) и MCITP (Microsoft Certified IT Professional), Red Hat, RHCE (Red Hat Certified инженер) и RHCSS (специалист Red Hat Certified Security) и CNA компании Novell (Novell Certified Administrator), а также сетевые сертификаты, упомянутые ранее.

Зарплаты для системных администраторов находятся в том же диапазоне, что и основной сети администратора. Эти две категории рабочих мест в сочетании обещание увидеть увеличение спроса, по крайней мере, до 2018 года с десятками тысяч новых позиций будут открываться .

Веб-администрирование

Веб-администратор, также известный как веб-мастер, но более точно называют администратор веб-сервера, отвечает за поддержание веб-сайтов. Это отличается от развития веб-сайтов (см Web Developer в других смежных профессий). В частности, веб-администратору необходимо устанавливать, настраивать, поддерживать, защищать и устранять неполадки веб-серверов. Веб-сервер является программным обеспечением, которое работает на одном или нескольких компьютерах. Веб-администратор может или не может быть ответственным за аппаратные средства. Как правило, веб-администратор не является

системным администратором на том или те компьютеры. Таким образом, веб-администратор должен будет работать с системным администратором по некоторым аспектам установки и конфигурации.
Конфигурация включает в себя создание конфигурационных файлов для указания директив, где веб-страницы будут расположены, которые могут получить доступ к их, есть ли какие-либо требования авторизации, какие формы безопасности могут быть реализованы, и какие документы будут закодированы, для примера, используя шифрование, а также то, что сценариев языки сервер может выполнить.
Мониторинг производительности веб-сервера позволяет администратору точно настроить параметры сервера. Это может включать в себя, например, установка большей мощности или несколько жестких дисков, добавив сжатие файлов, и уменьшения числа на стороне сервера сценариев. Кроме того, мониторинг с помощью автоматически генерируемых лог-файлов, предоставляет администратору запросов URL, которые могут быть данные, добытых обнаружить клиента поведение просмотра.
Обеспечение безопасности веб-сервера становится критическим для бизнеса. Веб-администратор может потребоваться создать файлы паролей и установить места, где серверные скрипты могут быть сохранены и проверены. Безопасность может быть реализована непосредственно с помощью программного обеспечения веб-сервера, или добавлена через операционную систему, или некоторую комбинацию. Вопросы безопасности удастся обнаружить, анализируя те же файлы журналов, которые используются для интеллектуального анализа данных.
Программное обеспечение веб-сервера состоит из различных меньших программ. Эти программы обрабатывают любое количество серверных задач, связанных с простого поиска файлов запросов URL для регистрации режимной обработки ошибок в URL перенаправлений. Есть бесплатные веб-серверы, такие как Apache, а также популярные коммерческие веб-серверы, такие как IBM WebSphere, Oracle WebLogic и Microsoft IIS. Веб-администратор может или не может участвовать в выборе программного обеспечения веб-сервера, но это работа веб-администратора, чтобы понять, программное обеспечение выбрать и подготовить рекомендации о том, как лучше поддерживать программное обеспечение соответствующим аппаратным системам. Аппаратные решения могут включать в себя, сервер распределения между физическими OUS многочисленными компьютерными серверами, сервер нескольких адресов, и следует использовать, какой-либо тип системы хранения. При мониторинге производительности веб-сервера,

веб-администратор может принимать обоснованные рекомендации по реконструкции и совершенствованию аппаратных средств.
Администраторы веб-сервера могут потребовать либо выполнение для разработки приложений для себя или для поддержки разработчиков. Поддержка может быть в виде установки дополнительных веб-серверов модулей, таких как те, которые могут выполнять Perl или PHP код. Там также могут быть политические решения, которые воздействуют на процесс развития, такие, как требование о том, что все сценарии выполнения вне веб-сервера на другом компьютере или все эти скрипты хранятся в CGI-BIN каталоге.
Хотя администрация веб-сервера может работать для крупной организации, которая хочет иметь присутствие в Интернете, есть также возможности работы для компаний, принимающих веб-сайтов. В таком случае, один сервер может быть использован для размещения нескольких веб-сайтов различных компаний. Администратор имеет дополнительные проблемы обеспечения (ы) производительность веб-сервера и работы с несколькими группами разработчиков веб-страниц.
Веб-мастеров зарплаты могут сильно варьироваться от $ 40000 до $ 90000 в среднем около $ 55000. Есть менее доступные для администраторов веб-сертификации. Первичная форма известна как Certified Administrator Internet Webmaster Server. Это один из многих под аббревиатурой CIW, но большая часть CIWS связана с веб-разработкой.
Администрирование баз данных
Как веб-администраторов и системных администраторов, администраторов баз данных имеют длинный список важных обязанностей, но с акцентом на проектировании, разработку и поддержку систем управления базами данных (СУБД). Задачи будут включать в себя установку, обслуживание, анализ производительности и устранения неполадок, как с другими административными районами. Существуют специальные приложения, параметры настройки производительности, создание учетных записей, и процедур, связанных с СУБД, которые отделены от тех, системных администраторов и администраторов сети. Администратор базы данных также может быть связан с интеграцией данных из старых систем к новым. Поскольку многие или большинство систем баз данных теперь доступны через Интернет (или, по крайней мере, в локальных вычислительных сетях), администратор должен также обеспечить надлежащие меры безопасности для удаленного доступа.
Есть много различных типов СУБД. Наиболее популярными являются реляционные базы данных, такие как Oracle, MySQL или Microsoft SQL

Server. Но другие типы технологии баз данных находятся на подъеме, включая базы данных NoSQL и баз данных, ориентированных на облачные вычисления. Как и во всех областях техники, продолжающийся рост новых технологий баз данных требует администраторов баз данных, чтобы быть в схеме постоянного обучения и самосовершенствования.
В отличие от веб-администраторов, которые, вероятно, не будут вовлечены в веб-разработку, администратор базы данных часто будет играть определенную роль в разработке приложений и развитий. Это может быть путем оказания помощи разработчикам приложений проектирования баз данных, необходимых для эффективной поддержки этих приложений. Для этого требуется уникальный набор навыков, которые включают в себя моделирование баз данных, разработка схемы, нормализации и настройки производительности. Из-за более специализированного характера администратора базы данных, зарплаты выше, чем в других ИТ-ролей, перечисленных выше. Срединные средняя заработная плата составляет около $ 70 000 с самым высоким уровнем зарплаты быть более $ 110 000. Сертификаты являются в первую очередь те из конкретного программного обеспечения СУБД, такие как Microsoft Certified администратор базы данных, Oracle Certified, и MySQL сертификации.

Специалист Компьютерной поддержки

Специалист Компьютерной поддержки является каким-то обобщающим термином. Конечно, некоторые из ролей специалиста поддержки может быть ответственность системного администратора. Специалист службы поддержки в первую очередь работает в качестве тренера по устранению неполадок. Учебные задачи могут включать производство материалов, таких как технических руководств, документации и учебных сценариев. Обучение может принимать многочисленные формы от группового обучения, индивидуального обучения, учебных видеофильмов, а также производства учебных видеофильмов и программного обеспечения.
r может включать в себя работу в справочную службу (смотрите ниже), обучение справочной службы военнослужащих, и контроль за справочной службы персонала. Он может также включать в себя помощь другому персоналу ИТ, таких как системный администратор (ы) и администратор (ы) сети. Задачи для специалиста компьютерной поддержки могут быть столь же обыденными, объяснить, как войти в систему или как запустить программу, отступая пользователя через какую-то задачу, такое как,

подключение к сетевому файловому серверу или принтеру, или же сложному открытию системы, почему организационные члены не могут получить доступ к ресурсу.
На начальном уровне, специалист компьютерной поддержки служит на службе. Он будет отвечать на телефонные звонки и электронные письма вопросов от пользователей. Типы вопросов, заданных справочной службы человека будет варьироваться в зависимости от организации, что человек работает. Например, если вы были на службу для поставщика услуг Интернета, вы можете принимать телефонные звонки от клиентов, которые не могут подключиться или медленное подключение к Интернету. В качестве справочной службы человека, вы должны быть в состоянии ответить на общие вопросы и знать, как найти ответы на менее распространенные вопросы или быть в состоянии направлять клиентов к другим, которые имеют более специализированные знания. Вы также услышите много вопросов, которые могли бы заставить Вас смеяться. Будьте готовы поговорить с некоторыми невежественными (технологически говоря) человек! Вот некоторые примеры (предположительно) истинных разговоров справочной службы. *
• "Какой компьютер у вас есть?" "белый ".
• Клиент: "Привет, это Роза. Я не могу вытащить мою дискету"
• Служба поддержки:" Вы пробовали нажимать кнопку
• "Клиент:" Да, конечно, она застряла"
• Служба поддержки: Это плохо; Я запишу"
• Клиент:". Нет. Подождите минуту. Я не вставила ее еще. Она, оказывается, все еще на моем столе. Простите."
• Служба поддержки: "Нажмите на иконку" Мой компьютер "на левой части экрана."
Клиент: "левый или слева от меня?"
• Служба поддержки: "Добрый день. Как я могу вам помочь "
Клиент:" Здравствуйте, я не могу напечатать "
Служба поддержки:" Вы бы могли нажать на старт "
Клиент:" Слушай приятель;?.. не кричите на меня! Я не Билл Гейтс, вы знаете! "
• Клиент: "Привет, добрый день, это Марта, я не могу печатать. Каждый раз, когда я пытаюсь, он говорит: "Не удается найти принтер. Я даже подняла принтер и поставила его перед монитором, но компьютер по-прежнему говорит, что он не может найти его."
* Взято с сайта http://www.funny2.com/computer.htm~~pobj.
• Служба поддержки: "Ваш пароль начинается с маленькой буквы" а ",

как в яблоке, прописной буквы« V », как и в Виктора, и число" 7 "." Клиент: "Является ли, '7' заглавными буквами"

• Клиент: "У меня есть огромная проблема. Друг поставил заставку на моем компьютере, но каждый раз, когда я двигаю мышь, она исчезает! "

• Служба поддержки: "Как я могу вам помочь?" Клиент: "Я пишу свою первую электронную почту." Служба поддержки: "Хорошо, и что за проблема?" Клиент: "Ну, у меня есть буква 'а 'в адресе, но как я могу получить круг вокруг него? "

С помощью удаленного доступа к рабочему столу, специалист компьютерной поддержки может "взять под контроль" компьютера пользователя, в удаленном режиме. Это позволяет специалисту лучше понять, диагностировать и устранять проблемы. Она также позволяет специалисту работать с небольшим входом от человека, который испытывает трудности. Это может сэкономить время, если человек не знает о компьютере или проблеме.

В отличие от административной карьеры, специалисты поддержки не могут иметь степень 4-ого года, ни специальных навыков, кроме тех, которые необходимы для работы в службу поддержки. Тем не менее, те, со степенью 4 года, скорее всего, получат акции и перейдут к более сложным, и, вероятно, более приятным, задачам. Зарплаты различаются, тем более, что специалисты поддержки могут работать неполный рабочий день или полный рабочий день. Диапазоны окладов могут быть как $ 25000 и выше, чем $ 70750, хотя специалисты поддержки чаще всего делают в $ 30000 до $ 45000 диапазона.

Управление ИТ

В дополнение к требованию сильные и часто специализированные технические навыки, все ИТ-карьеры, описанные здесь, требуют общения и совместной работы навыков. Эти мягкие навыки позволяют ИТ-специалистам, чтобы хорошо работать с конечными пользователями, разработчиками приложений, а также верхней Управляющей. Это одна из причин, почему работодатели сейчас ищут ИТ-персонал среди тех потенциальных сотрудников, которые имеют 4-летний ИТ-степень, которая обеспечивает всестороннее образование, помимо всего лишь технической подготовки.

Эти ИТ-специалисты, которые могут продемонстрировать межличностное общение, лидерство, и понимание бизнес-практики и может оказаться хорошей возможностью для будущих позиций управления ИТ. ИТ-менеджер это тот, кто понимает, как техническую сторону ИТ-инфраструктуры для того, чтобы понять проблемы, которые

необходимо устранить, а также сторону управления бизнесом в целях осуществления контроля технического персонала и легко общаться с высшим руководством. ИТ-менеджер будет менеджером проекта, который должен будет установить крайние сроки, работать с внешними составляющими, обрабатывать и, возможно, предложить бюджет, а также принимать решения в отношении проекта (ов), находящиеся под управлением. Кроме того, наем, и увольнение может быть частью обязанностей IT-менеджеров.

Позиции управления, как правило, с более высокой скоростью, чем зарплаты технических областях, описанных ранее. Если ваша цель в конечном счете, является получение такой позиции управления, вы бы лучше всего за счет увеличения ваших возможностей во время вашего обучения в колледже. Эти возможности включают в себя присоединение клубов и организаций колледжей, участвующих в управлении студента, а также принимая дополнительные занятия в бизнес-областях, таких как, управление проектами, экономика, бухгалтерский учет, и руководство. Всякий раз, студент, который ищет дальнейшего продвижения по службе, будет возвращаься в школу для продвинутой бизнес-степени, такие как MBA или MIS, или продвинутой технической степени, такие как степень магистра в CIT. ИТ-менеджеры могут потенциально сделать заработную плату в два раза больше технических работников.

Есть и другие карьеры, которые перекрывают содержание охвачены в 4-летнем IT степени. Список Следуя ИНГ обсуждаются некоторые из них.

Компьютерное преступление

Как правило, человек, который входит в карьеру компьютерной судебной экспертизы, будет иметь либо степени компьютерной криминалистики, или будет иметь высшее техническое образование с дополнительным фоном в уголовных дел. Техническое образование может быть информатикой, информационной технологией, или компьютерной инженерией. Компьютер судебно-медицинской экспертизы является новой и развивающейся областью, есть несколько программ бакалавриата, которые предлагают полный учебный план в этой теме. Тем не менее, многие компьютерные науки и ИТ-программы добавляют компьютерную криминалистику. Кроме того, выпускник программ в компьютерной криминалистике или родственный безопасности, находятся на подъеме.

Помимо технических навыков, полученных в области ИТ или научных программ ЭВМ, можно было бы необходимо практические знания в области операционных систем, компьютерной безопасности, компьютерной памяти, программные приложения, криптологии, и, возможно, оба языка программирования и программного обеспечения инженерные решения. Кроме того, сильные аналитические навыки являются существенными.
Человек, который работает в компьютерной криминалистики является не только обеспеченным компьютерной системой или ищет доказательства взломов и несанкционированного доступа. Вместо этого необходимо предоставить доказательства, в случаях компьютерных преступлений в правоохранительные органы, которые могли бы нанять вас. Таким образом, вы должны понять, как построить дело и как определить, собирать, хранить и представлять доказательства в судебном порядке.
Возможности трудоустройства в компьютерной судебной экспертизе имеются в большом количестве сегодня. Варианты включают работы непосредственно для сотрудников правоохранительных органов, консультации с правоохранительными органами, или работать для организаций, которые кровно заинтересованы в защите своих данных и преследования нарушителей, таких как банки и корпорации. Зарплаты существенно выше, в настоящее время, чем у других профессий ИТ, с годовыми окладами находится в диапазоне $ 75,000- $ 115,000 и в среднем на $ 85000. Тем не менее, без определенной степени в компьютерных судебно-медицинских экспертизах , ваши варианты могут быть более ограниченными, чем студенты, которые получили полную степень 4 года.

Веб-разработка

Есть много различных ролей, которые можно было бы играть в веб-разработке. Организация могла бы нанять одного человека, чтобы поднять свою компанию "вверх в Интернете" или компания может нанять штат разработчиков веб-страниц. Поэтому особую роль, которую веб-разработчик может играть может варьироваться в зависимости от компании и ситуации. Ясно, однако, заключается в том, что веб-разработчик имеет различные навыки, чем администратор веб-сайта, обсуждаемые в веб-администрирования.
навыки веб-разработчик включают в себя возможность использовать HTML, CSS (каскадные таблицы стилей) и различные языки сценариев.

Сценариев может быть сделано в JavaScript, Ruby, Python, Java-апплеты, Active Server Pages (ASP), Visual Studio .NET (включая Visual Basic, C ++ или C #), Perl и PHP. Несмотря на то, веб-разработчик не было бы ожидать, чтобы знать все эти языки, вполне вероятно, что веб-разработчик будет использовать некоторые из них. В то время как аспект программирования веб-разработчика важен, веб-разработчик не обязательно требует степень компьютерной науки (хотя мы склонны видеть много компьютерных ученых в качестве веб-разработчиков).

Веб-разработчику может понадобиться знание, как построить эстетическую веб-страницу с использованием правильного цвета, расположение, графики, анимации и так далее. Веб-разработчик также должен был бы понять многие технические и социальные вопросы, связанные с Интернетом. Например, понимание природы сети связи, пропускной способности, скорости кэширования диска, а также использование прокси-серверов поможет веб-разработчику создавать более эффективный и проще загружаемый веб-сайт. Понимание социальных проблем, может иметь решающее значение в обеспечении безопасности веб-сайта от атак, обеспечивая целостность данных веб-сайта, а также обеспечение того, чтобы содержание веб-сайта является политически корректным. Есть также ряд проблем доступности, так что людям, с различными типами недостатками можно еще посетить и маневрировать вокруг места без осложнений. Сейчас федеральный закон, который сайты федеральных агентств и ведомств, финансируемых правительством быть доступными для людей с гандикапом.

Сама Веб-разработка включает в себя несколько различных задач, которые включают в себя графический дизайн и искусство, бизнес, доступ к базам данных, телекоммуникации, взаимодействие человека с компьютером и программирование. В частности, разработка веб-приложений включает в себя написание на стороне сервера скрипты и другие серверные приложения в поддержку динамических веб-страниц и взаимодействия с сервером и базой данных. Позиции веб-разработчика пользуются большим спросом в эти дни. С другой стороны, у них есть огромный диапазон заработной платы, возможно, отражает широкий спектр компаний, которые хотят создать веб-портал для своих компаний. Зарплаты могут быть столь же низки как $ 30 000 и выше, чем в среднем $ 70000 диапазоне. Есть также целый ряд доступных сертификатов, чтобы улучшить свой статус. К ним относятся Microsoft Certified Professional Developer, несколько сертификатов CIW Web Developer (см "Web Administration"), HTML Developer сертификации и Sun Certified Web

Component Developer.

Программирование / инженер-программист

Исторически сложилось так, компьютерные программисты пришли из любого числа дисциплин, включая науки, математики и бизнеса. Степень информатики стала популярной, начиная примерно 1980. На сегодняшний день, компании не нанимают программистов, нанимают выпускников компьютерных наук программ (или в некоторых случаях, компьютерной техники). С увеличением акцента на программной инженерии, в некоторых школах теперь есть инженерные компьютерные программы, которые во многом похожи на компьютерные науки программ.

Компьютерная наука дисциплина обеспечивает другой тип фундамента, чем ИТ. Для компьютерного ученого, математическое обоснование необходимо изучить темы вычислимости и разрешимости. Эти темы дают ученым инструменты для измерения, как халькопирита лэнга данной проблемы является решение (или является ли она разрешимой вообще). Ученый также изучает организацию компьютера и операционной системы, чтобы понять основную природу компьютерной системы. Ученый также изучает широкий спектр алгоритмов и структур данных и реализует программы в ряде языков программирования.

Как и в случае ИТ-личности, инженер-программист / программного обеспечения, должен идти в ногу с полем знания. Тем не менее, в то время как ИТ-специалист будет в значительной степени изучать новейшие операционные системы, их особенности, инструменты и меры безопасности, компьютерное учение будет изучать новые языки и методы программирования. Есть сертификаты для программистов, хотя со степенью 4-летнего поля знаний, сертификаты могут быть не очень нужны. Зарплаты, как показано в таблице 16.1, как правило, выше в области компьютерной науки / программной инженерии, чем в других областях ИТ, за исключением ИТ-менеджмента.

Одной из наиболее сильнодействующей смесью курсовой, является объединение ИТ-степеней с научной компьютерной курсовой работой. Это обеспечивает ИТ человека с отличным фундаментом программирования. С другой стороны, ученый хорошо обслуживается принятия дополнительных ИТ-курсы, чтобы получить более четкое понимание операционных систем, сетей и компьютерной безопасности.

Информационные системы

Как описано в главе 15, IS существовали дольше, чем компьютеры. Информационной системой, по сути, является сбор данных и информации, используемых для поддержки управления организации. Чаще всего, организация является своего рода бизнесом, хотя это не должно быть так. Поле IS в некоторой степени связано с компьютерной наукой в том, что IS включает в себя изучение компьютерного оборудования, программного обеспечения и программирования. Тем не менее, акцент на программное обеспечение вращается вокруг программного обеспечения для бизнеса (например, СУБД, электронные таблицы, статистическое программное обеспечение, программное обеспечение для обработки транзакций), а также изучение аппаратных средств часто вращается вокруг сетей и в меньшей степени, хранения. Дисциплина охватывает целый ряд бизнес тематик, включая вводного бухгалтерского учета и экономики, и довольно часто включает в себя несколько курсов в области управления, а также мягкие навыки, которые менее подчеркиваются в информатике. Работник может взять на себя любое количество ролей в организации с целью проектного управления для разработки систем исследования. В качестве менеджера проекта, человек отвечает за планирование, проектирование, выполнение, мониторинг и контроль проекта. На этапах планирования и проектирования, менеджер должен будет производить спецификации, а также затраты времени и средств оценки. Менеджер может потребоваться, чтобы нанять сотрудников для поддержки проекта. Во время выполнения, мониторинга и контроля фаз, менеджер координирует с каждой группой реализации проекта, чтобы следить за прогрессом, определить проблемы, и решить проблемы путем получения дополнительных ресурсов. Несмотря на то, что диспетчер не должен обладать техническими навыками сотрудников, что он или она управляет, это очень важно, чтобы менеджер понять основные технологии.

В развитии есть сотрудник IS, который является частью команды внедрения. Реализация может потребовать программирование на языке высокого уровня (например, Visual Basic, COBOL или Java), но так же, как, вероятно, и более вероятно, будут использовать инструменты, которые обеспечивают Лицо с большей степенью абстракции над конкретными деталями компьютерного кода, Многие из этих инструментов упоминаются как 4-ое поколение языков (4GLs), подразумевая, что эти языки программирования, разработанные в ходе

четвертого поколения вычислительной техники. Эти языки включают генераторы отчетов, формировать генераторы и системы управления базами данных, такие как Excel, Access, SAS, SPSS, RPG, Mathematica и падежа инструменты, а совсем недавно XBase ++, XQuery и Oracle Reports. IS В обзоре рассматривается IS методы моделирования и IS инструменты. При моделировании, один пытается определить способ представления какого-то организационного поведения, будь то

1. Ты не должен использовать компьютер, чтобы причинить вред другим людям.

2. Ты не должен мешать работе компьютера других людей.

3. Ты не должен шпионить вокруг в компьютерных файлах других людей.

4. Ты не должен использовать компьютер, чтобы украсть.

5. Ты не должен использовать компьютер лжесвидетельствовать.

6. Ты не должен копировать или использовать собственное программное обеспечение, за которое вы не заплатили.

7. Ты не должен использовать другие ресурсы компьютера людей без разрешения или надлежащей компенсации.

8. Ты не должен соответствующий интеллектуальный выход других людей.

9. Ты будешь думать о социальных последствиях программы, которую вы пишете или системы, которую вы разрабатываете.

10. Ты будешь всегда использовать компьютеры таким образом, чтобы страховать внимание и уважение к ближним.

Рисунок 16.1
10 Заповедей компьютерной этики.

операций, используемых в организации IS или способа, в котором знание передается между организационными единицами или какими-либо процессами между ними. Учитывая модель, исследователь определяет методы Персонов сформировать какую-то задачу на модели. Модель и методы должны быть достаточно конкретными, так что исследователь способен моделировать процессы организации в достаточной степени, чтобы продемонстрировать, что модель и методы захвата ожидаемое поведение. Учитывая модель, исследователь может затем попытаться определить меры по совершенствованию процессов и политики организации.

Компьютерный специалист

Компьютерный техник часто кто-то, кто будет работать непосредственно на установке компьютерного оборудования, ремонта и устранения неполадок.
Инженеры (IEEE). Эти кодексы поведения в первую очередь связаны с программистами, и подчеркивают профессиональную компетентность, понимание законов, принимая профессиональный обзор своей работы, обеспечивая тщательную оценку рисков компьютерных систем, с учетом контрактов, стремясь улучшить свое понимание вычислительной техники и ее последствий, и т. д, Эти коды сочетают понятия этики, как они относятся к технологии, а также этика быть профессионалом. Есть целый ряд вопросов, которые любая организация должны решать в отношении надлежащей компьютерной этики, и, в частности, роль администраторов. К ним относятся понятия неприкосновенность частной жизни, право собственности, контроль, точность и безопасность.
• Защиту файлов, при каких обстоятельствах, следует рассматривать администратором других людей? При каких обстоятельствах компания должна использовать данные, накопленные о своих клиентах?
• Владение - кто владеет данными, что накапливаются и продукции, которая произведенная организацией?
• Контроль - до какой степени нужно контролировать действия сотрудника?
• Точность к кому же ответственность, точность падения конкретных сотрудников или всех сотрудников? В какой мере работа компании устраняет потенциальные ошибки в данных?
• безопасность, в какой-то степени, компания обеспечит целостность своих данных и систем?

В этом разделе мы сосредоточимся в первую очередь на этических дилеммах, которые могут возникнуть для ИТ-профессионала. После этого мы кратко рассмотрим некоторые вопросы политики, которые ИТ-специалисты могут предоставить в организации.
Для следующих ситуаций, представьте себя в данной ситуации. Что бы вы сделали?
• Ваш босс попросил вас контролировать сотрудников веб-серфингов, чтобы увидеть, если кто использует ресурсы компании, чтобы посмотреть или скачать порнографии. Этично ли для вас ЧАТИТЬСЯ с сотрудниками ?
• Ваш босс попросил вас изучить пространство файла конкретного сотрудника для незаконно загруженных предметов понижающего. Является ли это этично?
• Вы оказываетесь с некоторым временем простоя в работе. Вы решили освежить свои знания, посещая различные сайты связанных с использованием технологий. Среди них, вы читали о взломе компьютерных систем. Этично ли для вас, чтобы читать об этом? Этично ли для вас, чтобы читать о взломе в рабочее время?
• Вы заметили, что сервер электронной почты выполняется из дискового пространства. Вы берете на себя, чтобы искать во всех сотрудников по электронной почте, чтобы увидеть, если кто-то с помощью электронной почты компании для личного использования. Является ли это этичным ?
• установлен прокси-сервер для вашей организации. В первую очередь целью сервера является кэширование веб-страниц, которые другие пользователи могут захотеть посмотреть, тем самым экономя время. Тем не менее, прокси-сервер также создает журнал всех обращений. Этично ли для вас, чтобы просмотреть эти доступы, чтобы увидеть, какие сайты и страницы коллег были просмотрены?
• Вы веб-администратор (не веб-разработчик). Один из сайтов, что ваша компания хозяйничает, содержит контент, который вы лично оскорбительные. Что вы должны сделать? Если оно не является непристойным по стандартам сообщества, будет ли это изменить свое поведение? Что делать, если вы чувствуете, что содержание будет считаться непристойным вашим сообществом?
• Вы веб-разработчик. Вас попросили компанией, чтобы произвести некоторые веб-страницы с использованием контента, которые они предоставляют вам. Вы уверены, что содержание защищено авторским правом. Что вы должны сделать?
• Вы работаете в организации, которая готовит продукт для доставки.

Хотя вы и не являетесь частью команды внедрения, являясь ИТ человеком, вы знаете, что продукт еще не готов, но руководство заставляет их выпустить продукт в любом случае, ошибки и все. Что вы должны сделать?

• Ваш босс просит вас обновить операционную систему до более новой версии, которая известна как дыры в безопасности. Этично ли делать это?

• Вы боитесь, что ваш босс собирается уволить вас из-за ряда отсутствий (все из которых вы чувствуете, были оправданы). Вы сами решаете, в качестве меры предосторожности, чтобы создать секретный счет (бэкдор). Является ли это этично? Законно ли это?

• Ваша организация свободно позволяет людям обновлять свои страницы Facebook на сайте. Вы обеспокоены тем, что некоторые из ваших сотрудников выкладывают слишком много информации о себе. Это ваше место, чтобы принять участие?

• Ваша организация имеет политику, запрещающую людей от использования Facebook на работе. Вы заметили, один из ваших коллег изменил свою страницу Facebook в рабочее время. Если вы сообщить об этом?

Как видно из этих примеров ситуаций, вам может быть предложено поставить себя в положение, которое нарушает неприкосновенность частной жизни или доверие сотрудников. Когда вас попросили следить за сотрудниками, вас просят, чтобы Вы шпионили за ними. Для того чтобы быть этичным, необходимо учитывать множество факторов. Эти факторы включают в себя политику компании по использованию компьютеров сотрудников и сооружений, а также роль ИТ-специалистов. Без заявленной политики , трудно оправдать слежку за сотрудниками. С другой стороны, даже если компания имеет политику, насколько хорошо эта политика представлена сотрудникам? Если они не знают о политике, политика мало пользы.

В качестве ИТ-специалиста, ваша роль может включать в себя разработку и внедрение ИТ-политики (с другой стороны, политика может быть указана только руководством). Полисы должны отражать этическую позицию компании, а также закон. Политика должна охватывать все аспекты использования ИТ. В частности, политика должна определить:

• Роль, которую она играет в компании
• Правильное использование сотрудником ИТ
• права работника на личную жизнь (если таковые имеются)

• Правильное использование данных
• Безопасность и конфиденциальность данных
• Право собственности на идеи, разработанные с помощью ИТ в компании (интеллектуальная собственность)
• Как компания решает вопросы защиты авторских прав

Разработка политики может быть чуждым для ИТ-специалиста. Это может быть лучше всего заниматься управлением. Тем не менее, ИТ-специалист должен хорошо разбираться в политике организации для того, чтобы принимать решения, которые не нарушают политику. Не зная политики, ваши ответы на предыдущие вопросы может поставить вас на риск.
Подобно тому, как ИТ-специалист должен понимать как политику организации, а также надлежащую этическую позицию, понимание законов не менее важно. Рассмотрим ситуации, в которой ваш босс попросил вас установить программное обеспечение, которое он вручает вам на CD-R. Было ли это программное обеспечение, приобретенное на законных основаниях? Существуют ли лицензии с ним? Если нет, то вы идете вперед с установкой? Просто потому, что ваш работодатель просит вас выполнить задачу, не означает, что задача является либо этическим или юридическим. Зная закон, можно не только защитить вас, но и можно помочь вам сообщить своему работодателю о потенциально незаконных ситуациях. К сожалению, понимание законов, которые регулируют на информационные технологии и программное обеспечение может быть сложной задачей. Даже если вы можете определить законы, они часто пишутся неопределенно и в юридическом жаргоне.
Рассмотрим в качестве примера, что ваша компания создала политику, при которой все сотрудники будут известны их номера социального страхования. То есть, номер социального страхования будет служить в качестве уникального идентификатора для записей базы данных. Таким образом, он также используется в их журнале в процессе. Потому что это является частью их в журнале, номера социального страхования хранятся в мировых считываемых файлов в операционной системе (например, / и т.д. / пароль). В качестве системного администратора, что вы будете делать? Является ли использование номера социального страхования нарушением политики компании? Нет. Является ли это незаконным? Не строго говоря. Является ли это неэтично? Вероятно.
Вы убедите руководство, что с помощью номера социального обеспечения для журнала в целях является плохой идеей, и, следовательно, эти

защищенные данные удаляются из любого мирового считываемые файла. Тем не менее, эти значения все еще доступны в базах данных компании, о которых вы и база данных администраторов, и администраторы баз данных по-прежнему имеют доступ. Является ли это неэтично? Нет, потому что те, у кого есть доступ, имеют законную необходимость доступа к таким данным. Тем не менее, как может компания заверить своих сотрудников, что информация находится в безопасности? Люди, которые имеют доступ к защищенным данным, нуждаются в дополнительном обучении в понимании чувствительности таких данных. ИТ-специалисты, имеют дополнительную ответственность за защиту устройств, которые хранят эти конфиденциальные данные, такие как портативные компьютеры.

Вирус Flame и Stuxnet

Поскольку эта книга в настоящее время готовится к публикации, в новостях вышло о новом вирусе под названием Flame. Во-первых, некоторые фона. В 2009 году вирус Stuxnet (считается продуктом-израильского территории США, партнерства), атаковали различные иранские компьютеры, особенно тех, кто участвует в программе по обогащению урана в Иране. Stuxnet, и другой подобный вирус, Duqu, были небольшими вирусами, около 500 КБ. Оба Stuxnet и Duqu были обставленными как новаторской вредоносной программой на основе саботажа.
хотя Flame имеет аналогичные цели, компьютеры в Иране, а также в других странах Ближнего Востока и Восточной Европы, которые анализ показывает, что вирус был написан совершенно иной организации.
Пламя только появился на свет в 2012 году, но исследование вируса привело некоторых к мысли, что некоторые компьютеры были заражены, так как еще в декабре 2007 года.
Вирус Пламя около 20 МБ и намного большей сложности, чем Stuxnet. Его основная цель, кажется, чтобы следить за пользователями. Он может украсть данные (документы, файлы), запись разговорах с исподтишка включением микрофона компьютера, записывать нажатия клавиш, получить телефонные номера от соседних устройств Bluetooth с поддержкой, создавать и сохранять скриншоты, а также сканировать локальный интернет-трафика, чтобы получить имена пользователей и пароли так как они передаются по сети для аутентификации. Если этого недостаточно, вирус может также создать учетную запись бэкдор для

людей, так что они могут загружать больше компонентов вируса. Вирус запрограммирован, чтобы передавать накопленную информацию в один из более, чем 80 различных серверов в Интернете.
Сам Flame представляет собой набор единиц; начальные блоки, при заражении компьютера, используются для загрузки других единиц. Основной блок можно загрузить, извлекать, распаковывать и расшифровывать другие компоненты. Кроме того, вирус, кажется, способен прятаться от обнаружения.
Вирус Пламя может распространяться по сети и через USB-накопителей. Пламя было дано несколько названий различных организаций, включая Wiper, Viper и Flamer.

Другие социальные вопросы

ИТ-специалист должен понимать последствия ИТ-решений. Таким образом, за этики и права, помимо организационной политики, специалист по ИТ должен также понимать ключевые социальные проблемы. Некоторые из опасений, что наши лица общества описаны в данном разделе.
Цифровой разрыв. Есть пробелы в доступе к компьютерам и Интернету. Эти пробелы воздействия расы, пола, уровня доходов и географического местоположения. Гендерный разрыв значительно сократился в большинстве стран, особенно с ростом популярности социальных сетей. Но пробелы, которые происходят между технологически развитых и богатых стран над бедными странами продолжают расти. Поскольку Интернет предоставил тем, кто имеет доступ к огромному количеству информации, не имеющих доступа становятся еще более невыгодном положении по отношению к информированной жизни, здорового образа жизни, образования и занятости. Правительства и международные корпорации обратились в цифровых технологиях, но она по-прежнему является проблемой. Хотя, как ИТ-специалиста, вы можете никогда не иметь дело с последствиями цифрового разрыва, понимая это, вы могли бы помочь вашей организации создать разумные политики.
Компьютерные проблемы, связанные со здоровьем. Интенсивное использование компьютера может вызвать целый ряд проблем со здоровьем. Основными проблемами являются повторяющиеся травмы стресса (RSI) и напряжение глаз. Одной из наиболее распространенных форм RSI является кистевой туннельный синдром, мышечная проблема, которая воздействует на запястье. Это часто является причиной

обширных неправильного использования клавиатуры и мыши. Помимо RSI, части спины, боли в плечах и локтях, и даже бедра и деформации коленного сустава часто связывают с тяжелым весом компьютера. Перечень проблем, также включает в себя головную боль и стресс. Эти проблемы хорошо документированы, и существует целый ряд решений. Эргономика, изучение проектирования оборудования и устройств, которые лучше подходят человеческое тело, и HCI (взаимодействия человека с компьютером), дали нам лучшую мебель, компьютерные мониторы, которые легче на глаз, и менее деструктивные формы взаимодействия с компьютером (например, трек точка вместо мыши). Несмотря на эргономику , люди по-прежнему имеют проблемы со здоровьем из-за плохой осанки или просто потому что они забывают делать перерывы при использовании компьютера. Специалист ИТ должны понимать эргономические решения и использовать их при запросе вычислительных ресурсов.

Сохранение конфиденциальности. Угрозы личной жизни индивидуума продолжают расти. Существует уже множество частной информации о вас, которая обнародована. Это включает в себя такие факты, как ваша дата рождения, место жительства, вашего автомобиля (ей) вы едете, и когда и на ком вы женаты. Большая часть этой информации становится публично доступной, потому что, само по себе, он не представляет собой серьезную угрозу для вашей личной жизни или жизни. Тем не менее, с помощью социальных сетей, веб-сайты и блоги, люди, регулярно добровольчества больше информации.

Рассмотрим человека, который использует Tweeter, что он и его семья покидают город для недельной поездки. Умный вор мог найти дом человека, и, зная, что дом будет пустым в течение недели, перерыв его, и ограбить человека. Такой гласности собственной частной информации называется цифровая подстилка. Понимание угрозы такой саморекламы является важным для всех пользователей компьютеров в этом возрасте кражи личных данных. Информирование сотрудников угроз может упасть на ИТ-персонал. Тем не менее, эти угрозы могут быть гораздо более серьезными. Фишинг попытки используются мошенниками для получения защищенной информации от людей напрямую. Типичным примером является телефонный звонок (или по электронной почте) утверждают, что у авторизованного ИТ-лица, сообщая вам, что она потеряла соответствующие данные (например, пароль или номер социального страхования), и что вы должны предоставить абоненту с этой информацией или иначе ваш аккаунт будет отключен.

Интернет зависимость. Количество часов, которые многие люди теперь тратят с помощью компьютеров, превысило количество часов, которые они проводят с телевидением. Это замечательное изменение в нашем обществе. Одной из причин для этого сдвига в том, что люди становятся одержимы с электронными формами общения. Они включают в себя электронную почту, блоги и сайты социальных сетей. Это нужно всегда быть в сети уже окрестили интернет-зависимость, и некоторые люди страдают так сильно, что они не могут держать свои руки подальше от своих смартфонов, когда они находятся вдали от дома. Мы видим это в крайности, когда люди находятся в общественных мероприятиях (например, кино, на ужин) тратить больше времени на своих смартфонах, чем общаться с людьми лично. Эта конкретная социальная проблема не может напрямую повлиять на вашу организацию. Тем не менее, многие работодатели стали озабочены диверсий, которые влияют на их сотрудников. Интернет-зависимость может иметь ощутимое влияние на производительность труда.
Непристойность и цензура. Интернет открыл возможность для людей, чтобы читать и просматривать почти все на планете. К сожалению, не все представлено в Интернете будет проходить моральные компасы людей. Между порнографических сайтов, насильственному видео контента (видимая область на таких сайтах, как YouTube), и обличений вывешенные возмущенных лиц, есть материал в Интернете, что люди будут маркировать непристойным. Непристойность уже давно является проблемой в нашем обществе. Рейтинг услуг, Федеральная комиссия по связи (FCC), и сторожевые организации пытаются полиции кино, телевидения и радио. Тем не менее, полиции в Интернете это совершенно другое дело из-за его глобального характера и огромного содержания. Цензура может быть реализована в рамках организации через межсетевой экран и / или прокси-сервер, чтобы блокировать определенный контент или контент из конкретных источников. Как и многие из обсуждаемых вопросов выше, осведомленность

Цифровой мусор

Люди настолько привыкли размещать свои мысли на Facebook или через Twitter, или редактировать разные блоги и доски объявлений, что они беспечны информацией, которую они поставляют.
Рассмотрим следующее, казалось бы, невинное обновление "пойдем в

кино с FAM, сегодня домой около 3"
Представьте себе, что этот человек имеет "друга", который не стесняется воровать в небольших количествах, теперь друг знает, когда Вас уже не будет и как долго. Зачем рекламировать, что ваш дом будет пуст? и все же люди, регулярно (собирается в отпуск, взяв семью на обед, будет работать всю ночь, и т.д.).
Если вы не верите, то ваши невинные сообщения могут вовлечь вас в неприятности, сделайте небольшое исследование. Есть целый ряд веб-сайтов, созданных для получения информации о людях через социальные медиа и общественности. Сайт spokeo.com довольно тщательно это исследует. Проверьте это и введите свое имя. Вы можете обнаружить, что следующая информация была получена от Вас:
возраст, семейное положение, стоимость финансовых активов, политика и религия, интересы / хобби, род занятий, здоровье, имена членов вашей семьи, расположение вашего дома, стоимость вашего дома, снимки пола вашего дома.
Если бы это было не достаточно, чтобы напугать вас, считают, что существуют десятки сайтов, которые имеют сходную информацию о вас:

• mylife.com
• whozat.com
• socialpulse.com
• peekyou.com

перечисленные лишь некоторые из них!
и образование имеет важное значение. Политика в отношении надлежащего использования компьютерной техники, могут также предоставлять решения. Веб-разработчики и администраторы должны быть знакомы с вопросами, чтобы гарантировать, что их веб-сайт соответствует законам и правилам.

Непрерывное образование

Мы уже использовали термин «на протяжении всей жизни ученика." Что это значит? Фраза подразумевает, что вы будете продолжать учиться в течение всей жизни. Это очень важно для ИТ-специалистов, так как ИТ продолжает меняться. Напомним, различные истории, представленные в этом тексте компьютерных аппаратных средств, программного

обеспечения, операционных систем и языков программирования. Каждая область (компьютерная инженерия, программная инженерия, инженерные системы) потребуют, чтобы сотрудники, работающие в этой области, продолжали учиться. Без этого, так как новые технологии появляются, работник не в состоянии использовать их. Работник теперь держит компанию назад от совершенствования себя. Вполне вероятно, в таких условиях, что работник несёт ответственность и в конечном счете, увольняют. Таким образом, непрерывное образование в любой технологии ориентированные на поле деятельности и имеет решающее значение. Для ИТ-специалистов, непрерывное образование является не просто вопросом ценности для вашей организации, но и помощью вашей организации принимать обоснованные решения о будущих направлениях технологии.

Существуют многочисленные формы непрерывного образования, доступные в ИТ. Большинство из этих форм вращаются вокруг, и Вы их изучаете их по своему усмотрению. Есть много источников, доступных для обучения, но это чтобы взять на себя инициативу, чтобы найти и использовать эти источники самому. Среди форумов, доступных в торговых журналах, являются ли они частью популярной прессы, такие как Wired, New Scientist, журнал доктора Dobb, либо более ориентированных, таких как IEEE и ACM публикаций. Таблица 16.2 приведен список некоторых из наиболее актуальных публикаций ИТ из IEEE и ACM.[1]

Две проблемы с чтением, которые более ориентированны на журналы, заключаются в том , что они, как правило, подчёркивают передовые исследования, а не в доступных отраслей технологии, и они представлены на уровне, который мог бы быть более податливыми для аспирантов и преподавателей. Таким образом, как ИТ-специалисты, вы можете предпочесть вместо того, чтобы исследовать технологию посредством самых разнообразных книг и веб-сайтов.

Существует большой рынок технологических книг. На самом деле, почти каждая IT тема будет иметь несколько десятков книг, изданных от руководств пользователей к "фиктивных" направляющих книг стиля. Среди наиболее популярных ИТ-издателей - Томпсон Course Technologies, O'Reilly, Elsevier и Springer Verlag.

Многие технологии, ориентированные на сайты, создаются и поддерживаются ИТ-специалистами. Они публикуют свои знания в

1 Для получения дополнительной информации о IEEE см www.ieee.org, а также для получения дополнительной информации о ACM, посетите www.acm.org.

качестве сервиса для других людей, которые могут принести пользу, учась у них. Есть, например, большое количество сайтов, которые содержат информацию об операционной системе Linux, операционная система Windows 7, Интернет, веб-сервер Apache, и так далее. Эти сайты могут не только предоставить вам "как" знание, но и современные советы по известным проблемам и патчи безопасности.

Таблице 16.2 ACM и IEEE IT-Ориентированные Журналы

Журнал Имя	**IT- связанные темы**
ACM Сделки информации и системы безопасности	Криптография, механизмы авторизации, аудита, протоколы, угрозы, целостности данных, вопросы неприкосновенности частной жизни
ACM Сделки на интернет-технологии	Цифровые носители и цифровые права, электронная коммерция, Интернет преступления и закон, интернет производительность, одноранговые сети, безопасность и конфиденциальность
ACM Сделки по системам хранения	Системы хранения и архитектура, GRID области хранения, хранение сети, виртуализация, аварийное восстановление, кэширование
ACM Сделки по Веб	Интерфейс браузера, электронная коммерция, веб сервисы, XML, доступность, производительность, безопасность и конфиденциальность
Коммуникаций ACM	Коммуникации / сети, компьютерные системы, компьютеры и общество, системы хранения данных и поиска, производительности, персональные компьютеры, безопасность
IEEE Internet Computing	IT-услуги, технологии WWW
IEEE сети	Протоколы сети и архитектуры, проектирование протокола, программное обеспечение связи, сетевого управления и внедрения
IEEE технологии и общество	Воздействие на общество, история социальных аспектов, профессиональная обязанности
IEEE операции на компьютерах	Операционных систем, коммуника-ционные протоколы, производи-тельность, безопасность
IEEE операции по надежности и безопасная вычислительная среда	Оценка надежности и безопасности операционных систем и сетей, методы моделирования, безопасность, производительность
IEEE операции по беспроводной связи	Теоретические и практические темы в беспроводной связи в том числе систем-прототипов и новых приложений

Оценка надежности и безопасности операционных систем и сетей, методов моделирования, безопасности, производительности, теоретические и практические темы в беспроводной связи, включая системы прототипов и новых приложений ИТ-специалистов организации данных, кросс-функциональных систем, ИТ-прорывов, спекулируя на IT-достижений, новых технологий, электронной коммерции, групповой работы, широкополосные сети, средства безопасности.

Если у вас нет обучения, по своему усмотрению, руководствуясь формами, доступны продолжения по методу обучения. Они включают в себя курсы (будь то онлайн, с помощью технических школ, через вашу собственную организацию, или предлагаемые университетами), которые могут подготовить вас к квалификации или можете расширить или углубить свои знания. Многие компании будут поддерживать свои ИТ-специалистов, оплачивая различные формы непрерывного образования. Например, компания может заплатить за обучение в поиске либо ученую степень или сертификата. Разнообразие сертификатов, перечисленные в ИТ-Карьере. На сегодняшний день существует очень большое количество сертификатов связанных с ИТ. Когда вы проходите интервью для ИТ рабочих мест, вы можете спросить, считает ли организация ценными сертификатами их.

Есть также торговые конференции, происходящие во всем мире, несколько раз в год. Эти конференции включают авторами Adobe, Oracle, VMware, Microsoft и IBM. Другие охватывают такие темы, как облачные вычисления, веб-сервисы, Web 2.0, баз данных, сетевых связей, виртуализации, функциональной совместимости, новых технологий, криптографии и ИТ наилучшей практики, чтобы назвать только некоторые из них. Как и курсы повышения квалификации, он является общим для компаний, чтобы платить за своих ИТ-специалистов для участия в таких конференциях.

По сути дела, чтобы продолжить ИТ-образование, вы должны быть готовы учиться. Надеюсь, вы уже само обучаетесь и продолжаете свое образование, независимо от того, ваш работодатель находит его ценным или потому, что вам интересно, и что вы будете этим наслаждаться.