Магда Ю. С.

Raspberry Pi.

Руководство по настройке и применению

ДМК Пресс
издательство
Москва, 2014

УДК 004.42:004.3'144:621.3.049.774ARM
ББК 32.973.26-018.2
М12

Магда Ю. С.
М12 Raspberry Pi. Руководство по настройке и применению – М.: ДМК Пресс, 2014. – 188 с.

ISBN 978-5-94074-964-6

Быстрый прогресс современной электроники в последние годы существенно повлиял на все сферы человеческой деятельности, включая применение компьютерных технологий. Существенным прорывом стало создание полнофункциональных компьютерных систем на одном кристалле, так называемом System-On-Chip (SoC). В SoC интегрируются все основные функциональные блоки, присущие компьютерам (процессор, память, графический процессор и др.). На одном из таких SoC-кристаллов реализован один из наиболее популярных современных миниатюрных компьютеров, известный под названием Raspberry Pi.

Эта книга посвящена практическим аспектам применения Raspberry Pi, начиная от программирования простых систем управления и измерения на языке Python и заканчивая разработкой мультимедийных систем и созданием игровых приложений на языке Scratch. Хотя Raspberry Pi помещается на ладони, он способен выполнять многие функции, доступные мощным настольным системам. Многие популярные приложения, работающие на настольных компьютерах, могут выполняться и на Raspberry Pi. Вдобавок Raspberry Pi обладает мощными мультимедийными и графическими возможностями, в частности, при работе с 3D графикой, поэтому этот миниатюрный компьютер можно использовать как платформу для разработки игровых приложений, что может заинтересовать многих будущих программистов. Raspberry Pi можно использовать и для создания своих собственных измерительных и робототехнических систем с различными датчиками и исполнительными устройствами. Создание таких систем возможно благодаря наличию цифрового порта ввода/вывода (GPIO) – подобная возможность отсутствует в обычных настольных ПК.

Материал книги будет полезен самой широкой аудитории, начиная от школьников и студентов и заканчивая разработчиками приложений для мультимедиа, Интернета и систем управления.

УДК 004.42:004.3'144:621.3.049.774ARM
ББК 32.973.26-018.2

ISBN 978-5-94074-964-6

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Материал этой книги посвящен практическим аспектам применения миниатюрного компьютера Raspberry Pi, начиная от программирования простых систем управления и измерения на языке Python и заканчивая разработкой мультимедийных систем или созданием игровых приложений на языке Scratch.

Несмотря на то что Raspberry Pi – это всего лишь миниатюрный компьютер, помещающийся на ладони, он позволяет выполнять многие функции, доступные мощным настольным системам. Большинство приложений, работающих на настольных компьютерах, могут выполняться и на Raspberry Pi. Вдобавок Raspberry Pi обладает мощными мультимедийными и графическими возможностями, в частности при работе с 3D-графикой, поэтому этот миниатюрный компьютер можно использовать как платформу для разработки игровых приложений, что может заинтересовать многих будущих программистов.

С другой стороны, Raspberry Pi можно использовать для создания своих собственных измерительных и робототехнических систем с подключением различных датчиков и исполнительных устройств, таких, например, как электромагнитные реле и двигатели, к цифровым портам ввода/вывода (GPIO, General Purpose Input/Output). Наличие GPIO является существенным преимуществом Raspberry Pi по сравнению с настольными компьютерами, в которых подобные возможности отсутствуют.

Большинство обычных настольных компьютеров работают с популярными операционными системами Windows или Linux, которые обеспечивают доступ к аппаратным средствам и дают возможность использовать различные популярные приложения. Для Raspberry Pi были разработаны несколько вариантов операционных систем, наиболее популярной из которых является Raspbian OS. Raspbian OS была разработана специально для Raspberry Pi и является одной из модификаций Debian – одного из наиболее распространенных дистрибутивов Linux.

Аппаратная часть платы Raspberry Pi содержит центральный процессор, графический контроллер, оперативную память (RAM), а также различные интерфейсы для подключения внешних устройств.

Ниже (рис. 1) показан внешний вид модуля Raspberry Pi последней модификации (Model B Rev.2). Материал этой книги будет базироваться на данной модификации Raspberry Pi.

Для загрузки операционной системы Raspbian OS к модулю Raspberry Pi необходимо подключить какое-либо устройство постоян-

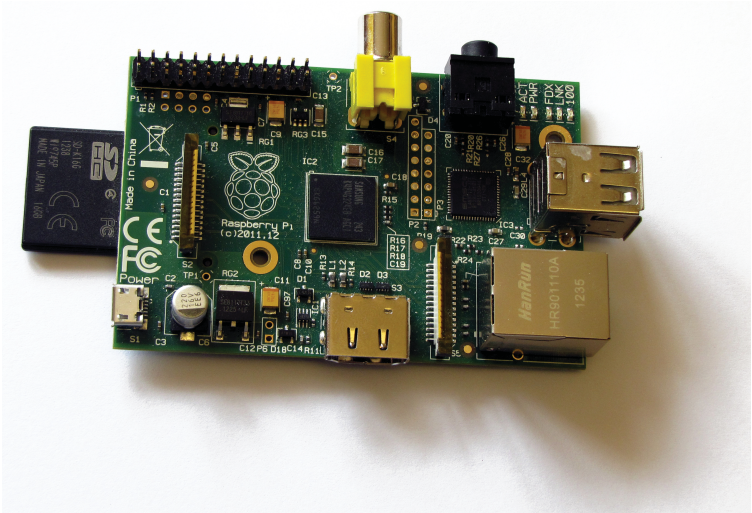

Рис. 1

ной памяти большой емкости («mass-storage device»). Для обычных настольных ПК таким устройством является жесткий диск. Для Raspberry Pi используется SD-карта памяти, содержащая Raspbian OS; пользователь может либо инсталлировать операционную систему самостоятельно, либо приобрести SD-карту с предустановленным загрузчиком Raspbian OS. Последний вариант более предпочтителен для новичков, не имеющих опыта настройки и конфигурирования операционных систем Linux. На рис. 1 показана плата Raspberry Pi с установленной SD-картой, на которую записана операционная система Raspbian OS.

Для функционирования модуля Raspberry Pi в качестве полнофункционального компьютера необходимо подключить клавиатуру и мышь к разъемам USB-платы, а также монитор с HDMI-интерфейсом (в качестве альтернативы можно использовать телевизионный приемник с композитным входом, но качество изображения будет хуже). К сожалению, в Raspberry Pi не предусмотрена работа с VGA-интерфейсом, хотя для подключения VGA-монитора можно воспользоваться адаптером VGA-HDMI. Важное замечание: такой адаптер должен запитываться от отдельного источника питания, чтобы избежать повреждения платы Raspberry Pi.

Модуль Raspberry Pi получает питание от источника напряжения +5 В с максимально допустимым током в нагрузке не менее 700 мА. Источник питания подключается к модулю посредством кабеля с микро-USB-разъемом. Необходимо учитывать то, что при недостаточной мощности источника питания модуль Raspberry Pi будет работать неустойчиво.

На плате также имеется разъем для подключения Raspberry Pi к сети Ethernet, что позволяет осуществлять доступ к ресурсам локальной сети и/или к Интернету.

Модуль Raspberry Pi, кроме функций, свойственных персональным компьютерам, может служить основой для разработки систем управления и измерения. Для этого на плате выведены сигнальные линии цифровых портов ввода/вывода (GPIO) на разъем P1. К этим линиям можно подключать внешние электронные цепи, управляемые программным способом.

В главе 1 мы рассмотрим более детально конфигурирование Raspberry Pi.

1 Сборка и запуск Raspberry Pi

В этой главе мы рассмотрим последовательность сборки и настройки аппаратной части модуля Raspberry Pi.

На первом шаге установим SD-карту с образом операционной системы в модуль Raspberry Pi. Затем присоединим клавиатуру и мышь к USB-разъемам модуля. Если в дальнейшем планируется использовать какое-либо дополнительное оборудование с USB-интерфейсом (например, Bluetooth, Wi-Fi-адаптер, внешний накопитель USB и т. д.), то можно расширить количество USB-портов системы, подключив расширитель USB к одному из свободных USB-слотов.

Для подключения Raspberry Pi к дисплею можно воспользоваться одним из вариантов, предлагаемых разработчиками системы. Если у пользователя имеется дисплей, позволяющий обрабатывать видеосигнал в формате HDMI (High Definition), то видеовыход HDMI на плате Raspberry Pi соединяется с видеовходом дисплея.

Многие современные телевизионные приемники и дисплеи работают с DVI-сигналами, поэтому для их подключения можно использовать недорогой DVI-адаптер и HDMI-кабель (модификаций 1.3–1.4). Модуль Raspberry Pi поддерживает выходные аудио/видеосигналы в формате HDMI, хотя входные сигналы в данном формате не поддерживаются.

Если у вас нет дисплея с HDMI-интерфейсом, то можно воспользоваться обычным аналоговым телевизионным приемником, соединив его с композитным выходом Raspberry Pi посредством стандартного RCA-кабеля желтого цвета. При использовании такого соединения выходной аудиосигнал для последующей обработки можно снимать с 3,5-миллиметрового джека на плате Raspberry Pi.

Многие мониторы все еще используют стандарт VGA для обработки видеосигналов. Хотя Raspberry Pi и не работает с VGA, тем не менее можно приобрести адаптер VGA-HDMI и соединить HDMI-выход платы Raspberry Pi с VGA-входом монитора. Один из таких адаптеров показан на рис. 1.1.

Рис. 1.1

Ниже приводится последовательность операций при подключении Raspberry Pi:

1. Установить SD-карту с загрузчиком в Raspberry Pi.
2. Подсоединить USB-клавиатуру и мышь к плате. Большинство из имеющихся в продаже клавиатур/мышей будут работать с Raspberry Pi.
3. Подсоединить один конец видеокабеля к Raspberry Pi (HDMI или композитный AV), а второй – к устройству отображения (телевизионный приемник либо монитор).
4. Присоединить дополнительные устройства к Raspberry Pi (USB Wi-Fi, кабель Ethernet и т. п.).
5. Подсоединить источник питания к разъему микро-USB на плате.

Если все действия выполнены правильно, система начнет загружаться. Следует помнить, что при первом включении процесс загрузки выполняется медленнее, поскольку происходит автоконфигурирование системы.

Несколько слов об особенностях подключения и настройки аппаратных интерфейсов.

Поскольку Raspberry Pi не имеет внутреннего устройства массовой памяти и не поставляется с предварительно записанным образом операционной системы, то для работы потребуется SD-карта с инсталлированной операционной системой. Существует несколько дистрибутивов операционной системы Linux, разработанных для Raspberry Pi. Материал книги базируется на Raspbian OS, которая является одной из модификаций популярной версии Debian Linux.

Образ операционной системы можно записать на SD-карту самому либо воспользоваться уже подготовленной для загрузки SD-картой, которая продается в многочисленных интернет-магазинах. Для начинающих пользователей лучше всего воспользоваться SD-картой с предустановленной операционной системой Raspbian OS.

При желании пользователь может сам создать дистрибутив операционной системы на SD-карте, воспользовавшись инструкциями из Интернета. Существует несколько различных вариантов для инсталляции операционной системы на SD-карту в зависимости от имеющихся у пользователя аппаратно-программных ресурсов; подробная информация по этой теме имеется на многочисленных форумах, посвященных Raspberry Pi.

В качестве периферийных устройств ввода информации подойдут любые стандартные клавиатура и мышь с USB-интерфейсом. Бес-

проводные мышь и клавиатура также будут работать, но в этом случае понадобится дополнительный порт USB, поскольку необходимо будет подключить Bluetooth dongle для коммуникации с периферией. Поскольку модуль Raspberry Pi Model B имеет только два USB-порта, то в этом случае понадобится подключить расширитель USB (USB-хаб). Более подробную информацию о подключении беспроводной периферии к плате Raspberry Pi можно найти в Интернете.

Подача питания на плату Raspberry Pi осуществляется через микро-USB-разъем. Для подачи питания в разъеме задействованы только выводы питания, поэтому через этот микро-USB-соединитель нельзя передавать данные.

В качестве блока питания подойдет стандартное зарядное устройство для мобильных телефонов, которое может обеспечить ток в нагрузке по меньшей мере 700 мА при постоянном напряжении на выходе +5 В. Желательно проверить параметры блока питания перед его подключением к плате RPi. Если блок питания не может обеспечить достаточный ток в нагрузке, то вполне возможно, что модуль Raspberry Pi запустится, но его работа может оказаться неустойчивой.

Вполне вероятно, что пользователь захочет подключить больше устройств к плате RPi (флэш-диски, видеокамеры, акустические системы, Bluetooth-устройства и т. д.). В этом случае лучше всего воспользоваться расширителем USB (USB-хабом) с отдельным источником питания, поскольку это снизит нагрузку на цепи питания платы RPi и позволит избежать нежелательной перегрузки блока питания RPi. Желательно использовать USB-хаб, поддерживающий протокол USB 2.0. USB 1.1 подойдет для клавиатуры и мыши, но для более скоростных устройств этого может оказаться недостаточно.

Для подключения к сети на плате имеется порт Ethernet со стандартным разъемом RJ-45. В Raspberry Pi Model B порт Ethernet может настраиваться автоматически при прямом подключении к маршрутизатору или другому компьютеру, поэтому нет необходимости в переходном (crossover) кабеле.

2 Установка и загрузка Raspbian OS

В этой главе будут описаны основные этапы установки операционной системы Raspbian OS на модуль Raspberry Pi. Начнем с конфигурирования SD-карты, на которую записывается образ операционной системы.

Многие производители SD-карт продают свои карты с уже предустановленной операционной системой. Во многих случаях, особенно для начинающих пользователей, такой вариант является наиболее предпочтительным. Тем не менее можно взять чистую SD-карту и установить операционную систему самостоятельно. Для установки загрузочного образа Raspbian OS операционной системы из Windows нужно выполнить несколько простых шагов:

1. Загрузить Raspbian OS, используя ссылку down-loads page at raspberry pi.org. Операционная система Raspbian распространяется в формате образа диска, который представлен побитово, так как он должен быть записан на SD-карту.
2. Загрузить программу Win32DiskImage.
3. Вставить SD-карту в картрайтер, запомнив букву диска, которая присвоена этому ресурсу в Windows Explorer.
4. Запустить Win32DiskImage и выбрать опцию **Raspbian disk image**.
5. Выбрать букву диска, соответствующую SD-карте, затем щелкнуть на этой иконке мышью, после чего выбрать опцию **Write**. Если в процессе записи образа диска возникают проблемы, то можно еще раз переформатировать SD-карту в Windows Explorer и записать образ диска повторно.
6. Извлечь SD-карту из картрайтера и установить ее в плату Raspberry Pi.

Необходимо учитывать, что нельзя просто перетянуть образ диска на SD-карту Windows Explorer, – следует сделать побитовую копию образа диска с картрайтера.

Для записи двоичного образа Raspbian OS в операционной системе Linux нужно выполнить такую последовательность шагов:

1. Открыть терминальную программу (LXTerminal, например) и проверить, какие дисковые тома смонтированы в системе, набрав команду

   ```
   pi@raspberrypi ~ $ df - h
   ```

2. SD-карта не должна быть подключена.
3. Вставить SD-карту в картридер и еще раз выполнить команду **df –h**.

Из списка смонтированных томов определить том, размещенный на SD-карте. Нам понадобится имя устройства, которое будет выглядеть примерно как /dev/sdd1. Следует учитывать то, что в зависимости от конфигурации системы имя устройства может отличаться от приведенного выше.

4. Для записи образа на SD-карту вначале нужно демонтировать устройство командой umount:

```
pi@raspberrypi ~ $ umount /dev/sdd1
```

Если файловую систему не удается размонтировать, то есть выдается ошибка, то необходимо проверить, нет ли открытых каталогов и/или файлов, находящихся на этом томе.

5. На следующем шаге следует определить символьное имя устройства SD-карты, которое в общем случае будет таким же, как и имя блочного устройства, только без номера раздела. Например, если имя блочного устройства /dev/sdd1, то имя символьного устройства будет /dev/sdd.
6. Необходимо разархивировать загруженный образ диска и поместить его в домашний каталог. Для записи образа операционной системы на SD-карту нужно воспользоваться командой **dd**:

```
sudo dd bs=1M if=~/2012-09-18-wheezy-raspbian.img of=/
dev/sdd
```

Эта команда выполняется от имени суперпользователя root и производит копирование файла-источника (if) в файл-приемник (of).

7. После успешного создания образа операционной системы нужно перегрузить Raspberry Pi.

При первой загрузке параметров Raspbian OS будет выведено окно конфигурации (программа **raspi-config**), в котором пользователь сможет выполнить общие настройки системы (рис. 2.1).

Программу конфигурации можно запустить в любой момент на работающей системе, выполнив команду

```
sudo raspi-config
```

Ниже приводятся основные опции утилиты **raspi-config** и их расшифровка.

Expand rootfs

Пользователь всегда должен выбирать эту опцию, для того чтобы получить доступ к SD-карте.

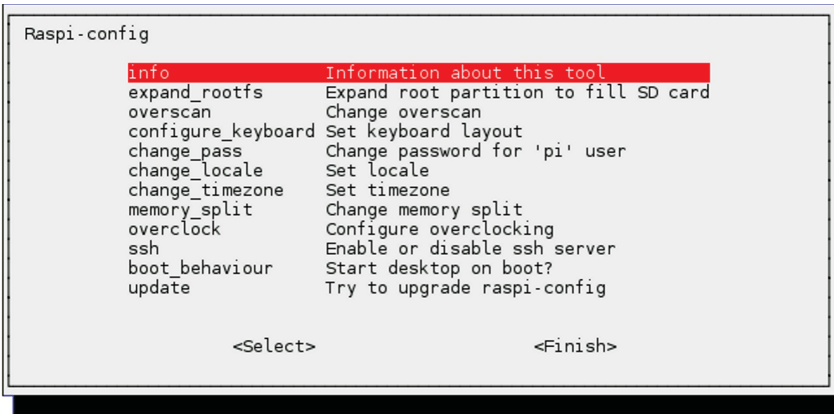

Рис. 2.1

Overscan

При первом запуске рекомендуется оставить данную опцию отключенной (disabled). В некоторых случаях для мониторов с высоким разрешением текст может выходить за границы экрана, тогда нужно включить эту опцию (enabled) и подобрать параметры так, чтобы изображение полностью находилось в пределах экрана монитора.

Keyboard

Настройки клавиатуры по умолчанию соответствуют UK-раскладке, хотя при необходимости можно установить собственную раскладку.

Password

С помощью этой опции можно поменять пароль по умолчанию (raspberry) на более секретный.

Change Locale

Эта опция позволяет установить параметры локализации и языковые настройки, включая таблицы перекодировки (по умолчанию установлен UK English по стандарту UTF-8).

Change timezone

Используя эту опцию, вы сможете выбрать параметры часового пояса.

Memory split

С помощью этой опции можно изменить объем памяти, используемой процессором и графической подсистемой. В настоящее время не рекомендуется менять настройки по умолчанию.

Overclock

С помощью этой опции можно установить тактовую частоту синхронизации процессора выше стандартной, равной 700 МГц. При первой загрузке рекомендуется оставить настройки по умолчанию или попробовать опции **Medium** или **Modest**. В дальнейшем можно поэкспериментировать, установив режим **Turbo**, позволяющий поднять тактовую частоту процессора до 1000 МГц.

SSH

Эта опция позволяет включить SSH-сервер, что даст возможность выполнить безопасное подключение к Raspberry Pi по сети. Желательно включить эту опцию.

Desktop Behavior

С помощью этой опции можно включить/отключить графический интерфейс пользователя. По умолчанию этой опции присвоено значение **Yes**, поэтому графическая оболочка загружается в процессе загрузки операционной системы. Если установить значение **No**, то по окончании процесса загрузки будет запущена текстовая консоль. Запустить графическую оболочку можно после входа в систему следующей командой:

```
pi@raspberrypi ~ $ startx
```

При работе с графическим интерфейсом командная строка будет доступна из терминальной программы, такой, например, как LXTerminal.

Update

Если компьютер подключен к Интернету, то эта опция позволяет обновить утилиту конфигурации.

После выполнения необходимых настроек компьютер следует перегрузить, для чего нужно выбрать опцию **Finish**. После переключения системы в режим командной строки нужно набрать команду:

```
pi@raspberrypi ~ $ sudo reboot
```

Для выключения Raspberry Pi следует выполнить команду **shutdown**. Для этого можно воспользоваться опцией **Shutdown** в меню **Logout**. Команду **shutdown** можно выполнить и из командной строки:

```
pi@raspberrypi ~ $ sudo shutdown -h now
```

3 Linux и Raspberry Pi

Функционирование программного обеспечения в Raspberry Pi выполняется в среде Raspbian OS, которая является адаптированной версией Debian Linux. В этой главе дается обзор основных возможностей Raspbian OS, а также наиболее популярных и полезных программ, выполняющихся в данной среде.

Операционная система Linux базируется на концепции открытого кода (open source), что позволяет пользователям свободно использовать и/или модифицировать ядро системы. Ядро Linux является сердцем операционной системы, осуществляя взаимодействие пользователя с аппаратными и программными ресурсами системы. Изначально под термином Linux подразумевалось именно ядро, хотя в настоящее время это определение относится ко всем приложениям (коллекциям или наборам) с открытым кодом, включенным в состав операционной системы.

Каждая такая коллекция программ может представлять собой отдельную реализацию (дистрибутив) Linux. В Raspberry Pi используется модифицированная версия дистрибутива Debian Linux, Raspbian OS.

Подобно другим операционным системам, таким как Windows или MacOS в Linux, реализован доступ нескольких к ресурсам системы, то есть Linux является многопользовательской системой с возможностью доступа к аппаратно-программным ресурсам для нескольких пользователей. При этом каждый пользователь системы имеет собственную учетную запись (account), для которой определены права доступа (привилегии) к тем или иным ресурсам Linux. Такой подход позволяет обеспечить эффективную защиту операционной системы от случайных или умышленных действий пользователя.

Наивысшие привилегии доступа к ресурсам операционной системы имеет суперпользователь (root), который может выполнять практически все операции по управлению аппаратными и программными ресурсами системы без каких-либо ограничений. Суперпользователь имеет практически неограниченный доступ к файловой системе, физическим и логическим (виртуальным) устройствам системы, поэтому работа в режиме суперпользователя *root* требует от пользователя высокой квалификации и осторожности, поскольку легко можно разрушить саму операционную систему. Режим суперпользователя *root* обычно применяется опытными пользователями, как правило, системными администраторами, для настройки и конфигурирования аппаратных и программных ресурсов Linux.

Режим суперпользователя не следует использовать в повседневной работе, поскольку всегда можно где-то ошибиться и нарушить работу операционной системы. Для выполнения обычных программ вполне достаточно привилегий обычного пользователя. В Raspbian OS таким пользователем является *pi*, имеющий по умолчанию пароль для входа *raspbian*. Этот пароль можно в любой момент изменить, если требуется большая безопасность системы. Пользователь *pi* имеет достаточно широкие полномочия по управлению системой, инсталляции и выполнению приложений, поэтому в большинстве случаев привилегий суперпользователя *root* не требуется.

В дальнейшем мы рассмотрим возможные случаи, когда без использования доступа посредством *root* нельзя обойтись.

Ниже (табл. 3.1) приводятся наиболее часто употребляемые термины при работе в операционной системе Linux, которые будут нередко использоваться далее в этой книге.

Таблица 3.1. Краткий перечень терминов и определений операционной системы Linux

Термин	Определение
Bash	Наиболее популярная командная оболочка, используемая в большинстве дистрибутивов Linux
Bootloader	Программное обеспечение, выполняющее загрузку ядра операционной системы Linux. В настоящее время наиболее популярным загрузчиком является GRUB
Console	Приложение командной строки для выполнения команд и приложений Linux
Desktop environment	Программное обеспечение для реализации графического интерфейса пользователя (GUI). К числу наиболее популярных графических оболочек относятся GNOME и KDE
Directory	Каталог (или, по-другому, папка) – общее название места, где хранятся файлы
Distribution	Этим термином обозначают конкретную версию (дистрибутив) операционной системы Linux, например Fedora Remix, Arch и Debian являются дистрибутивами Linux
Executable	Файл, который может быть выполнен как программа. Для запуска программы файл должен иметь атрибут «executable»
EXT2/3/4	Наиболее часто используемые в Linux форматы файловых систем
File system	Способ организации файлов и каталогов в операционной системе Linux
GNOME	Одна из наиболее популярных графических оболочек Linux
GNU	Проект для реализации свободно распространяемого программного обеспечения. Многие программы и утилиты, разработанные под эгидой этого проекта, используются в различных дистрибутивах Linux

Таблица 3.1 (окончание)

Термин	Определение
GRUB	Загрузчик ядра операционной системы Linux (GRand Unified Bootloader), разработанный в рамках проекта GNU
GUI	Графический интерфейс пользователя, в котором пользователь может управлять приложениями с помощью мыши
KDE	Еще одна популярная графическая оболочка
Linux	Ядро Linux, разработанное в рамках проекта GNU, или, по-другому, операционная система с открытым кодом
Live CD	Дистрибутив Linux, записанный на CD или DVD и не требующий инсталляции на жесткий диск
Package	Пакет (группа) взаимосвязанных файлов, требуемых для выполнения приложения. Пакеты программ обрабатываются менеджером пакетов
Package manager	Приложение для установки и настройки программного обеспечения, реализуемого в форме пакетов прикладных программ
Partition	Область жесткого диска, подготовленная для размещения файловой системы
root	Наиболее привилегированный пользователь Linux
Superuser	То же, что и root
Shell	Командная оболочка терминального приложения для выполнения команд Linux в текстовом режиме
Terminal	Приложение текстовой строки, позволяющее выполнять команды Linux, используя командную оболочку
X11	Графическая система X Window, позволяющая реализовать графический интерфейс пользователя

Для эффективного использования Raspberry Pi в качестве настольного ПК пользователь обязательно должен располагать базовыми знаниями операционной системы Linux, в частности версии Debian Linux, используемой в качестве базовой в Raspberry Pi. Все без исключения версии Linux обладают базовым набором свойств и характеристик, поэтому следующие разделы главы посвящены основам работы в операционной системе Linux, после чего мы ознакомимся с особенностями функционирования Debian Linux.

Основы функционирования операционной системы Linux

Операционная система Linux по своей сути является потомком классической и популярной, начиная с 70-х годов прошлого века, операционной системы UNIX и наследует ее лучшие характеристики.

С момента своего появления операционная система UNIX получила широкое распространение в вычислительных системах различной производительности, начиная от персональных компьютеров и заканчивая большими вычислительными комплексами. Такая популярность этой операционной системы обусловлена несколькими факторами.

Во-первых, это более чем трехдесятилетний цикл развития. За этот период операционная система UNIX выдержала проверку временем и проявила себя как очень эффективная программная среда. Во-вторых, программный код системы полностью написан на языке высокого уровня C, что делает ее понятной для пользователей и разработчиков программного обеспечения. Это позволяет относительно легко вносить изменения как в существующие версии UNIX, так и переносить операционную систему на другие аппаратные платформы.

Кроме того, во многих случаях версии этой операционной системы поставляются вместе с исходными текстами, что позволяет легко адаптировать UNIX под специфические требования, после чего перекомпилировать систему.

Изначально операционная система UNIX создавалась как многопользовательская и многозадачная система, ориентированная в первую очередь на выполнение серверных или управляющих функций для клиентских компьютеров. Такие подходы, а также мощные встроенные средства удаленного администрирования, способствовали тому, что UNIX заняла лидирующие позиции на рынке интернет-серверов и серверов баз данных. Немаловажную роль в популяризации операционной системы сыграла и структура ее файловой системы, представляющая единую иерархическую систему с унифицированным доступом как к файлам данных, так и к аппаратным ресурсам, таким как диски, терминалы, принтеры, сеть, память и т. п.

В практическом плане операционная система UNIX предоставляет пользователю целый ряд преимуществ, некоторые из них перечислены ниже:

- все популярные приложения (пакеты офисных программ, базы данных и др.) известных производителей программного обеспечения, как правило, реализованы для работы в операционной системе UNIX;
- UNIX поддерживает широкий спектр средств передачи информации, включая многочисленные сетевые и коммуникационные протоколы, что обеспечивает эффективную работу операционной системы в сетях;

- операционная система UNIX имеет весьма развитые средства системного и сетевого управления, в том числе эффективные инструменты для удаленного администрирования и настройки;
- UNIX является мощной платформой для разработки приложений, в нее включены наиболее популярные языки разработки приложений, средства работы с базами данных, а также другое программное обеспечение;
- операционная система поддерживает все основные аппаратные процессорные архитектуры, включая надежную поддержку SMP, MMP и кластерных систем. В других серверных средах такая поддержка отсутствует;
- практически все важнейшие промышленные, международные, официально утвержденные и неофициальные стандарты были впервые разработаны для UNIX и только впоследствии распространились и на другие операционные системы. В настоящее время основным стандартом является разработанная консорциумом X/Open Единая спецификация UNIX, содержащая более тысячи интерфейсов прикладных программ и поддерживаемая всеми основными производителями операционной системы UNIX. Несмотря на то что различные версии UNIX обладают уникальными возможностями, все они отвечают требованиям стандарта POSIX (Portable Operating System Interface, переносимый интерфейс операционной системы), а также стандарту X/Open Portability Guide, Edition 4 (XPG 4) и сертифицированы X/Open на соответствие стандартам UNIX 93;
- операционная система UNIX к настоящему времени утвердилась как платформа для персональных приложений, приложений для рабочих групп и приложений корпоративного класса.

В настоящее время единого стандарта для UNIX не существует, и на рынке присутствует множество версий этой операционной системы, каждая из которых имеет свои названия и особенности. Тем не менее все без исключения UNIX-системы имеют однотипную архитектуру, интерфейсы и среду программирования, что дает основание считать все UNIX-системы в той или иной степени родственными. Простота и способность операционных систем UNIX к расширению и модификациям являются весьма серьезными преимуществами по сравнению с другими системами, поэтому UNIX стали переносить на множество платформ.

Linux является одной из наиболее популярных в настоящее время реализаций операционной системы UNIX. Эта версия UNIX

обладает большинством свойств, присущих другим реализациям, и, кроме того, включает некоторые дополнительные возможности. Linux – это полная многозадачная многопользовательская операционная система, допускающая одновременную работу многих пользователей.

Операционная система Linux очень популярна среди миллионов пользователей. GNU/Linux вместе с набором инструментальных средств, по оценкам экспертов, охватывает около 40% рынка UNIX. Многие компании выпускают дистрибутивы Linux – пакеты, включающие ядро, множество утилит, приложений и программное обеспечение для установки ОС. GNU/Linux получила поддержку многих ведущих производителей программного обеспечения, таких как Oracle, IBM и др.

Для разработки программного обеспечения, работающего в Linux, был создан специальный фонд под названием Free Software Foundation (FSF), цель которого заключается в поиске источников финансирования разработки программного обеспечения GNU. Несмотря на относительно короткую историю существования, под эгидой проекта GNU было создано и адаптировано огромное количество программ, среди которых наиболее известными являются gcc (компилятор GNU C) и bash (командная оболочка).

Большинство из свободно распространяемого через Интернет программного обеспечения для Linux может быть откомпилировано для работы с любым дистрибутивом с минимальными изменениями. Более того, все исходные тексты для Linux, включая ядро, драйверы устройств, библиотеки, пользовательские программы и инструментальные средства, также распространяются свободно.

К специфическим особенностям Linux следует отнести контроль работ по стандарту POSIX (используемый оболочками, такими как csh и bash), работу с псевдотерминалами, поддержку национальных и стандартных клавиатур с динамически загружаемыми драйверами клавиатур.

Операционная система Linux поддерживает различные типы файловых систем. Некоторые из них, такие, например, как файловые системы ext2/3/4fs, были созданы специально для Linux. Кроме того, поддерживаются и другие типы файловых систем, такие, например, как Minix и Xenix. Система включает реализацию файловой системы MS-DOS, позволяющую прямо обращаться к файлам MS-DOS на жестком диске. Наконец, для работы с CD-ROM поддерживается стандарт файловой системы ISO 9660.

Как и другие операционные системы, Linux обеспечивает полный набор протоколов TCP/IP для сетевой работы. Сюда входят драйверы устройств для многих сетевых карт Ethernet, SLIP (Serial Line Internet Protocol, обеспечивающий пользователям доступ по TCP/IP при последовательном соединении), PLIP (Parallel Line Internet Protocol), PPP (Point-to-Point Protocol), NFS и т. д. В систему включена поддержка всего спектра сервисов TCP/IP (FTP, telnet, NNTP и SMTP).

Архитектура Linux

В этом разделе мы рассмотрим базовые концепции построения операционной системы Linux и взаимодействие ее различных функциональных частей. Знание основных принципов функционирования системы является основой для успешной работы в Linux и помогает эффективно использовать возможности этой мощной операционной системы.

В основу построения операционных систем Linux было положено несколько основных принципов.

- **Надежность.** Операционная система должна быть по возможности максимально надежной. Сбои в работе системы в большинстве случаев могут вызываться неполадками в аппаратной части, неправильными действиями пользователя либо некорректной работой программного обеспечения.

  Во всех этих случаях для повышения надежности требуется каким-то образом сохранять работоспособность системы, обеспечивая определенный минимальный уровень функционирования и возможности восстановления. Самым неприятным следствием сбоя в работе может быть потеря важных пользовательских данных, поэтому сохранение и восстановление данных являются основным критерием надежности системы.

  Одним из эффективных методов повышения надежности системы является разделение программного кода операционной системы и кода пользовательской программы таким образом, чтобы программа пользователя не могла разрушить выполняющийся программный код Linux. С другой стороны, пользовательская программа должна иметь доступ к различным ресурсам операционной системы: памяти, процессору, жестким дискам, периферийным устройствам (принтерам, плоттерам,

накопителям на магнитных лентах и т. д.), что является потенциально опасным для надежного функционирования системы. Компромиссным решением стала концепция построения операционной системы на основе базового программного модуля («ядра»), выполняющего обслуживание запросов программ пользователя, одновременно изолируя их от прямого доступа к ресурсам системы.

- **Возможность** одновременной работы нескольких пользователей с одной системой. В этом случае необходимо предоставлять ресурсы операционной системы (вероятно, одни и те же) нескольким пользователям одновременно, причем возможности доступа к одним и тем же ресурсам для разных пользователей могут отличаться.

  Операционные системы, допускающие работу в таком режиме, называются многопользовательскими. Очень часто разным пользователям требуется доступ к одним и тем же ресурсам, например к файлу на диске. При этом одни пользователи могут читать и записывать данные в файл, в то время как другие могут только читать данные из файла или вообще не иметь доступа к данным.

  Многопользовательская операционная система должна обладать механизмами разделения доступа к ресурсам и, кроме того, иметь возможности для защиты общих ресурсов от несанкционированного доступа. Linux относится именно к такому классу операционных систем.
- **Возможность** одновременного выполнения множества процессов. В операционной системе должна быть возможность выполнения одновременно множества процессов (работающих программ), как пользовательских, так и системных.

  При этом должен обеспечиваться механизм синхронизации процессов. Это означает, что операционная система должна контролировать запуск, выполнение и уничтожение процессов, обеспечивая работающим процессам доступ к требуемым ресурсам наиболее эффективным способом (мультизадачность).
- **Унифицированный** (единообразный) способ доступа к ресурсам операционной системы. Эта концепция положена в основу построения файловой системы Linux. В операционной системе Linux объектами файловой системы являются как файлы программ или данных, так и устройства, например принтеры, жесткие диски, терминальные линии.

Подобный подход очень удобен, поскольку позволяет работать с единым интерфейсом и использовать одни и те же функции как для работы с файлами данных, так и с файлами устройств. Существенным преимуществом такого подхода является и то, что можно использовать единые принципы для разделения ресурсов системы в многопользовательской среде и установки защиты объектов файловой системы.

Кроме того, разработчики программного обеспечения могут использовать единый интерфейс для разработки программ, что значительно снижает трудоемкость.

Реализованная на основе этих концепций операционная система Linux обладает следующими характеристиками:

- она легко переносима на другие аппаратные платформы;
- допускает работу в режиме вытесняющей многозадачности, обеспечивая работу процессов в изолированных адресных пространствах в виртуальной памяти;
- обеспечивает поддержку одновременной работы многих пользователей;
- поддерживает работу асинхронных процессов;
- имеет иерархическую файловую систему;
- обеспечивает поддержку независимых от устройств операций ввода/вывода путем использования специальных файлов устройств;
- предоставляет стандартный интерфейс для программ (программные каналы) и пользователей (командный интерпретатор, не входящий в ядро операционной системы);
- имеет встроенные средства мониторинга операционной системы.

Вышеперечисленные возможности операционной системы Linux можно представить пятиуровневой моделью (рис. 3.1).

Как видно по рис. 3.1, операционную систему Linux можно представить как совокупность пяти взаимосвязанных функциональных частей:

- аппаратной части (hardware);
- ядра;
- интерфейса системных вызовов;
- системных служебных программ (утилит) и командных оболочек (командных интерпретаторов);
- пользовательских программ.

Аппаратная часть представляет собой физические ресурсы си-

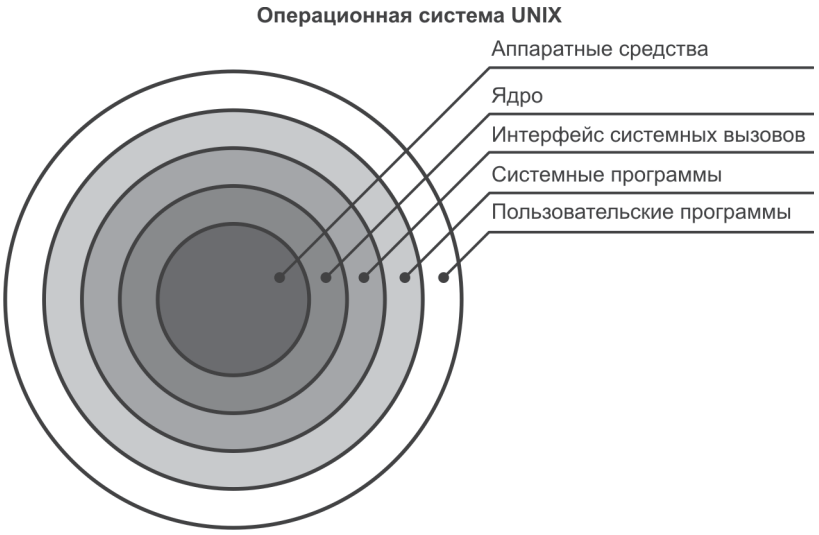

Рис. 3.1

стемы (процессор, оперативная память, жесткий диск, устройства ввода/вывода), непосредственный доступ к которым может осуществлять только ядро операционной системы. Прикладные пользовательские программы не могут получить прямого доступа к оборудованию системы, а взаимодействуют с ним посредством ядра, вернее через системные вызовы функций ядра.

Такое построение операционной системы обусловлено несколькими причинами. Во-первых, это гарантирует надежность работы системы, поскольку невозможно произвольным образом, например из программы пользователя, изменить конфигурацию системы, что чревато крахом UNIX. Во-вторых, ядро операционной системы балансирует работу всех подсистем без вмешательства пользователя, обеспечивая тем самым оптимальную производительность работы.

В-третьих, ограничение доступа к аппаратным средствам пользовательских программ обеспечивает надежную работу аппаратуры, поскольку ядро управляет функционированием физических устройств посредством специальных программ – драйверов устройств.

Операционная система Linux является многопользовательской и многозадачной системой. Это означает, что в ней могут работать одновременно несколько пользователей, каждый из которых может выполнять несколько задач одновременно.

В основе функционирования операционной системы Linux, как было упомянуто, лежит взаимодействие между пользовательскими программами, ядром и аппаратными ресурсами. Фактически функционирование операционной системы определяется особенностями работы ядра, поэтому остановимся на взаимодействии ядра и остальных функциональных частей Linux более подробно.

Компиляция и сборка ядра операционной системы UNIX обычно выполняются статически. Это означает, что ядро загружается как один большой исполняемый программный модуль при инициализации системы. Такой тип ядра называют монолитным. Некоторые версии операционных систем допускают работу с другим типом ядра, который называют модульным или, иногда, микроядром. При использовании модульного ядра дополнительные модули программного кода (обычно это драйверы устройств) подгружаются в оперативную память динамически, то есть по мере необходимости, например при включении аппаратного устройства. Такие дополнительные модули реализованы в виде загружаемых модулей ядра.

Преимущество модульного ядра состоит в том, что базовый модуль ядра имеет небольшой размер, быстрее загружается и требует меньше ресурсов операционной системы. В то же время монолитное ядро работает чуть быстрее, поскольку не требуется переключений контекста выполняемых процессов (что имеет место в случае загрузки/выгрузки дополнительных модулей) и дополнительной синхронизации, как при использовании отдельных модулей. Кроме того, в монолитном ядре реализовано намного больше функций управления аппаратурой, поскольку такое ядро содержит драйверы устройств.

В настоящее время большинство ядер Linux реализованы как комбинированные, то есть, являясь в принципе монолитными, допускают загрузку дополнительных модулей во время работы операционной системы. Современные версии ядра Linux-систем в большинстве своем обладают еще одной особенностью. Эта особенность – возможность функционирования ядра как совокупности отдельно выполняющихся потоков (kernel threads). Подобная особенность называется многопоточностью ядра и позволяет повысить эффективность функционирования как ядра, так и всей операционной системы. В первом приближении поток можно представить себе как отдельный выполняющийся фрагмент программного кода в рамках одного процесса. При этом ядро управляет выполнением потоков и их синхронизацией.

Более высокая эффективность выполнения многопоточных функций обусловлена тем, что переключение контекста отдельных потоков требует меньше времени, чем переключение контекста отдельных процессов. В значительной степени подобный выигрыш получается за счет того, что потоки выполняются в общем адресном пространстве, в то время как каждый процесс требует отдельного адресного пространства, а это занимает определенное время.

Особенностью современных операционных систем Linux является еще и то, что ядро таких систем, кроме обработки отдельных потоков, поддерживает также работу многопоточных пользовательских программ, что повышает их производительность. В этом случае многопоточные приложения выполняются как совокупность элементарных (lightweight) процессов, которые используют общее адресное пространство, общие страницы памяти и открытые файлы. Естественно, что для получения выигрыша в производительности выполняющаяся программа должна поддерживать реализацию многопоточности.

Большинство реализаций ядра операционной системы Linux выполняется в режиме невытесняющей многозадачности (non-preemptive multitasking). Это означает, что операционная система не может прерывать выполнение процесса, происходящегося в режиме ядра. Ядро, работающее в режиме вытесняющей многозадачности (preemptive multitasking), используется, как правило, в UNIX-подобных операционных системах, функционирующих в режиме реального времени.

Независимо от архитектуры ядро Linux-системы обеспечивает поддержку многопользовательского и многозадачного режима работы, выполняя следующие функции:

- создание, выполнение, остановку и завершение процессов, а также синхронизацию их взаимодействия;
- планирование приоритетов выполнения процессов путем выделения им времени центрального процессора. В этом случае центральный процессор выполняет процесс в течение определенного ядром интервала времени, после чего процесс приостанавливается и ядро начинает выполнение другого процесса. Через определенный интервал времени приостановленный процесс возобновляет выполнение и т. д.;
- выделение исполняемому процессу определенного объема оперативной памяти. В этом случае ядро защищает адресное пространство процесса от доступа из других процессов, одно-

временно позволяя разным процессам совместно использовать участки адресного пространства на определенных условиях. Если системе требуется некоторый объем свободной памяти, ядро освобождает память за счет процесса. При этом контекст (данные и ссылки) процесса сохраняется на жестких дисках или иных внешних устройствах. Такая реализация системы Linux называется системой со свопингом (подкачкой). Если же на жестком диске сохраняются страницы памяти, то такая система называется системой с замещением страниц;

- выделение памяти на устройствах постоянного хранения информации (жесткие диски и магнитные ленты) для обеспечения эффективного хранения и выборки данных пользователя. Эта функция ядра реализуется через обращение к файловой системе Linux.

Файловая система управляется посредством функций ядра, которые выделяют внешнюю память для файлов, выполняют отображение физической структуры файловой системы в логическую форму, доступную для работы пользователей, а также устанавливают атрибуты доступа к объектам файловой системы, защищая пользовательские файлы от несанкционированного доступа; управляют доступом процессов к периферийным устройствам, таким как терминалы, накопители на магнитных лентах и сетевое оборудование.

С точки зрения пользователя функции ядра можно представить схемой, показанной на рис. 3.2.

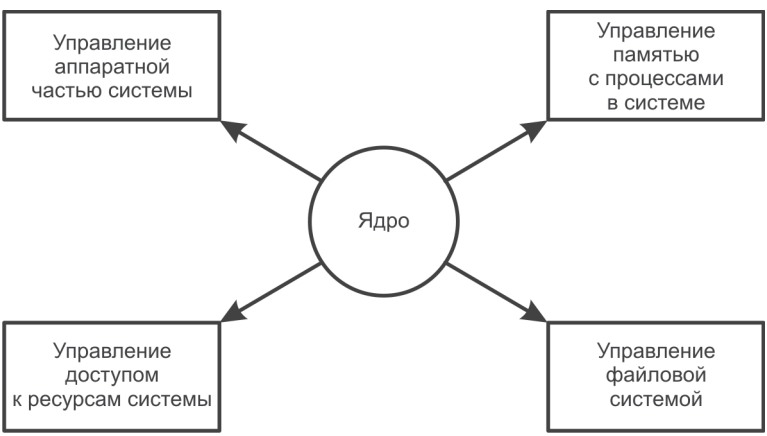

Рис. 3.2

Ядро операционной системы является прозрачным для пользовательской программы. Это означает, что детали взаимодействия программы пользователя и операционной системы скрыты от пользователя. К примеру, если программа пользователя обращается к какому-либо файлу и записывает в него данные, то ядро системы выполняет последовательность довольно сложных действий: определяет местоположение файла на носителе, получает информацию о расположении требуемых данных в физических секторах накопителя, определяет место записи блока данных в физическую область дискового пространства и т. д. Наконец, ядро вызывает драйвер устройства и передает ему параметры и код операции (записи данных), после чего и выполняется запись данных.

Кроме вышеперечисленных, ядро реализует ряд необходимых функций по обеспечению выполнения процессов пользовательского уровня, за исключением функций, которые реализуются на самом пользовательском уровне.

Например, ядро выполняет определенные действия, необходимые для работы командного интерпретатора shell. Такие функции командного интерпретатора, как чтение вводимых с терминала данных, динамическое создание процессов, синхронизация выполнения процессов, открытие программных конвейеров и переадресация ввода/вывода, реализуются через системные вызовы ядра.

Здесь нужно сделать небольшое отступление и чуть более подробно остановиться на некоторых терминах и понятиях, нередко используемых при анализе функционирования операционной системы Linux. Эти термины будут встречаться очень часто и являются фундаментальными при анализе операционной системы. К таким терминам относятся «программа» и «процесс».

Под термином «программа» мы будем понимать записанный на носителе исполняемый файл, в то время как термин «процесс» в первом приближении будет означать программу, находящуюся в стадии выполнения. Ввиду важности понятия «процесс» остановимся на нем подробнее.

Процесс можно представить как исполняемый модуль программного кода, которому предоставлены определенные ресурсы системы (память, процессорное время и т. д.). В операционной системе Linux может одновременно выполняться множество процессов (многозадачность), причем их число логически не ограничивается, и одна программа может создавать множество процессов.

При этом существующие в системе процессы могут создавать новые или завершать другие процессы. Ядро операционной системы синхронизирует выполнение этапов процесса и управляет реакцией на наступление различных событий. Благодаря наличию системы защиты процессы выполняются независимо и не влияют друг на друга.

Создание и выполнение процессов иллюстрирует рис. 3.3.

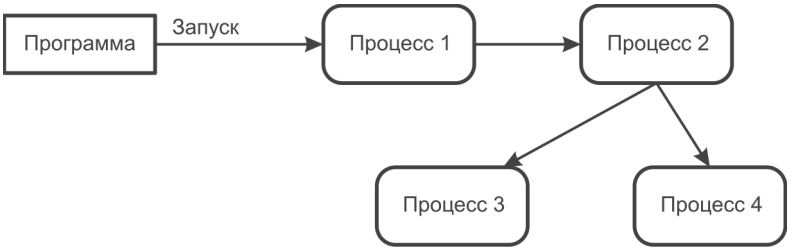

Рис. 3.3

В простейшем случае программа порождает один процесс, в других случаях таких процессов может быть множество. Термин «процесс» применим не только к пользовательским или системным программам, но и к ядру, поскольку оно само функционирует как совокупность взаимосвязанных процессов.

Все процессы, выполняющиеся в среде операционной системы Linux, могут выполняться либо в пользовательском режиме (User Mode), либо в режиме ядра (Kernel Mode). Подобное разделение обусловлено архитектурой системы и возможностью доступа процессов к ресурсам системы (об этом упоминалось ранее в этой главе). Пользовательский режим не позволяет напрямую обращаться к аппаратным ресурсам системы и системным структурам данных, в то время как процесс, выполняющийся в режиме ядра, имеет такую возможность.

В пользовательском режиме могут работать не только программы пользователя, но и значительная часть системных программ, входящих в состав операционной системы.

Возникает вопрос: каким образом пользовательский процесс в случае необходимости может получить доступ к ресурсам системы?

Операционная система Linux предоставляет процессам, работающим в режиме пользователя (User Mode), набор интерфейсов для взаимодействия с аппаратными устройствами, такими как процес-

сор, жесткие диски, принтеры и т. д. Linux реализует подобные интерфейсы между режимом пользователя и аппаратурой посредством так называемых системных вызовов (system calls), которые взаимодействуют с функциями ядра. Совокупность системных вызовов образует «интерфейс системных вызовов» (см. рис. 3.1).

Системные вызовы инициируют смену контекста выполнения процесса: процесс, работающий в режиме пользователя, переключается на выполнение в защищенном режиме (режим ядра). Такое переключение позволяет процессу вызывать защищенные процедуры ядра для выполнения системных функций. Таким образом, системные вызовы обеспечивают программный интерфейс для доступа к управлению системными ресурсами, такими как память, дисковое пространство и периферийные устройства. Системные вызовы реализованы в виде библиотеки времени выполнения (runtime library), а многие из них используются командными интерпретаторами shell.

Следует отметить, что системные функции ядра для корректной работы требуют упаковки аргументов специальным образом.

В качестве примеров системных вызовов можно привести низкоуровневые функции ввода/вывода, такие как **`open()`**, **`read()`**, **`write()`** и **`close()`**.

Системные вызовы обеспечивают выполнение целого ряда операций:

- трансляции операций пользователя в запросы к драйверам устройств;
- создания, запуска и уничтожения процессов;
- ввода/вывода;
- доступа к файлам и дисковым устройствам;
- поддержки терминальных устройств.

Наличие интерфейсов в форме системных вызовов предоставляет определенные преимущества разработчикам программ. Во-первых, процесс разработки программ становится намного легче, поскольку программисту нет необходимости изучать особенности программных интерфейсов низкого уровня для каждого из устройств.

Во-вторых, повышается надежность системы в целом, поскольку ядро Linux может проверить корректность запроса программы пользователя на уровне интерфейса, прежде чем ответить на такой запрос.

Наконец, наличие таких интерфейсов позволяет легко переносить программы на другие реализации Linux.

Рассмотрим пример взаимодействия пользовательского процесса и ядра операционной системы Linux. Предположим, что процессу пользователя необходимо открыть файл на диске для записи или чтения. В упрощенном виде последовательность шагов при открытии файла показана на рис. 3.4.

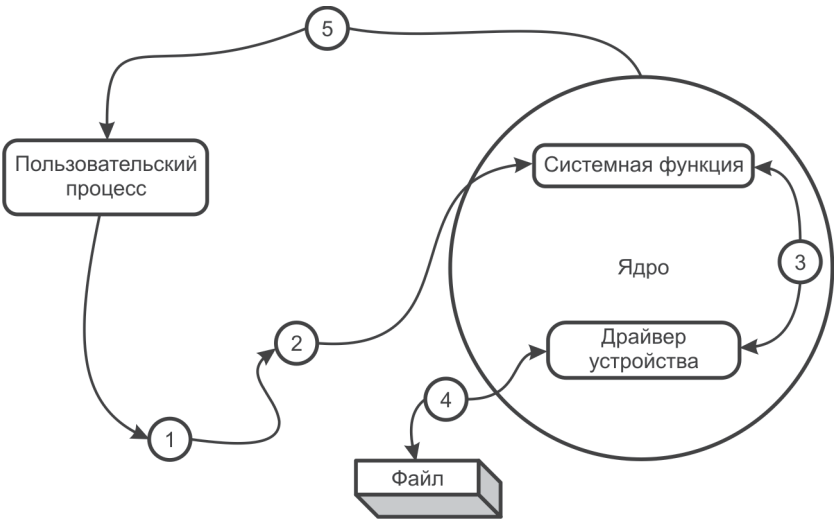

Рис. 3.4

На шаге 1 процесс пользователя инициирует запрос на доступ к файлу (открытие файла). При этом программный код пользовательского процесса выполняет системный вызов при помощи функции **`open()`** (шаг 2). Далее (шаг 3) выполняется переключение контекста процесса из пользовательского режима в режим ядра и происходит обращение к системной функции ядра, соответствующей системному вызову **`open()`**.

Системная функция выполняет ряд операций, необходимых для открытия файла, находящегося на диске, например формирует запросы к драйверу устройства (шаг 3) и анализирует статус (состояние) требуемого ресурса, полученный от драйвера. Драйвер устройства обращается к требуемому ресурсу и формирует статусную информацию и данные для системной функции (шаг 4).

На шаге 5 процесс пользователя получает от ядра результат выполнения запроса либо в виде определенных структур данных (для

системного вызова **open()** это дескриптор открытого файла), либо сообщение об ошибке. Нужно отметить, что системные вызовы в той или иной степени используются всеми без исключения пользовательскими и системными программами.

Следующий уровень функциональности, который мы рассмотрим, – системные программы (см. рис. 3.1). К системным программам, или, по-другому, к системному программному обеспечению, относят командные оболочки shell (интерпретаторы), команды и утилиты системного администрирования, драйверы и протоколы коммуникаций. Как известно, операционная система Linux включает ряд стандартных системных программ для выполнения задач администрирования, конфигурирования и поддержки файловой системы. Кроме того, к этой группе программ следует отнести утилиты:

- настройки параметров конфигурации системы;
- перекомпоновки ядра (если она необходима) и добавления новых драйверов устройств;
- создания и удаления учетных записей пользователей;
- создания и подключения физических файловых систем;
- установки параметров контроля доступа к файлам.

В качестве пользовательских программ могут выступать командные файлы, написанные с помощью командного интерпретатора shell, или разработанные на одном из языков высокого уровня (C, Python или др.) приложения. К пользовательским программам относятся многочисленные текстовые и графические редакторы, программы отправки и получения электронной почты и т. д.

Следует отметить, что в некоторых случаях пользовательским программам не требуется обращение к функциям ядра. Процессы, порожденные пользовательскими программами, защищены от других пользовательских процессов, не имеют доступа к функциям ядра, кроме как через системные вызовы, и, кроме того, не могут непосредственно обращаться к пространству памяти ядра.

Рассмотрим более подробно взаимодействие пользовательского процесса и ядра операционной системы.

Пространство памяти ядра представляет собой область памяти, в которой процессы ядра, или, по-другому, процессы, работающие в контексте ядра, реализуют функции ядра. Пространство ядра – защищенная область, и пользователь получает к ней доступ только через интерфейс системных вызовов. Пользовательский процесс не имеет прямого доступа ко всем инструкциям и физическим устрой-

ствам – их имеет процесс ядра. Процесс ядра также может менять карту памяти, что необходимо для переключения процессов (смены контекста). Пользовательский процесс начинает работать в режиме ядра в тот момент, когда выполняется программный код ядра посредством системного вызова.

Поскольку пользовательские процессы и ядро не имеют общего адресного пространства памяти, необходим механизм передачи данных между ними. При выполнении системного вызова аргументы вызова и соответствующий идентификатор процедуры ядра передаются из пользовательского пространства в пространство ядра. Идентификатор процедуры ядра передается либо через регистры процессора, либо через стек, а аргументы системного вызова передаются через область памяти вызывающего процесса. Область памяти пользовательского процесса содержит информацию о процессе, необходимую ядру, причем пользовательский процесс не может обращаться к пространству ядра, но ядро может обращаться к пространству процесса.

Одной из важнейших функций операционной системы является управление файловой системой. Файловая система представляет собой иерархически организованную структуру объектов, называемых файлами. Иногда выделяют группу объектов файловой системы, называемых каталогами, хотя каталог является разновидностью файла. Файловая система Linux обеспечивает унифицированный интерфейс доступа к данным, расположенным на различных носителях, и к периферийным устройствам. Файловая система контролирует права доступа к файлам, выполняет операции создания и удаления файлов, а также осуществляет запись/чтение данных файла.

Поскольку большинство прикладных функций выполняется через интерфейс файловой системы, следовательно, права доступа к файлам определяют привилегии пользователя в системе.

Особо следует отметить, что в операционной системе UNIX понятие «файл» трактуется в более широком смысле, чем в других операционных системах.

В общеупотребительном смысле под файлом понимают упорядоченный определенным образом набор данных в определенном формате (текст, графика, двоичные данные). В операционной системе UNIX к объектам файловой системы, помимо файлов, содержащих данные, также относятся и специальные файлы устройств (накопителей на жестких дисках, принтеров, сетевых интерфейсов и т. д.), а также сокеты и программные каналы.

Система управляет объектами файловой системы при помощи унифицированного набора системных функций ядра, к которым могут обращаться пользовательские процессы посредством системных вызовов, таких, например, как `open()`, `read()`, `write()` и `close()`. При этом все файловые операции, с точки зрения программы пользователя, выполняются одинаково как для файлов, содержащих данные, так и для специальных файлов устройств.

Далее мы рассмотрим важнейшие элементы программной архитектуры Linux, а начнем с учетных записей пользователей.

Учетные записи пользователей

Для того чтобы пользователь мог работать в операционной системе, нужно сначала создать для него учетную запись (account). В процессе создания учетной записи пользователю присваиваются имя, пароль, а также создается каталог, в который по умолчанию будет попадать пользователь после успешного входа в систему. Такой каталог часто называют домашним каталогом пользователя.

Любой пользователь операционной системы UNIX, помимо домашнего каталога, получает в свое распоряжение определенный набор ресурсов с установленными правами доступа к этим ресурсам. Например, пользователь может читать и записывать в файлы, выполнять печать документов на принтере, передавать и принимать сообщения электронной почты.

Для каждого пользователя в системе устанавливаются вполне определенные права (атрибуты) доступа к ресурсам, причем для разных пользователей они могут отличаться.

Все ресурсы операционной системы имеют определенные права доступа, причем часть таких прав устанавливается в процессе инсталляции операционной системы, остальные атрибуты доступа устанавливаются или изменяются специальным пользователем root, наделенным всеми правами по управлению (администрированию) операционной системы.

Пользователь root имеет все права на подключение, удаление и модификацию учетных записей, назначение и изменение паролей, назначение кода доступа к ресурсам операционной системы. Администрирование учетных записей напрямую связано с безопасностью операционной системы, поэтому правильное управление пользователями является основой безопасности операционной системы Linux.

Далее мы рассмотрим более подробно, каким образом можно работать с учетными записями пользователей операционной системы, и начнем с регистрации пользователей.

Прежде чем пользователь начнет работу в системе, он должен зарегистрироваться в ней. Во время регистрации каждому пользователю назначаются уникальный идентификатор сеанса и права доступа, позволяющие выполнять команды интерпретатора shell. В этот момент операционная система добавляет запись в файл, где хранятся записи о зарегистрированных пользователях.

Специальная программа, имеющая название getty, выдает на экран дисплея приглашение для регистрации пользователя. Для регистрации пользователь должен ввести специальный идентификатор (регистрационное имя), представляющий собой строку символов из цифр и букв алфавита. После ввода идентификатора процесс getty запускает программу регистрации login, которая использует в качестве параметра введенный идентификатор пользователя.

Программа login выполняет все необходимые действия по регистрации пользователя и установке параметров сеанса – процесса с уникальными идентификаторами пользователя и группы. Прежде всего процесс login проверяет содержимое файла /etc/passwd, чтобы определить, требуется ли ввод пароля при входе в систему. Если требуется, то система выдает приглашение к вводу пароля на экран.

Пароли большинства операционных систем в зашифрованном виде обычно хранятся в файле /etc/shadow. При успешной регистрации пользователя операционная система устанавливает параметры для текущего сеанса работы. Каждый сеанс пользователя получает числовой идентификатор пользователя и группы. Оба этих идентификатора определяют права доступа пользователя к объектам файловой системы, и без них невозможно выполнить ни одну команду.

Наконец, операционная система запускает командную оболочку (командный интерпретатор shell), которая позволяет выполнять команды.

Идентификация пользователей в операционной системе определяется несколькими файлами, имеющими ключевое значение для их работы: /etc/passwd, /etc/group и /etc/shadow. Информация о пользователе, сохраненная в этих файлах, позволяет системе устанавливать, изменять и выполнять другие функции управления учетной записью пользователя. Любой сеанс работы в UNIX начинается с регистрации пользователя.

После регистрации пользователя система устанавливает его параметры для текущего сеанса работы. Идентификатор пользователя (UID) – это 32-разрядное целое число между 0 и 2 147 483 647. Значение 0 идентификатора присваивается суперпользователю root. Идентификаторы 1 и 2 присваиваются пользователям с административными правами (bin, daemon).

Рассмотрим более подробно, каким образом пользователю назначаются права доступа к файлам. Как известно, все объекты файловой системы имеют два идентификатора: пользователя и группы. Оба они наследуются от создавшего их процесса и определяют права доступа к файлу. Пользователь и группа, идентификаторы которых связаны с файлом, считаются его владельцами. Изменить владельца файла можно с помощью команд chown и chgrp. Выделяют три категории доступа к объектам файловой системы:

- владелец файла (процесс, идентификатор пользователя которого равен идентификатору владельца файла);
- члены группы (процессы, идентификатор группы которых совпадает с идентификатором группы, установленной для файла);
- прочие – все остальные процессы.

Любому файлу при создании присваивается код доступа, представляющий собой комбинацию битов в индексном дескрипторе файла. Код доступа представляет собой комбинации единичных значений битов, имеющие такой смысл:

- 000400 (100h) – разрешается чтение для владельца файла;
- 000200 (80h) – разрешается запись для владельца файла;
- 000100 (40h) – разрешается запуск на выполнение владельцу файла;
- 000040 (20h) – разрешается чтение членам группы;
- 000020 (10h) – разрешается запись членам группы;
- 000010 (8h) – разрешается выполнение членам группы;
- 000004 (4h) – разрешается чтение прочим пользователям;
- 000002 (2h) – разрешается запись прочим пользователям;
- 000001 (1h) – разрешается выполнение прочим пользователям.

Все коды показаны в восьмеричном формате, а в скобках представлены их шестнадцатеричные значения, которые в данной нотации условимся обозначать с завершающим символом «h».

Код доступа к файлу может быть изменен только привилегированным пользователем root. Нам уже встречался термин «привилегированный пользователь», поэтому дадим более точное его определение.

Привилегированный пользователь – это пользователь, имеющий процесс с идентификатором, равным нулю. Независимо от кода защиты файла привилегированный пользователь имеет доступ ко всем файлам системы. Обычно в учетном файле пользователей /etc/passwd имеется привилегированный пользователь с именем root. Иногда вместо термина «привилегированный пользователь» используют другой – «суперпользователь».

Отдельные поля кода доступа предназначены для установки специальных атрибутов, позволяющих изменить идентификаторы пользователя и группы:

- 004000 (800h) – разрешение смены идентификатора пользователя при обращении к файлу (так называемый «Set-UID» флаг);
- 002000 (400h) – разрешение смены идентификатора группы при обращении к файлу (так называемый «Set-GID» флаг).

Зачем нужны такие атрибуты? Смысл состоит в том, что идентификаторы, полученные при запуске процесса («реальные идентификаторы»), могут отличаться от идентификаторов, полученных после выполнения системного вызова **`exec()`** («эффективные идентификаторы»). Изначально реальные и эффективные идентификаторы всегда совпадают, но если код доступа к файлу предусматривает смену идентификаторов, то после загрузки исполняемого файла с помощью системного вызова **`exec()`** реальные идентификаторы процесса изменяются на идентификаторы владельца (или группы) выполняемого файла, то есть становятся эффективными.

Права доступа процесса проверяются по его эффективным идентификаторам. Если, например, владельцем файлов данных TEXT1, TEXT2 и TEXT3 является пользователь user, то полный доступ к этим файлам разрешен только ему. Предположим, что пользователю user принадлежат исполняемые файлы PROGRAM1, PROGRAM2 и PROGRAM3.

Пусть требуется, чтобы другие процессы, которые хотят выполнять операции над файлами TEXT1, TEXT2 и TEXT3, могли это сделать с помощью программ PROGRAM1, PROGRAM2 и PROGRAM3. Для этого в коде доступа файлов PROGRAM1, PROGRAM2 и PROGRAM3 необходимо установить биты разрешения смены идентификатора пользователя для процесса, вызвавшего на выполнение один из этих файлов. В этом случае другой процесс (не пользователя user!), которому требуется доступ к одному из файлов TEXT1, TEXT2 или TEXT3, должен запустить через системный

вызов **exec()** соответствующую программу обработки PROGRAM1, PROGRAM2 или PROGRAM3, иначе он вообще не получит доступа к защищенным их владельцем файлам данных TEXT1, TEXT2 или TEXT3.

Установку и изменение требуемых атрибутов доступа к файлу для пользователя, а также флагов Set-UID и Set-GID можно выполнить с помощью системного вызова **chmod()**. Можно воспользоваться также командой **chmod**, которая представляет собой «обертку» соответствующего системного вызова и доступна из командной оболочки shell.

Информацию о реальных и эффективных идентификаторах выполняющегося процесса можно получить с помощью системных вызовов **getuid()** и **getgid()**. Выполняющийся процесс может динамически изменять свои идентификаторы с помощью системных вызовов **setuid()** и **setgid()**, однако такое допускается только для привилегированного процесса (выполняющегося с идентификатором root) или такого, у которого реальный идентификатор совпадает с устанавливаемым.

Например, специальная программа passwd позволяет изменить пароль пользователя в учетном файле пользователей /etc/passwd. Владельцем этих файлов является суперпользователь root. Код доступа файла /etc/passwd разрешает выполнять запись данных только владельцу, в то время как код доступа исполняемого файла passwd разрешает смену пароля пользователя. Следовательно, пользователь, отличный от root, может изменить свой пароль только с помощью программы passwd, что является корректным по отношению к пользователю, который всегда может изменить свой пароль, не сообщая об этом системному администратору.

Ключевую роль в регистрации и управлении учетными записями пользователей играют несколько файлов, упоминавшихся ранее в этой главе: /etc/passwd, /etc/group и /etc/shadow. Рассмотрим их более подробно и начнем с анализа формата файла паролей passwd, на котором основано применение других форматов подобных файлов паролей. Этот файл обычно расположен в каталоге /etc и состоит из однострочных записей, представляющих собой отдельные учетные записи. Поля каждой записи отделены друг от друга двоеточием.

Файл passwd имеет одинаковую структуру во всех операционных системах Linux. Вот пример записи файла /etc/passwd для пользователя yury:

```
$ cat /etc/passwd|grep yury
yury:x:500:500:Yury:/home/yury:/bin/bash
$
```

Все записи имеют семь полей, разделенных двоеточием. Приведем расшифровку (слева направо) отдельных полей, в качестве примера используя показанную выше запись:

- поле 1 – регистрационное имя пользователя (yury);
- поле 2 – зашифрованный пароль (x);
- поле 3 – идентификатор пользователя (500);
- поле 4 – идентификатор группы (500);
- поле 5 – информация о пользователе (Yury);
- поле 6 – рабочий каталог пользователя (/home/yury);
- поле 7 – используемая командная оболочка (/bin/bash).

Для работы в многопользовательских системах Linux имеет механизм группового доступа. Несмотря на то что учетная запись пользователя может входить в одну или несколько групп, принадлежать она может только одной группе. Для этого в файле паролей /etc/passwd присутствует поле идентификатора группы (group ID, GID).

Информация о принадлежности пользователей к тем или иным группам находится в файле /etc/group. Чтобы сделать пользователя членом какой-либо группы, необходимо внести соответствующие коррективы в файл /etc/group. Следует заметить, что в ранних версиях операционной системы UNIX пользователь мог входить только в одну группу. Для современных систем такое ограничение снято, и пользователь может являться членом 16 групп одновременно. Поле идентификатора группы в настоящее время практически не используется операционной системой, тем не менее ему присваивается определенное значение.

Рассмотрим теперь, как учетные записи пользователей создаются, изменяются и удаляются. Для создания, изменения и удаления учетных записей в операционных системах Linux имеется специальная группа команд. Вот их краткое описание:

- **useradd** – используется для создания новой учетной записи пользователя или изменения информации о пользователе;
- **userdel** – используется для удаления регистрационного имени пользователя из системы;
- **passwd** – позволяет изменить пароль пользователя;
- **su** – выполняет команду с заменой идентификатора пользователя и группы;
- **login** – инициализирует сессию пользователя в системе;

- **id** – отображает реальный и эффективный идентификатор пользователя и группы;
- **pwconv**, **pwunconv**, **grpconv**, **grpunconv** – выполняет преобразование обычных и теневых файлов паролей и групп;
- **pwck** – проверяет целостность файла паролей.

Проанализируем эти команды более подробно и начнем с команды **useradd**. Вызванная без опции **-D**, команда **useradd** создает новый бюджет пользователя, используя параметры командной строки. Остальные параметры принимаются по умолчанию. После выполнения команды новая учетная запись пользователя будет зарегистрирована в системе, будет создан домашний каталог пользователя, куда копируются файлы инициализации.

Создание, удаление и управление учетными записями может выполнять только суперпользователь root. Рассмотрим некоторые, наиболее часто используемые опции команды **useradd**:

- **-u идентификатор** – указывает идентификационный номер пользователя (UID). Этот номер должен представлять собой неотрицательное целое число, меньшее по значению, чем системный параметр MAXUID. Если номер не указан, то по умолчанию будет использован следующий доступный UID. Например, если в системе уже используются UID с номерами от 100 до 105, то следующий UID будет равен 106. Следует отметить, что идентификаторы с номерами 0–99 зарезервированы системой и использоваться не могут;
- **-o** – эта опция позволяет создать дубликат UID. Использовать ее следует крайне осторожно, поскольку усложняется обеспечение безопасности системы в целом из-за наличия неоднозначности соответствия UID определенному пользователю;
- **-g группа** – представляет собой целочисленный идентификатор или символьное имя существующей группы. Она определяет основную (primary) группу для нового пользователя;
- **-G группа** – представляет собой несколько элементов списка, разделенных запятыми, каждый из которых является целочисленным идентификатором или символьным именем существующей группы, при этом список может состоять из одного элемента. Содержимое списка устанавливает принадлежность пользователя к дополнительным группам, которые могут быть определены с помощью команды **newgrp**;
- **-d каталог** – начальный (домашний) каталог нового пользователя. По умолчанию в качестве начального используется ката-

лог HOME/registration_name, где HOME – базовый каталог для начальных каталогов новых пользователей, а registration_name – регистрационное имя нового пользователя;

- **-s Shell** – полный путь к командному интерпретатору, используемому пользователем сразу же после регистрации. По умолчанию этому полю незначение не присваивается, поэтому система использует стандартный командный интерпретатор /usr/bin/sh. Для командной оболочки Shell нужно указывать существующий исполняемый файл;
- **-c комментарий** – любая текстовая строка, кратко описывающая регистрационное имя (обычно указывает фамилию и имя реального пользователя). Эта информация хранится в записи пользователя в файле /etc/passwd. Размер данного поля не должен превосходить 128 символов;
- **-m** – создает домашний каталог для нового пользователя, если таковой еще не существует. Если каталог уже существует, вновь созданный пользователь должен обладать правами доступа к указанному каталогу;
- **-k skel_dir** – выполняет копирование содержимого каталога skel_dir в начальный каталог нового пользователя вместо использования стандартного «шаблонного» каталога /etc/skel, содержащего стандартные файлы, определяющие среду работы пользователя. Каталог skel_dir должен существовать до выполнения операции;
- **-f активно_дней** – максимально допустимый интервал времени в днях между использованиями регистрационного имени пользователя, когда это имя еще не объявляется недействительным. Обычно в качестве значений используются положительные целые числа;
- **-e дата** – дата, начиная с которой регистрационное имя пользователя нельзя будет применять: после этой даты ни один пользователь не сможет войти в систему, используя данное регистрационное имя;
- **login** – строка символов, задающая регистрационное имя для нового пользователя. В ней не должны присутствовать символы двоеточия и перевода строки, а первый символ не должен быть прописной буквой.

Рассмотрим пример создания учетной записи пользователя с регистрационным именем alex. Для создания учетной записи пользователя необходимо зарегистрироваться в системе как суперпользо-

ватель root. Перед созданием учетной записи желательно просмотреть опции по умолчанию для команды **useradd**. Они могут быть, например, такими:

```
$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

По результату выполнения команды **useradd -D** можно сделать несколько важных выводов. Во-первых, в качестве корневого каталога для вновь создаваемых пользователей используется каталог /home. Во-вторых, пустое поле значения параметра EXPIRE означает, что учетная запись пользователя никогда не будет заблокирована. Наконец, в качестве командного интерпретатора для всех вновь создаваемых пользователей по умолчанию установлен /bin/bash.

Создадим учетную запись пользователя alex. Для этого введем команду **useradd**, в которой зададим регистрационное имя:

```
$ sudo useradd alex
```

Если команда выполнена успешно, то пользователь **alex** будет зарегистрирован в системе, и в файл /etc/passwd будет добавлена запись, идентифицирующая этого пользователя. Просмотреть ее можно с помощью команды **cat**:

```
# cat /etc/passwd|grep alex
alex:x:2307:2307::/home/alex:/bin/bash
```

Команда **useradd** автоматизирует процесс регистрации пользователя, но можно сделать это вручную, если нужно установить какие-либо индивидуальные параметры для пользователя, например командную оболочку или домашний каталог. Процедура не очень сложна и требует выполнения нескольких шагов, перечисленных далее.

Предположим, необходимо создать учетную запись пользователя с регистрационным именем newuser. Вначале просмотрим файл /etc/passwd/ на предмет поиска наибольшего UID. Наибольшее значение UID (2307) имеет вновь созданный пользователь alex, поэтому следующим UID может быть 2308. Возьмем это на заметку и выполним следующую последовательность шагов:

1. Добавим в файл /etc/passwd запись о пользователе newuser:

   ```
   $ echo newuser:x:2308:2308::/home/newuser:/bin/bash >>/
   etc/passwd
   ```

2. Далее создадим начальный каталог пользователя newuser, выбрав путь к /home/newuser:

   ```
   $ sudo mkdir /home/newuser
   ```

3. Делаем пользователя newuser владельцем каталога /home/newuser:

   ```
   $ sudo chown newuser newuser
   ```

4. Приводим в соответствие записи файлов /etc/passwd и /etc/shadow с помощью команды pwconv:

   ```
   $ sudo pwconv
   ```

Команда **pwconv** создает файл shadow из passwd, при этом может использоваться и существующий файл shadow (он будет перезаписан). Команда работает следующим образом:

1. Сначала удаляются те записи в теневом файле shadow, которые не существуют в основном файле паролей passwd.
2. Обновляются те теневые записи, для которых в полях пароля в основном файле не стоит «**x**». Добавляются все недостающие теневые записи.
3. Пароли в основном файле заполняются символами «**x**».

Удалить учетную запись пользователя в операционных системах Linux можно с помощью команды **userdel**. Эта команда удаляет информацию о пользователе из системы, выполняя соответствующие изменения в регистрационных файлах и файловой системе. Кроме этого, команда запоминает идентификационный номер удаляемого пользователя (UID) в файле /etc/security/ia/ageduid, для того чтобы этот идентификатор не использовался повторно в течение определенного периода времени. Такой механизм называется «устареванием идентификатора» (UID aging).

Команда имеет синтаксис:

```
userdel [-r] [-n_месяцев] имя
```

Опции команды имеют следующий смысл:

- **-r** – удаление начального каталога пользователя из системы, при этом каталог должен существовать. Если команда выполнена успешно, файлы и подкаталоги в начальном каталоге будут недоступны;

- **-n_месяцев** – задает интервал времени в месяцах, указывающий, как долго идентификатор пользователя должен устаревать перед повторным применением. Если параметр равен –1, то идентификатор пользователя никогда не будет повторно использован, если он равен 0, то идентификатор пользователя можно использовать немедленно. Если опция **-n** не задана, принимается значение устаревания по умолчанию.

Изменить параметры учетной записи пользователя в операционной системе Linux можно при помощи команды **usermod**. Эта команда модифицирует файлы, содержащие информацию об учетных записях пользователей. Допустимы следующие опции:

- **-A метод|DEFAULT** – указывает новый метод идентификации пользователя и представляет собой имя программы, отвечающей за допустимую идентификацию пользователя. Строку DEFAULT можно использовать для установки стандартного метода идентификации;
- **-c комментарий** – указывает на другой комментарий для записи пользователя в файле паролей;
- **-d домашний_каталог** – новый домашний каталог пользователя. При указании опции –m содержимое текущего домашнего каталога будет перемещено в новый домашний каталог, который будет создан, если еще не существует;
- **-e дата** – дата, после которой учетная запись пользователя устареет. Дата указывается в формате MM/DD/YY;
- **-f активно_дней** – число дней между датой устаревания пароля и датой, когда учетная запись пользователя будет заблокирована. Значение, равное 0, блокирует учетную запись пользователя в момент устаревания пароля, а значение –1 запрещает блокировку (значение по умолчанию);
- **-g группа** – имя группы или номер группы, которая будет присвоена пользователю после входа в систему, причем группа с указанным именем должна существовать. Номер группы также должен ссылаться на существующую группу (по умолчанию равен 1);
- **-G дополнительная_группа** – список дополнительных групп. Данный пользователь также является членом этих групп. Каждая группа отделяется от следующей группы запятой без пробелов. Группы являются предметом для некоторых ограничений, например группа, заданная с опцией **-g**. Если пользова-

тель является членом группы, которая не находится в списке, то пользователь будет удален из группы;

- **-l новое_имя** – имя пользователя будет изменено с имя на новое_имя. Ничего другого сделано не будет. В частности, домашний каталог пользователя должен быть, вероятно, изменен;
- **-s Shell** – имя командного интерпретатора, который будет использоваться новым пользователем при входе в систему. Установку этого поля в пустое значение будет выбирать системный shell по умолчанию;
- **-u uid** – числовое значение идентификатора пользователя (UID). Это значение должно быть уникальным, исключение составляет использование опции **-o**. Значение должно быть положительным. Как было сказано ранее, значения между 0 и 99 обычно зарезервированы для системных бюджетов. Для любых файлов, владельцем которых является пользователь и которые находятся в домашнем каталоге пользователя, идентификатор пользователя (UID) будет изменяться автоматически. Для файлов вне домашнего каталога пользователя идентификатор пользователя должен быть изменен вручную.

Важное замечание: если пользователь находится в системе, изменить его имя не удастся. Рассмотрим практический пример использования команды **usermod**. Ранее мы создали учетную запись пользователя newuser. Изменим регистрационное имя пользователя и домашний каталог. Напомню, как выглядит запись для пользователя newuser в файле /etc/passwd:

```
$ cat /etc/passwd|grep newuser
newuser:x:2308:2308::/home/newuser:/bin/bash
```

Присвоим пользователю newuser регистрационное имя moduser и изменим домашний каталог на /home/moduser:

```
$ usermod -l moduser -d /home/moduser -m newuser
```

Можно проверить, что изменения выполнены успешно:

```
$ cat /etc/passwd|grep moduser
moduser:x:2308:2308::/home/moduser:/bin/bash
```

Проверить наличие домашнего каталога пользователя **moduser** можно, задав команду:

```
$ ls -l /home
total 24
drwxr-xr-x 10 moduser root 4096   Aug 18 22:51 moduser
drwx------ 26 yury    yury 4096   Aug 18 00:24 yury
```

Здесь нужно сделать одно важное замечание: при изменении регистрационного имени пользователя его UID остается неизменным. Это свидетельствует о том, что операционная система оперирует с одной и той же учетной записью пользователя, несмотря на то что регистрационное имя пользователя изменилось. Таким образом, можно сделать очень важный вывод: для UNIX определяющим фактором при работе с пользователем является его идентификатор (UID).

Еще одной командой, которую мы проанализируем, является **passwd** – она позволяет изменить пароли пользователей. Обычные пользователи могут изменить пароль только для своей учетной записи, в то время как суперпользователь root может изменять пароли всех пользователей.

Команда **passwd** также позволяет изменить информацию об учетной записи: полное имя пользователя, его командный интерпретатор, дату истечения срока используемого пароля и интервал времени, в течение которого пароль действует.

Команда имеет синтаксис:

```
passwd [-f] имя
passwd [-g] [-r|R] группа
passwd [-x max] [-n min] [-w warn] [-i inact] имя
passwd {-l|-u|-d|-S} имя
```

Если пользователь имеет пароль, то перед установкой нового пароля команда **passwd** предлагает ввести старый пароль, который хранится в зашифрованном виде. Пользователь может ввести правильный пароль только один раз, хотя суперпользователь root может пропустить этот шаг, поэтому если пароль забыт, то его все же можно изменить. После введения пароля команда проверяет наличие разрешения на изменение пароля в данное время. Если это не разрешено, программа passwd завершает свою работу без изменения пароля.

Рассмотрим использование некоторых опций команды **passwd** более подробно.

- **-g** – замена пароля для заданной группы. Эта опция требует наличия прав суперпользователя или администратора за-

данной группы. Опция **-r** применяется совместно с опцией **-g** для удаления текущего пароля заданной группы, что делает группу доступной всем членам. Опция **-R** используется совместно с опцией **-g** для ограничения доступа к группе всем пользователям;

- информация об истечении срока действия пароля может быть изменена суперпользователем с помощью опций **-x**, **-n**, **-w** и **-i**. Опция **-x** используется для установки максимального числа дней, в течение которых пароль остается допустимым, при этом после max дней требуется его изменение. Опция **-n** служит для установки минимального числа дней, по истечении которых пароль может быть изменен. Пользователю запрещается изменять пароль в течение min дней. Опция **-w** предназначена для установки числа дней, в течение которых пользователь будет получать предупреждающее сообщение об истечении времени действия его пароля, причем сообщения будут выводиться в течение warn дней, напоминая пользователю, сколько дней осталось до момента устаревания его пароля. Опция **-i** запрещает использование учетной записи пользователя по истечении промежутка времени после устаревания пароля. Если устаревший пароль остается неизменным в течение inact дней, он не будет вновь принят системой.

Обслуживание учетной записи. Учетные записи пользователей могут быть заблокированы и разблокированы при помощи опций **-l** и **-u**. Опция **-l** блокирует учетную запись, изменяя пароль таким образом, что он становится непригодным для шифрования. Опция **-u** разблокирует учетную запись, изменяя пароль к его предыдущему значению.

Статус учетной записи может быть получен с помощью использования опции **-S**. Статусная информация состоит из шести полей. Первое поле кодируется следующим образом: L – если бюджет пользователя заблокирован, NP – если не существует пароля для данной учетной записи и P – если пароль используется. Во втором поле указана дата последнего изменения пароля. Следующие четыре поля – минимальное время до истечения срока действия пароля, максимальное время до истечения срока действия пароля, период вывода предупреждающего сообщения об истечении срока действия пароля и период неактивности для этого пароля.

Опция **-f** требует от пользователя изменить пароль при следующем входе в систему.

При выборе пароля можно руководствоваться нижеприведенными соображениями. Безопасность пароля зависит в первую очередь от применяемого алгоритма шифрования и размера ключа. В операционных системах UNIX метод криптографии основывается на алгоритме NBS DES, который имеет очень высокую степень безопасности, при этом размер ключа зависит от выбранного пароля. Еще одно эмпирическое правило: лучше не выбирать пароль, в котором используются литературные выражения. Нередко пользователь выбирает пароль, основанный на личных данных (месяц, год рождения и т. д.). Подобный пароль расшифровывается довольно легко, поэтому такой «алгоритм шифрования» не годится.

Вместе с тем пароль должен хорошо запоминаться, поэтому неплохим вариантом в этом смысле является пароль, представляющий собой знакомое слово, части которого разделены специальными символами, или законченное слово, состоящее из двух слов, объединенных вместе и разделенных специальными символами или цифрами.

Примерами таких паролей являются Pe&ter$sbu)rg, my!Pas#s%word. Еще одним методом составления паролей выступает комбинирование первых символов какой-либо фразы из литературного произведения.

Воспользуемся, например, для создания пароля фразой

```
Astala vista
```

Пароль может быть таким:

```
As79tal01avi&sta
```

Это достаточно бессмысленный пароль, который, однако, несложно запомнить.

Файловая система Linux

Файловая система – наиболее важная и функционально самая сложная часть операционной системы Linux, выполняющая функцию организации хранимых ресурсов операционной системы.

Особенностью Linux является то, что все объекты операционной системы представляют собой файлы. Это означает, что файлами являются физические устройства, например последовательный и параллельный порты, а также области дисковой и оперативной памяти, именованные каналы и сетевые соединения (сокеты).

Такая унификация очень удобна, поскольку обеспечивает единый программный интерфейс для доступа к объектам независимо от их реализации на физическом уровне. Например, системные вызовы **`open()`**, **`read()`**, **`write()`** и **`close()`** позволяют выполнять операции ввода-вывода единообразным способом как для файлов, находящихся на жестком диске, так и для физических портов ввода-вывода или программных каналов.

Естественно, в файловую систему входят собственно и файлы в том смысле, в каком мы привыкли их воспринимать, – в виде наборов данных, содержащихся на жестких дисках или иных носителях.

Рассмотрим более подробно основные типы файлов, используемые в операционной системе. К ним следует отнести файлы, содержащие двоичные данные (бинарные файлы), файлы устройств, сокеты и именованные каналы, а также символические и жесткие ссылки.

Бинарные файлы – это наборы двоичных битов, содержащие тот или иной тип информации: текст, рисунки или программы, аудиоданные и т. д. Этот тип файлов наиболее широко используется, и когда говорят о файле, то обычно подразумевают именно такие объекты. Интерпретация данных, записанных в двоичные файлы, возлагается на операционную систему или другие программы. Например, когда говорят о текстовом файле, то подразумевают набор текстовых данных, содержащихся в этом файле и обрабатываемых программой-редактором. Если говорят о программном файле, то это означает, что данные, записанные в такой файл, интерпретируются операционной системой как набор определенных команд, подлежащих выполнению.

Файлы устройств позволяют операционной системе Linux и другим программам взаимодействовать с аппаратными средствами и периферийными устройствами системы. Здесь необходимо сделать важное замечание: нужно отличать файлы устройств от драйверов устройств. Управление конкретным физическим устройством осуществляется посредством специальной программы, которая называется драйвером устройства. Файлы устройств сами по себе не являются драйверами: их можно представить как шлюзы, через которые драйвер получает запросы.

Когда ядро получает запрос к файлу устройства, оно просто передает этот запрос соответствующему драйверу. Таким образом, файлы устройств позволяют программам взаимодействовать с драйверами ядра. По своей структуре они не являются файлами данных, но об-

рабатываются базовыми средствами файловой системы, а их характеристики записываются на диск. Взаимодействие пользовательской программы, файла устройства и драйвера показано на рис. 3.5. Здесь пользовательская программа обращается к устройству печати, подсоединенному к параллельному порту (ему может соответствовать файл устройства /dev/lp0):

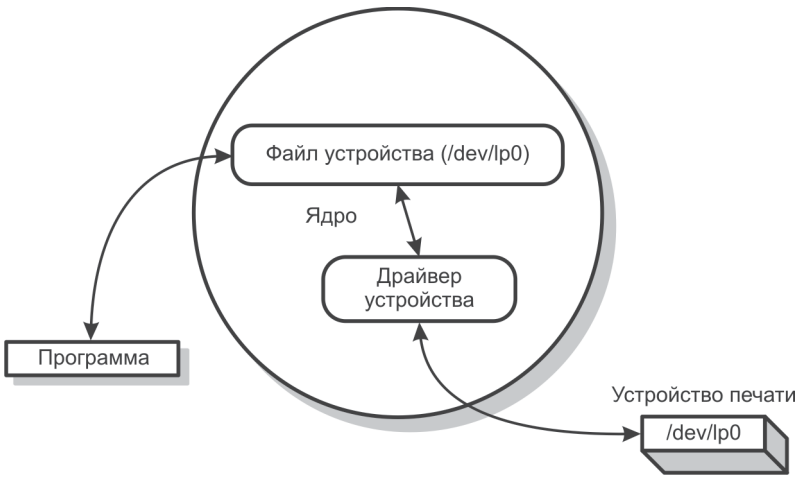

Рис. 3.5

Файлы устройств, в свою очередь, можно разделить на два типа: файлы байт-ориентированных устройств и файлы блок-ориентированных устройств. Файлы байт-ориентированных устройств позволяют связанным с ними драйверам выполнять свою собственную буферизацию ввода/вывода. Файлы блок-ориентированных устройств обрабатываются драйверами, которые осуществляют ввод/вывод большими блоками и возлагают обязанности по выполнению задач буферизации на ядро.

Некоторые типы аппаратных средств, такие как накопители на жестких дисках и DVD, могут быть представлены файлами любого типа.

В системе может присутствовать несколько однотипных устройств, поэтому файлы устройств характеризуются двумя номерами: старшим и младшим. Старший номер устройства информирует ядро, к какому драйверу относится данный файл, а младший номер сообщает драйверу, к какому физическому устройству следует обращать-

ся. Например, старший номер устройства 6 в Linux обозначает драйвер параллельного порта. Первый параллельный порт (/dev/lp0) будет иметь старший номер 6 и младший номер 0.

Некоторые драйверы используют младший номер устройства нестандартным способом. Например, драйверы накопителей на магнитных лентах часто руководствуются им при выборе плотности записи, а также определяют, нужно ли перемотать ленту после закрытия файла устройства. В некоторых системах драйвер терминала, управляющий последовательными устройствами, использует младшие номера устройств для идентификации модемов.

Рассмотрим еще один тип файла Linux, который называют сокетом. Сокеты инкапсулируют соединения между процессами, позволяя им взаимодействовать, не подвергаясь влиянию других процессов. В Linux поддерживается несколько типов сокетов, определяемых протоколом взаимодействия между процессами. Использование большинства сокетов предполагает наличие сети.

Тем не менее так называемые локальные или UNIX-сокеты можно использовать и для взаимодействия процессов на локальной машине. Следует заметить, что TCP-сокеты вполне подходят для межпроцессорного взаимодействия, поскольку любая UNIX-система поддерживает интерфейс обратной связи (loopback interface) с сетевым адресом 127.0.0.1, который можно использовать при формировании соединения через сокеты.

Другие процессы, не участвующие во взаимодействии, распознают сокеты как элементы каталога, но выполнять операции чтения и записи над ними не могут. Взаимодействие процессов посредством сокетов очень популярно в самой операционной системе: с сокетами работают система печати, система X Window и система Syslog. Более детально взаимодействие процессов по сети мы будем рассматривать в последующих главах, сейчас же замечу, что сокеты создаются с помощью системного вызова **`socket()`**. При закрытии соединения с обеих сторон сокет можно удалить посредством команды **`rm`** либо системного вызова **`unlink`**.

Еще один тип файлов, часто используемый, – именованные каналы. Подобно сокетам, именованные каналы обеспечивают межпроцессорное взаимодействие на одной машине. Именованные каналы создаются командой **`mknod`**, а удаляются командой **`rm`**.

К отдельному типу объектов файловой системы можно отнести символические и жесткие ссылки, которые используются для обеспечения альтернативных способов обращения к файлам.

В современных файловых системах Linux определен интерфейс уровня ядра, позволяющий работать с различными аппаратными интерфейсами. При этом часть файлов обрабатывается традиционной дисковой подсистемой, другие управляются отдельными драйверами ядра, как в случае сетевых файловых систем, где используется драйвер, перенаправляющий запросы серверу на другой компьютер.

Большинство операционных систем Linux поддерживают несколько типов файловых систем. Кроме базовой версии, берущей начало от 4.3BSD, поддерживаются и другие файловые системы, например обладающие повышенной надежностью или упрощенными средствами восстановления после сбоев (VXFS в HP-UX), а также системы, построенные на иных типах носителей (компакт-диски стандарта ISO-9660).

Файловую систему можно рассматривать, с одной стороны, как логическую структуру, в виде дерева каталогов и файлов, с четко установленной иерархией. Именно в таком виде и представляется файловая система пользователю Linux. С другой стороны, файловая система – это совокупность расположенных на физическом носителе упорядоченных и неупорядоченных двоичных данных. Пользователь обычно не имеет доступа непосредственно к блокам данных на носителях, хотя и может управлять ими при помощи программ, в которых используются соответствующие системные вызовы. Управление физической файловой системой – прерогатива функций ядра, поэтому вмешательство в этот процесс очень опасно, даже если вы хорошо представляете себе, что делаете.

Файловые системы обычно располагаются на жестких дисках, каждый из которых состоит из одной или нескольких логически связанных групп цилиндров, называемых разделами (partitions). Физическое расположение и размер раздела устанавливаются при форматировании диска. В операционных системах Linux разделы являются независимыми устройствами, доступ к которым осуществляется как к различным носителям данных. Один раздел содержит, как правило, только одну физическую файловую систему.

Для операционных систем Linux разработано много типов файловых систем (ufs, ext2/3/4, vxfs и т. д.), каждая из которых имеет свои особенности. Операционная система и работающие программы пользователей работают не с физическими блоками данных, а с логическими структурами данных, которые образуют логическую файловую систему.

Несколько слов о терминологии. Очень часто при использовании термина «файловая система» подразумевается, что речь идет о логической файловой системе. В дальнейшем, если не будет особо оговорено, эти термины мы будем употреблять как синонимы.

Конечный пользователь вряд ли смог бы эффективно работать с объектами файловой системы, если бы имел дело с логическими структурами, поэтому для конечного пользователя операционная система Linux использует интерфейс высокого уровня, в котором файловая система представляется как единая иерархическая структура, организованная в виде дерева каталогов.

Основой любой файловой системы является корневой каталог (обозначается как /). Все остальные каталоги и файлы располагаются в рамках структуры, порожденной корневым каталогом (в нем и в его подкаталогах), независимо от их физического местонахождения. Замечу, что каталог является обычным файлом, содержащим информацию о других файлах. Он позволяет четко структурировать объекты файловой системы. Структуру файловой системы Linux можно представить себе так, как показано на рис. 3.6.

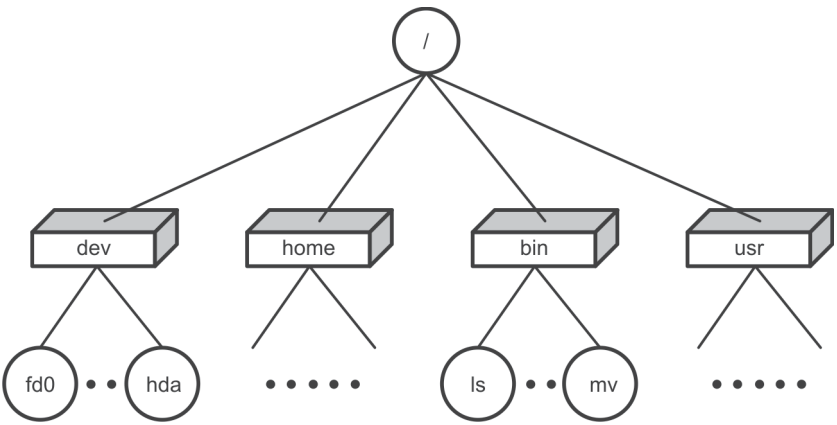

Рис. 3.6

Цепочка имен каталогов, через которые необходимо пройти для доступа к заданному файлу, вместе с именем этого файла называется путевым именем. Путевые имена могут быть абсолютными (например, /tmp/afile) или относительными (например, book3/filesystem). Последние интерпретируются, начиная с текущего каталога. На име-

на файлов и каталогов накладываются определенные ограничения: имя каталога должно состоять не более чем из 255 символов, а в отдельном путевом имени не должно быть более 1023 символов.

Для корневого каталога обязательно создается отдельная физическая файловая система, а сам он является точкой ее монтирования, о чем свидетельствует наличие подкаталога lost+found. Корневой каталог должен быть всегда доступен и монтируется автоматически при запуске системы. Следует сказать, что при такой структуре все остальные физические файловые системы с формальной точки зрения для функционирования операционной системы Linux не нужны.

В большинстве операционных систем используется более-менее стандартная структура каталогов файловой системы, при этом все каталоги имеют предопределенные назначения (хотя соблюдать это вовсе не обязательно). Назначение каталогов приводится далее:

- /bin – содержит пользовательские выполняемые программы – сейчас обычно является символической ссылкой, указывающей на /usr/bin;
- /dev – каталог для специальных файлов устройств – может иметь подкаталоги для различных классов и типов устройств, например dsk, rdsk, rmt, inet;
- /etc – каталог для конфигурационных файлов – может иметь подкаталоги для различных компонентов и служб (конфигурационные файлы в Linux – обычные текстовые файлы);
- /home – каталог для размещения начальных каталогов пользователей – часто является точкой монтирования отдельной физической файловой системы;
- /lib – каталог для библиотек – сейчас обычно является символической ссылкой, указывающей на /usr/lib;
- /lost+found – подкаталог, имеющийся в каждом каталоге, являющемся точкой монтирования физической файловой системы на диске;
- /mnt – точка монтирования для файловых систем на съемных носителях или дополнительных дисках (может содержать подкаталоги для отдельных типов носителей, например CD-ROM или floppy, может быть пустой);
- /opt – каталог для дополнительного коммерческого программного обеспечения (может быть пустым или отсутствовать);
- /proc – каталог псевдофайловой системы, предоставляющей в виде каталогов и файлов информацию о ядре, памяти и процессах, работающих в системе;

- /sbin – каталог для системных выполняемых программ, необходимых для решения задач системного администрирования;
- /tmp – каталог для временных файлов – имеет установленный sticky-бит (от англ. *sticky* – липучка) и доступен для записи и чтения всем пользователям, обычно создается в виде отдельной физической файловой системы, в том числе в виртуальной памяти;
- /var – в Linux этот каталог является аналогом каталога, используемого для хранения файлов различных сервисных подсистем (/usr/spool), например файлов журналов системы. Так, основной журнал системы, управляемый доменом syslogd, размещается в виде нескольких файлов в подкаталоге /var/adm. Там же, в файле /var/adm/messages, сохраняются сообщения времени загрузки. Имеет смысл создавать отдельную физическую файловую систему для размещения этого каталога;
- /usr – в этом каталоге находятся выполняемые программы, библиотеки, заголовочные файлы, справочные руководства (/usr/share/man), исходные тексты ядра и утилит системы операционной системы Linux и т. д. Часто каталог является точкой монтирования отдельной физической файловой системы. Далее представлены основные его подкаталоги:
  - ✧ /usr/bin – основные выполняемые программы и утилиты;
  - ✧ /usr/include – заголовочные файлы библиотек; может содержать подкаталоги;
  - ✧ /usr/lib – статически и динамически компонуемые библиотеки; может содержать подкаталоги;
  - ✧ /usr/local – каталог для дополнительного свободно распространяемого программного обеспечения, содержит структуру подкаталогов, аналогичную корневому каталогу (bin, etc, include, lib и т. д.).

Наличие, назначение и использование других каталогов верхнего уровня и подкаталогов зависит от версии операционной системы Linux, установленного системного и прикладного программного обеспечения и конфигурации системы, созданной пользователем или системным администратором.

В операционной системе Linux путевое имя файла принято называть жесткой ссылкой. Большинство файлов в операционной системе Linux имеет всего одну жесткую ссылку. Однако при помощи команды **`ln`** пользователь может создавать дополнительные жесткие ссылки на файлы.

Пусть имеется путевое имя файла /home/user1/user1.text. Тогда после выполнения команды

```
$ ln /home/user1/user1.text /home/user1/text.new
```

к этому же файлу можно обращаться как к /home/user1/text.new.

Ссылка, которую называют символической, или «мягкой», обеспечивает возможность вместо путевого имени файла указывать псевдоним. Когда ядро обнаруживает символическую ссылку при поиске файла, оно извлекает из нее хранящееся в ней путевое имя. Различие между жесткими и символическими ссылками состоит в том, что жесткая ссылка – прямая, то есть указывает непосредственно на индексный дескриптор файла, тогда как символическая ссылка указывает на файл по имени. Файл, адресуемый символической ссылкой, и сама ссылка физически являются разными объектами файловой системы.

Символические ссылки создаются командой **ln -s**, а удаляются командой **rm**. Поскольку они содержат произвольные путевые имена, то могут указывать на файлы, хранящиеся в других файловых системах, и даже на несуществующие файлы. Иногда несколько символических ссылок образуют цикл.

Для файла /home/user1/user1.text вместо жесткой ссылки можно создать символическую:

```
$ ln -s /home/user1/user1.text /home/user1/text.new
```

Обозначение .. в путевых именах, включающих символические ссылки, лучше не использовать, поскольку по символическим ссылкам нельзя проследовать в обратном направлении.

До сих пор мы рассматривали файловую систему, не выдвигая никаких предположений о том, каким образом операционная система делает доступными ресурсы файловой системы для пользователя. Все современные операционные системы скрывают детали операций с файловой системой от пользователя, предоставляя ему полностью функциональную логическую файловую систему в виде дерева каталогов. Все основные операции по инициализации физических файловых систем, распределению дискового пространства, форматированию дисков и их подключению к системе выполняются, как правило, в процессе инсталляции операционной системы Linux и в большинстве случаев осуществляются автоматически или при минимальном вмешательстве пользователя.

Тем не менее нередко возникают ситуации, когда пользователь должен выполнить определенные манипуляции с файловой системой вручную, например если требуется подключить или отключить файловую систему или же изменить параметры подключения для нее. Другой случай, весьма неприятный, – нарушение целостности файловой системы, в результате чего возникают ошибки при выполнении файловых операций, что требует восстановления файловой системы. Во всех подобных случаях понимание процессов, происходящих при манипулировании файловой системой, просто необходимо для правильных действий.

В следующем разделе мы рассмотрим более подробно подключение и отключение файловых систем, а также контроль их функционирования.

Подключение, отключение и восстановление файловых систем

Перед использованием файлов система Linux должна выполнить процедуру подключения, или, в терминах Linux, монтирования файловой системы. Жесткий диск может содержать один или несколько логических разделов, на которые он разбит дисковым драйвером, причем каждому разделу соответствует файл устройства с определенным именем. Процессы обращаются к данным раздела, открывая соответствующий файл устройства и затем выполняя запись и чтение из него.

Раздел диска может содержать логическую файловую систему, которой системная команда **mount** (монтировать) связывает файловую систему из указанного раздела на диске с существующей иерархией файловых систем. Команда **umount** (демонтировать) выполняет обратную операцию, выключая файловую систему из иерархии. Команда **mount** дает пользователям возможность обращаться к данным в дисковом разделе как к файловой системе, а не как к последовательности дисковых блоков. Все физические файловые системы, кроме корневой (/), считаются съемными в том смысле, что они могут быть как доступными, так и недоступными для пользователей.

Команда **mount** сообщает системе, что блочное устройство или удаленный ресурс доступен для пользователей в указанной точке монтирования, которая к моменту монтирования уже должна существовать. В этом случае точка монтирования становится кор-

невым каталогом вновь смонтированного устройства или ресурса. Таким образом, команда **mount** монтирует физическую файловую систему или ресурс к общей логической файловой системе.

Команда **mount** имеет следующий синтаксис:

```
mount [-v | -p]
mount [-F ФС] [-V] [-o опции]{устройство|точка_монтирования}
mount [-F ФС] [-V] [-o опции]{устройство точка_монтирования}
```

Команда **mount** при указанных во время вызова аргументах проверяет их, за исключением устройства устройство, и вызывает соответствующий модуль монтирования для указанного типа файловой системы. Вызванная без аргументов команда **mount** отображает список всех смонтированных файловых систем, считывая информацию из соответствующей таблицы.

Если указан неполный список аргументов (например, указаны только устройства или точки монтирования, или указаны оба этих аргумента, но не задан тип файловой системы), команда **mount** просматривает таблицу стандартных файловых систем в поисках недостающих аргументов, после чего вызывает соответствующий модуль монтирования для соответствующего типа файловой системы.

Обратная процедура по отношению к монтированию – демонтирование – выполняется командой **umount** со следующим синтаксисом:

```
umount [-V] [-o опции]{устройство|точка_монтирования}
```

В большинстве систем занятую файловую систему демонтировать невозможно. Для успешного демонтирования в ней не должно быть открытых файлов и выполняющихся процессов. Если демонтируемая файловая система содержит исполняемые файлы программ, они не должны быть запущены.

Команды **mount** и **umount** могут работать со следующими основными опциями:

- **-v** – дополнительно отображаются тип файловой системы и флаги. Поля точка_монтирования и устройство переставлены;
- **-p** – отображает список смонтированных файловых систем в формате таблицы смонтированных файловых систем;
- **-F** – задает тип файловой системы для монтирования. Тип файловой системы должен быть либо задан, либо определяться по таблице стандартных файловых систем в ходе монтирования;
- **-V** – отображает результирующую командную строку, но не выполняет команду. Командная строка генерируется с помощью

опций и аргументов, указанных пользователем, путем добавления к ним, при необходимости, информации, взятой из таблицы стандартных файловых систем;

- **-o** – задает специфические опции для указанного типа физической файловой системы.

Любой пользователь может вызывать команду **mount** для получения списка смонтированных файловых систем и ресурсов. Например:

```
pi@raspberrypi / $ mount
/dev/root      on /          type ext4
(rw,noatime,data=ordered)
devtmpfs       on /dev       type devtmpfs
(rw,relatime,size=216132k,nr_inodes=54033,mode=755)
tmpfs          on /run       type tmpfs
(rw,nosuid,noexec,relatime,size=44880k,mode=755)
tmpfs          on /run/lock  type tmpfs
(rw,nosuid,nodev,noexec,relatime,size=5120k)
proc           on /proc      type proc
(rw,nosuid,nodev,noexec,relatime)
sysfs          on /sys       type sysfs
(rw,nosuid,nodev,noexec,relatime)
tmpfs          on /run/shm   type tmpfs
(rw,nosuid,nodev,noexec,relatime,size=89740k)
devpts         on /dev/pts   type devpts
(rw,nosuid,noexec,relatime,gid=5,mode=620)
/dev/mmcblk0p1 on /boot      type vfat
(rw,relatime,fmask=0022,dmask=0022,codepage=cp437,
iocharset=ascii,shortname=mixed,errors=remount-ro)
```

Монтирование и демонтирование файловых систем разрешено только суперпользователю root.

Команда **mount** добавляет запись в таблицу смонтированных файловых систем, а **umount** удаляет запись из этой таблицы. Поля в таблице смонтированных устройств разделены пробелами и предоставляют информацию о:

- типе блочного специального устройства;
- точке монтирования;
- типе смонтированной файловой системы;
- опциях монтирования;
- времени, когда файловая система была смонтирована.

Для корректного выполнения операций монтирования/демонтирования необходима дополнительная информация, которая нахо-

дится в таблице стандартных файловых систем (файл /etc/fstab). В соответствующих полях этой таблицы, разделенных пробелами или символами табуляции, описаны стандартные параметры для физических файловых систем:

- специальное блочное устройство или имя монтируемого ресурса;
- неформатированное (специальное символьное) устройство для проверки утилитой **fsck**;
- стандартный каталог монтирования;
- тип файловой системы;
- числовой параметр, используемый командой **fsck** для принятия решения об автоматической проверке файловой системы и о порядке этой проверки по отношению к другим файловым системам;
- признак автоматического монтирования файловой системы;
- опции монтирования.

Вот как может выглядеть содержимое таблицы /etc/fstab, отображаемое с помощью команды cat:

```
pi@raspberrypi / $ cat /etc/fstab
proc            /proc  proc    defaults              0       0
/dev/mmcblk0p1 /boot   vfat    defaults              0       2
/dev/mmcblk0p2 /       ext4    defaults,noatime      0       1
# a swapfile is not a swap partition, so no using
swapon|off from here on, use   dphys-swapfile swap[on|off]
for that
```

Контроль дискового пространства

В процессе функционирования операционной системы UNIX любой пользователь рано или поздно сталкивается с проблемой нехватки дискового пространства. Даже если в системе установлены дисковые накопители большой емкости, наступает момент, когда места на дисках не хватает. Современное программное обеспечение использует, как правило, значительные ресурсы постоянной памяти, сохраняя при этом устойчивую тенденцию к дальнейшему увеличению такого потребления.

Для контроля используемых ресурсов файловой системы в UNIX предусмотрены специальные утилиты, такие как **du** и **df**.

Команда **df** позволяет получить информацию о смонтированных файловых системах. Она выдает отчет о доступном и использован-

ном дисковых пространствах на файловых системах. Синтаксис команды может быть представлен как

```
df [опции] [файл]
```

Здесь опции и параметры определяют формат отображаемой информации о файловых системах. При запуске без аргументов **df** выдает отчет по доступному и использованному пространству для всех смонтированных файловых систем (всех типов). Запущенная с опцией **-k** команда **df** отображает размеры файловых систем в килобайтах. В этом случае информация о каждой физической файловой системе отображается в отдельной строке и включает в себя:

- имя специального файла или смонтированного ресурса;
- общий и используемый объем дискового пространства;
- объем, доступный для использования обычными пользователями;
- процент свободного дискового пространства в файловой системе;
- точку монтирования.

Если в качестве параметра команды **df** задано имя файла, то отображается отчет по файловой системе, которая его содержит. При этом если параметр файл является дисковым файлом устройства, содержащим смонтированную файловую систему, то отображается доступное пространство для этой файловой системы, а не той, где содержится файл устройства. Команда **df** поддерживается стандартами POSIX и GNU, при этом некоторые параметры являются специфичными для каждого из стандартов.

Для команды **df** в стандарте POSIX все размеры выдаются в блоках по 512 байтов, но если задана опция **-k**, то используются блоки размером по 1024 байта. Формат вывода не стандартизован, кроме случая, когда используется опция **-P**. Кроме того, если файл является каталогом или именованным каналом (FIFO), то результат не определен.

Для команды **df** в стандарте GNU все размеры отображаются в блоках по 1024 байта (если размер блока не задан), за исключением случая, когда установлена переменная **POSIXLY_CORRECT**: тогда размер блока соответствует POSIX-версии этой команды. В стандарте GNU используются опции:

```
[-ahHiklmPv]
[-t тип_файловой_системы]
[-x тип_файловой_системы]
```

```
[--block-size=размер]
[--print-type]
[--no-sync]
[--sync]
[--help]
[--version]
```

Рассмотрим более подробно некоторые опции:

- **-k** – используется размер блока в 1024 байта вместо размера по умолчанию 512 байтов;
- **-P** – вывод осуществляется в шесть колонок с заголовком «Filesystem N-blocks Used Available Capacity Mounted on». Размер блока устанавливается по умолчанию (512 байтов) или 1024 байта при задании опции **-k**;
- **-a**, **--all** – включает в список вывода файловые системы, имеющие размер 0 блоков (не выводятся по умолчанию);
- **--block-size=размер** – отображает размер в блоках, размер в байтах которых задан;
- **-k**, **--kilobytes** – отображает выводимую информацию в блоках по 1024 байта;
- **-l**, **--local** – отображает данные только о локальных файловых системах;
- **-t тип_файловой_системы**, **--type=тип_файловой_системы** – показывает только файловые системы с указанным типом, при этом разрешается задавать несколько типов файловых систем с использованием нескольких опций **-t**. По умолчанию отображается информация обо всех файловых системах.

Рассмотрим несколько примеров использования команды **df**:

```
pi@raspberrypi/ $ df -k
Filesystem   1K-blocks     Used  Available   Use%   Mounted on
rootfs        14883856  3317320   10810416    24%   /
/dev/root     14883856  3317320   10810416    24%   /
devtmpfs        216132        0     216132     0%   /dev
tmpfs            44880      240      44640     1%   /run
tmpfs             5120        0       5120     0%   /run/lock
tmpfs            89740        0      89740     0%   /run/shm
/dev/mmcblk0p1   57288    18984      38304    34%   /boot
```

В этой команде используется опция **-k**, поэтому вывод на консоль выполняется в пересчете на однокилобайтовые блоки.

В следующем примере задается опция **-a**, поэтому на дисплей выводится список всех файловых систем, включающий специальные файловые системы:

```
pi@raspberrypi / $ df -a
Filesystem    1K-blocks     Used  Available  Use%   Mounted on
rootfs         14883856  3317320   10810416   24%   /
/dev/root      14883856  3317320   10810416   24%   /
devtmpfs         216132        0     216132    0%   /dev
tmpfs             44880      240      44640    1%   /run
tmpfs              5120        0       5120    0%   /run/lock
proc                  0        0          0    -    /proc
sysfs                 0        0          0         -/sys
tmpfs             89740        0      89740    0%   /run/shm
devpts                0        0          0    -    /dev/pts
/dev/mmcblk0p1    57288    18984      38304   34%   /boot
```

Заданная без опций, команда **df** может отображать информацию в таком виде, как показано в примере:

```
pi@raspberrypi / $ df
Filesystem    1K-blocks     Used  Available  Use%  Mounted on
rootfs         14883856  3317320   10810416   24%  /
/dev/root      14883856  3317320   10810416   24%  /
devtmpfs         216132        0     216132    0%  /dev
tmpfs             44880      240      44640    1%  /run
tmpfs              5120        0       5120    0%  /run/lock
tmpfs             89740        0      89740    0%  /run/shm
/dev/mmcblk0p1    57288    18984      38304   34%  /boot
```

Команда **du** позволяет вывести на консоль отчет об использовании дискового пространства заданными файлами, а также каждым каталогом в иерархии подкаталогов для каждого указанного каталога. Имейте в виду, что команда **du** показывает не фактический размер файла в байтах, а отображает только количество выделенных блоков или байтов на диске.

Размер блока зависит от настройки соответствующих параметров операционной системы и изменяется в широком диапазоне. Обычно используют блоки размером в 1 Кбайт, а минимальное дисковое пространство, выделяемое для файла и каталога, устанавливают равным 4 блока или 4 Кбайт (для блока размером в 1 Кбайт).

Такая группа блоков называется кластером. Если для файла выделено 12 Кбайт дискового пространства, то говорят, что он занимает 3 кластера. Размер кластера, в свою очередь, также может варьироваться (обычно он равен 4 блокам). Очень часто возникает ситуация, когда для файлов, имеющих разные размеры, выделяется одинаковое дисковое пространство. Например, для файлов с размерами

2 байта и 2 Кбайта будет выделено на жестком диске одинаковое пространство в 4 Кбайта, то есть 1 кластер.

Рассуждая о свободном дисковом пространстве, нужно учитывать тот факт, что операционная система располагает свободным дисковым пространством, пересчитанным на количество свободных кластеров. Это означает, что фактический размер файлов на диске и выделенное под них дисковое пространство никогда не совпадают в принципе, поэтому свободное дисковое пространство определяется не количеством свободных байтов, а количеством свободных кластеров.

Эту особенность следует учитывать при работе с файловой системой Linux. Именно поэтому размеры блока и кластера влияют на многие параметры функционирования операционной системы, особенно на ее быстродействие.

Следует учитывать, что операционная система работает с файлами быстрее, если используются кластеры больших размеров, однако в этом случае уменьшается свободное дисковое пространство, что необходимо учитывать при настройке UNIX.

Если команда **du** выполняется без аргументов, то на консоли отображается отчет о дисковом пространстве для текущего каталога. Команда поддерживает опции как стандарта POSIX, так и GNU. Вот некоторые опции стандарта POSIX и их смысл:

- **-a** – отображает размеры для всех файлов;
- **-k** – использует размер блока 1024 байта вместо 512 по умолчанию;
- **-s** – отображает информацию только для указанных явно аргументов, а не для их подкаталогов;
- **-x** – вычисляет размеры только для той файловой системы, где расположен объект, заданный параметром.

Команда **du** может принимать и опции стандарта GNU:

- **-a**, **--all** – показывает размеры для всех файлов;
- **-b**, **--bytes** – отображает размеры в байтах вместо килобайтов;
- **--block-size=количество** – отображает размеры в блоках размером количество байтов;
- **-c**, **--total** – отображается итоговая информация по всем параметрам после того, как они будут обработаны. Эта возможность может быть использована для подсчета суммарного использованного дискового пространства для всего списка заданных файлов и каталогов;
- **--exclude=шаблон** – при рекурсивном выполнении пропускает файлы или каталоги, чьи имена совпадают с заданным шаб-

лоном. Этот шаблон может быть любым файловым шаблоном bash;

- **-k**, **--kilobytes** – отображает размеры в килобайтах (1024 байта);
- **-m**, **--megabytes** – отображает размеры в мегабайтах (1 048 576 байтов);
- **-s**, **--summarize** – отображает только суммарный итог для каждого аргумента;
- **-S**, **--separate-dirs** – выдает размер каждого каталога в отдельности, не включая размеров подкаталогов;
- **-x**, **--one-file-system** – пропускает каталоги, находящиеся в другой файловой системе.

Размеры указываются в блоках по 1024 байта (если размер не задан посредством опций), за исключением случая, когда задана переменная окружения **POSIXLY_CORRECT**. Тогда размер блока соответствует версии POSIX.

Команда **du** без параметров показывает размер дискового пространства, занимаемого файлом или каталогом в целом, например:

```
pi@raspberrypi / $ du $HOME/developer/PRIMERS
20    /home/pi/developer/PRIMERS
```

Результат выдается в блоках, размер которых определяется Linux. В данном примере информация о занимаемом дисковом пространстве рассчитывается, исходя из размера блока в 1 Кбайт. Для определения объема дискового пространства при другом размере блока нужно использовать опцию **-B**:

```
pi@raspberrypi / $ du -B 512 $HOME/developer/PRIMERS
40      /home/pi/developer/PRIMERS
```

Здесь используется блок размером в 512 байтов. Для получения размера занимаемого дискового пространства в байтах можно выполнить команду **du** с опцией **-b**:

```
pi@raspberrypi / $ du -b $HOME/developer/PRIMERS
9981    /home/pi/developer/PRIMERS
```

Если команда **du** задана с опцией **-c**, то кроме информации по отдельным каталогам выдается суммарный размер используемого дискового пространства:

```
pi@raspberrypi / $ du -b -c $HOME/developer/PRIMERS
9981    /home/pi/developer/PRIMERS
9981    total
```

Перед подключением файловой системы необходимо, чтобы она существовала на физическом диске, либо самому ее создавать. Почти все реализации операционной системы Linux позволяют создавать файловые системы автоматически, без участия пользователя в процессе инсталляции. Тем не менее могут возникнуть ситуации, требующие создания файловых систем вручную, например при невозможности восстановления файловой системы после повреждения. Кроме того, знание, как создается файловая система, может помочь при анализе ее функционирования. Сейчас мы рассмотрим некоторые наиболее важные аспекты создания файловых систем.

В Linux-системах для создания файловых систем используется команда **mkfs**. Для создания файловой системы в командной строке указывается ее тип (ФС), а также целый ряд специальных опций и операндов. Вот синтаксис команды:

mkfs [-F ФС][-V][-m] [-o *опции*] *устройство размер* [*операнды*]

Задаваемые опции и операнды зависят от типа создаваемой файловой системы. Основные из них представлены далее:

- **-F** – определяет тип создаваемой файловой системы. Он задается либо в командной строке, либо берется из файла, содержащего таблицу стандартных файловых систем (/etc/fstab в Linux);
- **-V** – отображает введенную командную строку без выполнения самой команды. Напомню, что командная строка создается как при помощи опций, указанных пользователем, так и путем добавления информации, взятой из таблицы стандартных файловых систем. Эта опция позволяет проверить корректность командной строки;
- **-m** – отображает командную строку, использованную для создания файловой системы, при этом файловая система уже должна существовать. Эта опция неприменима с другими параметрами;
- **-o** – задает опции, специфические для указанного типа физической файловой системы, например специальное символьное устройство, на котором будет создана файловая система, или размер файловой системы в 512-байтовых блоках.

До сих пор мы рассматривали файловые системы в предположении, что они функционируют нормально. К сожалению, иногда файловые системы повреждаются. Это вызывается разными причинами, в числе которых можно назвать износ физического носи-

теля информации, неисправность электронных схем или, нередко, случайное разрушение логических структур файловой системы (суперблок, таблица индексных дескрипторов и т. д.) самим пользователем.

Проблема целостности (consistency) файловой системы является основой корректной работы операционной системы и ее надежности. В самой операционной системе UNIX предусмотрено несколько операций, обеспечивающих более надежную работу файловой системы. Так, например, во время записи файла на жесткий диск его индексный дескриптор и блоки устанавливаются в определенном порядке, при этом такую операцию называют «упорядочением записи».

Одним из способов повышения надежности является периодическая запись системных буферов на жесткий диск, известная как «автоматическая модификация». Кроме этого, все современные системы Linux в процессе работы выполняют определенные процедуры диагностирования.

Тем не менее целиком полагаться на средства проверки операционной системы не стоит. Будет лучше, если пользователь протестирует файловую систему при возникновении сомнений в ее корректной работе. Повысить надежность работы файловой системы можно и в том случае, если соблюдать некоторые простые правила. Вот они:

- при выключении компьютера используйте стандартные для этого случая команды операционной системы, например **shutdown**. Это обеспечивает корректное закрытие всех файлов и демонтирование всех файловых систем;
- наиболее важные данные записывайте на альтернативный носитель, например CD;
- время от времени проверяйте целостность файловой системы с помощью встроенных программных средств, например утилиты **fsck**.

Программа **fsck** включена во все версии операционных систем и очень полезна при проверке целостности файловых систем. При подозрении, что что-то не в порядке с файловой системой, всегда используйте **fsck**. Утилита позволяет найти и исправить повреждения (inconsistency) в файловых системах, причем допускает два режима работы – автоматический или интерактивный.

При значительных повреждениях файловой системы программу fsck лучше запускать в интерактивном режиме – при этом пользователь должен подтверждать каждый свой шаг, что вынуждает задумываться, прежде чем что-то сделать. Имейте в виду, что некоторые

исправления могут привести к определенным потерям данных, что отображается в диагностических сообщениях.

В интерактивном режиме перед каждым исправлением программа **fsck** ожидает ответа от пользователя – «Да» (YES) или «Нет» (NO). Если при запуске утилиты указать параметр -y, fsck допускает ответ «Да» и не делает паузы для ответа. Замечу, что программа **fsck** – многопроходная команда контроля файловых систем, поэтому на каждом проходе файловой системы выполняются различные этапы: проверка блоков и размеров файлов, полных имен файлов, связности, карты свободных блоков (возможно, с ее перестройкой), подсчет ссылок и т. д.

Перед запуском утилиты **fsck** файловая система должна быть демонтирована. Кроме того, если исправления вносятся в критическую файловую систему, например в корневую, то после **fsck** систему необходимо перезагрузить.

Права доступа к файлам

Все объекты файловой системы имеют те или иные атрибуты доступа, позволяющие или запрещающие выполнять те или иные операции как отдельным пользователям, так и группам пользователей.

Для каждого файла устанавливается определенная комбинация прав доступа, представляющая собой набор из девяти битов. Она определяет, например, кто имеет право читать содержимое файла, записывать в него данные или выполнять его (если это исполняемый файл). Биты этого набора вместе с другими тремя битами, определяющими способ запуска исполняемых файлов, образуют код режима доступа к файлу. Все двенадцать битов режима хранятся в 16-битовом поле индексного дескриптора вместе с четырьмя дополнительными битами, указывающими тип файла.

Четыре последних бита устанавливаются в момент создания файла и не должны изменяться. Биты доступа может изменить либо владелец файла, либо суперпользователь root при помощи команды **chmod**. Эти биты вместе с остальной информацией отображаются на экране командой **ls**.

В коде режима доступа есть биты специального назначения, которым соответствуют восьмеричные значения 4000 и 2000. Это биты смены идентификатора пользователя (SUID) и смены идентификатора группы (SGID), позволяющие программам пользователя полу-

чать доступ к файлам и процессам, которые в иных обстоятельствах недоступны пользователю.

Бит, которому в коде режима доступа соответствует восьмеричное значение 1000, называется sticky-битом. В ранних версиях операционной системы UNIX с небольшим объемом памяти, когда требовалось, чтобы отдельные программы постоянно оставались в памяти, sticky-бит был очень важен – он запрещал выгрузку программ из памяти. В настоящее время, когда повсеместно применяются недорогие и емкие модули памяти и быстродействующие дисковые накопители, sticky-бит уже не нужен, поэтому современные UNIX-системы его игнорируют.

Особенностью Linux является то, что нельзя устанавливать биты прав доступа отдельно для каждого пользователя. Вместо этого используются разные трехбитовые комбинации (триады) для владельца файла, группы, которой принадлежит файл, и прочих пользователей. Каждая триада состоит из бита чтения, бита записи и бита выполнения (для каталога последний называется битом поиска).

Код режима доступа часто представляют в виде восьмеричного числа, так как каждая цифра в нем выражается тремя битами со следующими значениями:

- три старших бита (восьмеричные значения 400, 200 и 100) определяют права доступа к файлу со стороны его владельца;
- три средних бита (восьмеричные значения 40, 20 и 10) задают доступ для пользователей группы;
- младшие три бита (восьмеричные значения 4, 2 и 1) определяют права доступа к файлу для остальных пользователей.

При этом старший бит каждой триады определяет доступ по чтению, средний – доступ по записи и, наконец, младший бит определяет права на выполнение.

Каждый пользователь включается только в одну из категорий, соответствующих одной из триад битов режима, при этом из получившихся комбинаций выбираются те, которые гарантируют самые строгие права доступа. Так, например, права доступа для владельца файла всегда определяются триадой битов владельца и никогда – битами группы. При этом вполне возможна ситуация, когда другие пользователи наделены бóльшими правами доступа к файлу, чем владелец, но подобные конфигурации используются довольно редко.

Для текстового файла, например, установленный бит чтения разрешает читать данные, а бит записи – изменять данные. Удалить или переименовать файл можно в том случае, если установлены соответствующие биты прав доступа для каталога, где хранится имя файла.

Бит выполнения определяет возможность выполнить файл как программу или командный сценарий. Программа представляет собой исполняемый файл, содержащий двоичные данные, интерпретируемые как команды процессора и выполняемые непосредственно центральным процессором. Командный сценарий или командный файл обрабатывается интерпретатором shell или какой-нибудь другой программой (например, sed или Python).

Для каталогов интерпретация бита выполнения несколько иная, чем для файла. Если для каталога установлен только бит выполнения, то разрешается вход в него, но получить список его содержимого нельзя. Содержимое каталога можно просмотреть в том случае, если установлены биты чтения и выполнения. Установленные биты записи и выполнения позволяют создавать, удалять и переименовывать файлы в данном каталоге.

Права доступа к файлам и каталогам можно установить при помощи команд **chmod** и **chown**, причем выполнять подобные действия может либо владелец файла, либо суперпользователь root.

Напомню, что команда **chmod** устанавливает права доступа к указанным файлам и имеет следующий синтаксис:

```
chmod [-fR] абсолютные_права файл ...
chmod [-fR] символьные_права файл ...
```

Первым аргументом команды **chmod** является спецификация прав доступа. Второй и последующий аргументы задают имена файлов, а также права доступа, подлежащие изменению.

Изначально код доступа в операционных системах UNIX задавался в виде восьмеричного числа, хотя в современных версиях поддерживается и более наглядная система мнемонических обозначений. Восьмеричное представление более удобно для системных администраторов, хотя при этом можно задать только абсолютное значение режима доступа. Преимуществом мнемонического обозначения прав доступа является то, что можно сбрасывать и устанавливать отдельные биты режима.

При использовании восьмеричной нотации первая цифра относится к владельцу, вторая – к группе, а третья – к остальным пользователям. Если необходимо задать биты SUID/SGID или sticky-бит, следует указывать не три, а четыре восьмеричные цифры. Первая цифра в этом случае будет соответствовать трем специальным битам.

Опция **-f** команды отключает вывод сообщения о невозможности установки прав доступа. При помощи опции **-R** можно установить

или изменить права доступа рекурсивно для всех подкаталогов, указанных в списке файлов.

Абсолютные права доступа, указанные опцией **абсолютные_права**, задаются восьмеричным числом, в то время как символьные права доступа, определяемые опцией **символьные_права**, указываются в виде списка выражений (через запятую), например:

```
[пользователи] оператор [права, ...]
```

Элемент списка пользователи определяет, для кого задаются или изменяются права. Он может принимать значения **u**, **g**, **o** и **a**, относящиеся к владельцу, группе, остальным пользователям, а также ко всем категориям пользователей соответственно. Более подробно символ **u** (user) обозначает владельца файла, символ **g** (group) – группу, символ **o** (others) – других пользователей, символ **a** (all) – всех пользователей сразу.

```
chmod [-fR] абсолютные_права файл ...
chmod [-fR] символьные_права файл ...
```

Первым аргументом команды **chmod** является спецификация прав доступа. Второй и последующий аргументы задают имена файлов, а также права доступа, подлежащие изменению. Изначально код доступа в операционных системах UNIX задавался в виде восьмеричного числа, хотя в современных версиях поддерживается и более наглядная система мнемонических обозначений. Восьмеричное представление более удобно для системных администраторов, хотя при этом можно задать только абсолютное значение режима доступа. Преимуществом мнемонического обозначения прав доступа является то, что можно сбрасывать и устанавливать отдельные биты режима.

Таблица 3.2. Коды прав доступа в команде chmod

Восьмеричное число	Двоичное число	Маска режима доступа
0	000	
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Например, команда

```
$ chmod 711 myfile
```

предоставляет владельцу все права доступа к файлу myfile и право на выполнение – остальным пользователям. Мнемонические спецификации представлены в табл. 3.3.

Таблица 3.3. Примеры мнемонических спецификаций команды chmod

Спецификация	Описание
u+w	Владельцу файла дополнительно разрешается выполнять файл
ug=rw,o=r	Владелец и группа получают право на чтение/запись, остальные пользователи – право чтения
a-x	Все пользователи лишаются права выполнения
ug=srx,o=	Владелец и группа получают право чтения/выполнения, и устанавливается бит SUID; доступ к файлу остальным пользователям запрещен
g=u	Группе назначаются такие же права, что и владельцу

Рассмотрим пример изменения прав доступа. Предположим, что в текущем каталоге имеется файл test, для которого мы будем изменять права доступа, как показано далее, одновременно наблюдая за результатами:

```
$ sudo chmod +w test
$ ls -l test
-rw-r--r--    1 yury 50                  0 Dec 11 2:05 test
$ sudo chmod a+w test
$ ls -l test
-rw-rw-rw-    1 yury 50                  0 Dec 11 2:10 test
$ sudo chmod u+x,g=x,o= test
$ ls -l test
-rwx--x---    1 yury 50                  0 Dec 11 2:13 test
$ sudo chmod ug-x,og+r,u=rwx test
$ ls -l test
-rwxr--r--    1 yury 50                  0 Dec 11 2:14 test
$ sudo chmod 644 test
$ ls -l test
-rw-r--r--    1 yury 50                  0 Dec 11 2:15 test
```

При создании нового файла ему присваиваются права доступа, определяемые пользовательской маской режима создания файлов. Встроенная команда **umask** интерпретатора shell присваивает поль-

зовательской маске режима создания файлов указанное восьмеричное значение. Три восьмеричные цифры соответствуют правам на чтение/запись/выполнение для владельца, членов группы и прочих пользователей соответственно.

Команда имеет следующий синтаксис:

```
umask [-S] [маска]
```

Выполненная без параметров, команда **umask** отображает текущее значение маски. По умолчанию значение маски задается и отображается в восьмеричном виде как число. Маска используется при указании прав доступа следующим образом: это значение необходимо «вычесть» из максимальных прав доступа (777 для выполняемых файлов, которые создаются компиляторами, и 666 для обычных файлов). Вот пример:

```
$ umask
022
```

При данном значении маски обычные текстовые файлы будут создаваться с правами 666 – 022 = 644, как это показано на примере создания файла test:

```
$ > test
$ ls -l test
-rw-r--r--    1 yury 50                    0 Nov 12 1:39 test
```

Операция «вычитания» для значения маски формально выполняется как побитовое логическое И дополнения маски и максимальных прав доступа. Для нашего примера расчет прав доступа посредством маски показан в табл. 3.4.

Таблица 3.4. Расчет маски

Значение	Результат
Двоичное значение маски	000010010 (022)
Дополнение маски	111101101 (755)
Максимальное значение прав	110110110 (666)
Логическое И предыдущих двух строк	110100100 (644)
Результирующие биты прав	110100100 (644)

Опция **-S** требует отображать значение маски в символьном виде. В этом случае отображаются биты прав доступа у вновь создаваемого файла. При этом берется во внимание и бит выполнения:

```
$ umask -S
u=rwx,g=rx,o=rx
```

Команду **umask** нередко включают в файлы начального запуска, устанавливающие рабочую среду для начального командного интерпретатора.

Вот еще один пример создания файла, но при другом значении маски (257):

```
$ umask 257
$ > test
$ ls -l test
-r---w----    1 yury 50                0 Nov 12 10:34 test
```

Владелец файла, а также суперпользователь root могут изменить владельца и группу-владельца файла, используя команду **chown**.

Команда имеет такой синтаксис:

```
chown [-h][-R] владелец[:группа] файл ...
```

Если нужно изменить только группу, владеющую файлом, можно использовать команду **chgrp**:

```
chgrp [-h][-R] группа файл
```

В качестве первого аргумента обеих команд используется имя нового владельца или новой группы соответственно. Кроме того, для выполнения команды **chgrp** необходимо быть владельцем файла и входить в назначаемую группу или иметь права суперпользователя root.

Опция **-h** требует изменить владельца файла, на который указывает символическая ссылка, а не саму ссылку, как это принято по умолчанию.

В большинстве версий команд **chown** и **chgrp** предусмотрен флаг **-R**, который задает смену владельца или группы не только самого каталога, но и всех его подкаталогов и файлов. Например, последовательность команд

```
chmod 755 -red
chown -R red -red
chgrp -R color -red
```

можно использовать для конфигурирования начального каталога нового пользователя после копирования в него стандартных фай-

лов сценариев. Не следует пытаться выполнять команду **chown** для файлов, имена которых начинаются с точки:

```
$ sudo chown -R red -red/.*
```

Указанному шаблону поиска соответствует также файл ~red/.., вследствие чего команда изменит и владельца родительского каталога.

В операционных системах UNIX, совместимых с System V, пользователи могут беспрепятственно поменять владельца собственного файла при помощи команды chown, тогда как в BSD-совместимых системах эту команду может выполнять только суперпользователь root.

Необходимо учитывать, что после смены владельца файла бывший владелец будет иметь права доступа, установленные новым владельцем. Пусть требуется сменить владельца файла test, атрибуты которого показаны далее:

```
$ ls -l
total 2
-rw-r--r--      1 user1    others        6 Dec 10 16:19 test
```

Из результата выполнения команды **ls** видно, что владельцем файла test является пользователь user1, в сеансе которого и выполняются команды. Сменим владельца файла на yury:

```
$ sudo chown yury test
$ ls -l
total 2
-rw-r--r--      1 yury others            6 Dec 10 16:19 test
```

После выполнения команды **chown** видно, что пользователем файла test является yury. Если теперь в сеансе пользователя user1 попробовать заменить владельца опять на user1, то будет выдана ошибка:

```
$ sudo chown user1 test
UX:chown: ERROR: testf: Not privileged
```

Помимо рассмотренных нами атрибутов доступа, объекты файловой системы операционной системы UNIX имеют и другие атрибуты. Рассмотрим их более подробно. Напомню, что информация о каждом файле хранится в структуре, называемой индексным дескриптором.

Индексный дескриптор содержит порядка сорока информационных полей, но большинство из них используется только ядром. Поль-

зователя операционной системы интересует, как правило, только небольшая часть информации, находящейся в индексных дескрипторах:

- количество жестких ссылок;
- владелец файла;
- группа-владелец файла;
- код прав доступа;
- размер файла;
- время последнего обращения;
- время последней модификации;
- тип файла.

Всю эту информацию можно легко получить с помощью команды `ls -l`. Рассмотрим пример:

```
$ ls -l /bin/sh
-rwxr-xr-x   1    root   bin   85924 Sep 27   2012 /bin/sh
```

Здесь в первом поле задаются тип файла и маска режима доступа к нему. Первый символ – дефис, следовательно, это обычный файл. Обозначения различных типов файлов представлены односимвольными кодами (табл. 3.5).

Таблица 3.5. Кодирование типов файлов в команде ls -l

Тип файла	Символ	Создается командой	Удаляется командой
Обычный файл	–	Редакторы, cp и др.	–
Каталог	d	mkdir	rmdir, rm -r
Файл байт-ориентированного устройства	c	mknod	rm

Следующие девять символов в этом поле представляют три набора (триады) битов режима. В выводе команды `ls` они представляются литерами `r`, `w` и `x` (чтение, запись и выполнение соответственно). В данном примере владелец обладает полным доступом к файлу, а остальные пользователи – только правом на чтение и выполнение.

Если бы был установлен бит смены идентификатора пользователя (SUID), то вместо `x` стоял бы символ `s`. Аналогично если бы был установлен бит смены идентификатора группы (SGID), то вместо `x` для группы стоял бы символ `s`. Последний бит режима (право выполнения для остальных пользователей) представляется буквой `t`, когда для файла задан sticky-бит. Если биты SUID/SGID или sticky-бит установлены, а надлежащий бит выполнения – нет, эти

биты представляются, соответственно, символами, указывающими на наличие ошибки и игнорирование данных атрибутов.

Следующее поле содержит счетчик ссылок на файл. На этом месте стоит единица, а это значит, что /bin/sh – единственное имя, под которым известен данный файл. Всякий раз при создании жесткой ссылки на файл этот счетчик увеличивается на единицу. Что же касается каталогов, то любой из них имеет как минимум две жесткие ссылки: одну – из родительского каталога и одну – из специального файла внутри самого каталога. Следует отметить, что символические ссылки в счетчике не учитываются.

Следующие два поля – владелец и группа-владелец файла. В данном случае владельцем файла является суперпользователь root, а файл принадлежит группе bin. Ядро операционной системы хранит эти данные не в виде строк, а как идентификаторы пользователя и группы. Если невозможно получить символьное представление имен, в этих полях будут отображаться числа. Такое случается, если запись пользователя или группы была удалена из файла /etc/passwd или /etc/group соответственно.

Далее следует поле, отображающее размер файла в байтах: данный файл имеет размер 85 924 байта, то есть почти 84 Кбайта. Следующее поле содержит дату последнего изменения: 27 сентября 2012 г., и, наконец, в последнем поле вывода содержится имя файла: /bin/sh.

Если мы имеем дело с файлом устройства, то вывод команды **ls** будет другим, например:

```
$ ls -l /dev/ttya
crw-rw-rw-   1    root   daemon 12, 0   Dec 20 2011 /dev/ttya
```

Результат работы этой команды иной, чем в предыдущем примере: вместо размера в байтах показаны старший и младший номера устройства. Имя /dev/ttya относится к первому устройству, управляемому драйвером устройства 12 (в данной системе это драйвер терминала).

При поиске жестких ссылок часто бывает полезной команда **ls -i**, отображающая для каждого файла номер индексного дескриптора. Жесткие ссылки, указывающие на один и тот же файл, будут иметь один и тот же номер.

Операционная система автоматически отслеживает такие атрибуты, как время изменения, число ссылок и размер файла, автоматически устанавливая корректные значения. В то же время права доступа и идентификаторы принадлежности файла могут быть модифицированы явным образом с помощью команд **chmod**, **chown** и **chgrp**.

Операции с файлами

Значительная часть времени пользователя уходит на операции с файлами, такие как копирование, перемещение, удаление и поиск. Далее мы проанализируем, какие средства предлагает операционная система UNIX для выполнения таких задач. Начнем с копирования и перемещения файлов.

Копирование файлов

Для копирования объектов файловой системы используется команда **cp**. С ее помощью копируются файлы или каталоги, причем можно либо копировать один файл в другой, заданный файл, либо копировать группу файлов в заданный каталог. Синтаксис команды таков:

```
cp [опции] файл путь
cp [опции] файл... каталог
```

Остановимся более детально на специфике работы команды **cp**. Если последним параметром является существующий каталог, то **cp** копирует исходные файлы в этот каталог, сохраняя при этом их имена. Если в качестве параметров заданы два файла, то **cp** копирует первый файл во второй.

В том случае, если командная строка содержит более двух параметров, не являющихся опциями самой команды, а последний параметр не является именем какого-либо каталога, то возникает ошибка.

Например, если /a – каталог, то команда

```
$ cp -r /a /b
```

скопирует /a в /b/a и /a/x в /b/a/x в том случае, если каталог /b уже существует. Если /b не существует, то эта команда создаст его и скопирует /a в /b и /a/x в /b/x. Наконец, если /b является обычным файлом, то команда завершится с ошибкой.

Права доступа к скопированным файлам и каталогам вычисляются путем логического умножения (операция И) кода доступа исходных файлов на 0777, а также с учетом **umask** пользователя (за исключением случая, когда задана опция **-p**). Если выполняется рекурсивное копирование каталогов, то вновь создаваемые каталоги временно получают права доступа, вычисленные путем логического сложения (операция ИЛИ) исходного кода доступа со значением

S_IRWXU (0700), разрешая тем самым чтение, запись и поиск во вновь созданных каталогах.

Попытка скопировать файл сам в себя ни к чему не приводит, кроме того что выдается сообщение об ошибке.

Удаление файлов

Для удаления файла с помощью команды **cp** необходимо указать опцию **-f**.

Команда поддерживает как опции стандарта POSIX, так и опции GNU. Вот опции POSIX:

- **-f** – удаляет существующие файлы;
- **-I** – выдает запрос относительно перезаписи существующих файлов, в которые происходит копирование. Копирование выполняется только в случае положительного ответа;
- **-p** – сохраняет исходные атрибуты файла, такие как владелец, группа, права доступа (включая SUID- и SGID-биты), время последней модификации и время последнего доступа к файлу. Если попытка установки владельца или группы приводит к ошибке, биты SUID и SGID сбрасываются;
- **-R** – выполняет рекурсивное копирование каталогов, отслеживая ситуации, когда встречаются объекты, не являющиеся обычными файлами или каталогами, например именованные каналы (FIFO) или специальные файлы. В подобных случаях создаются корректные копии таких объектов;
- **-r** – выполняет рекурсивное копирование каталогов, производя при этом не определенные стандартом действия, если встречаются объекты, не являющиеся обычными файлами или каталогами.

В стандарт POSIX 1003.1–2003 добавлены три опции, определяющие способы обработки символических ссылок:

- при выполнении нерекурсивного копирования символические ссылки разыменовываются;
- при выполнении рекурсивного копирования с опцией **-r** результаты зависят от реализации;
- при выполнении рекурсивного копирования с опцией **-R** необходимо принимать во внимание дополнительные опции:
  - **-H** – разыменовываются символические ссылки, указанные в списке параметров. В то же время символические ссылки, которые встречаются во время рекурсивного копирования, копируются;

- ✧ **-L** – разыменовываются все символические ссылки, как заданные в списке параметров, так и встретившиеся при выполнении рекурсивного копирования;
- ✧ **-P** – не разыменовываются никакие символические ссылки, как те, что заданы в списке параметров, так и те, которые встречаются во время рекурсивного копирования. В этом случае выполняется обычное копирование символических ссылок.

В последующих примерах применения команды **cp** мы будем придерживаться стандарта POSIX.

Вот некоторые примеры использования команды **cp**. Для копирования одного каталога в другой можно выполнить команду:

```
$ cp -r DIR ARCH_DIR
```

Здесь каталог DIR вместе со своим содержимым копируется в каталог ARCH_DIR.

В следующем примере команда

```
$ cp -r USR1 USR2 ARCH
```

копирует содержимое каталогов USR1 и USR2 в каталог ARCH.

Перемещение файлов

Остановимся на операции перемещения файлов. Перемещение файлов в операционной системе Linux выполняется с помощью команды mv, имеющей следующий синтаксис:

```
mv [опции...] исходный_файл файл_назначения
mv [опции...] исходный_файл... каталог
```

Если последний аргумент является именем существующего каталога, то **mv** перемещает указанные файлы в этот каталог. В том случае, если заданы только два файла, то имя первого файла будет изменено на имя второго. Наконец, если последний аргумент не является каталогом и задано более чем два файла, то будет выдано сообщение об ошибке.

Так, команда

```
$ mv /a/x/y /b
```

переименует файл /a/x/y в /b/y, если /b является существующим каталогом, и в /b, если нет.

Если при переименовании исходного_файла в файл_назначения последний существует (при заданной опции **-i**) или если выполнить запись в него невозможно, а стандартным выводом является терминал, и не задана опция **-f**, то **mv** запрашивает у пользователя разрешение на замену этого файла. Если ответ отрицательный, то файл пропускается.

Когда исходный_файл и файл_назначения находятся в одной файловой системе, то изменяется имя файла, а владелец, права доступа, временные штампы остаются неизменными. Если же они находятся в разных файловых системах, то исходный_файл копируется и затем удаляется. Во время выполнения операции команда **mv**, если это возможно, будет копировать время последней модификации, время доступа, идентификаторы пользователя и группы и права доступа к файлу. Если копирование идентификаторов пользователя и/или группы закончилось неудачно, то биты SETUID и SETGID скопированного файла сбрасываются.

Поддерживаются следующие опции POSIX:

- **-f** – запрос на подтверждение операции не выдается;
- **-I** – выдается запрос на подтверждение операции, когда файл_назначения существует. Если заданы обе опции **-f** и **-i**, используется только последняя из них.

Вот примеры использования команды **mv**:

```
$ mv qt qt.OLD
```

Здесь файл test переименовывается в файл test.old.

К недостаткам команды **mv** следует отнести то, что она не работает с шаблонами файлов приемника и источника. Если бы потребовалось, например, чтобы все файлы, соответствующие шаблону t*, были переименованы в файлы t*.old, то возникла бы ошибка:

```
$ mv t* t*.old

mv: when moving multiple files, last argument must be a
directory
Try 'mv --help' for more information.
```

Создание каталогов

Команды, которые мы рассматривали до сих пор, в основном работают с обычными файлами. Для управления каталогами в системе UNIX имеются две команды – **mkdir** и **rmdir**.

Новый каталог можно создать с помощью команды **mkdir**. В самой простой форме эта команда использует один параметр – имя каталога – и создает каталог с этим именем.

Опытные пользователи с помощью одной команды **mkdir** могут создавать несколько каталогов сразу, перечисляя их в одной командной строке. Синтаксис команды таков:

```
$ mkdir [опции] [список_каталогов]
```

Вместе с командой **mkdir** можно использовать две опции. Опция **-m** позволяет задать в восьмеричной или символьной форме права доступа (как и для команды **chmod**), которые будут присвоены создаваемым каталогам.

При использовании ключа **-p**, кроме указанного каталога, будут созданы также и любые требуемые промежуточные каталоги. При этом если у пользователя нет прав на запись в родительский каталог, то новый каталог не будет создан. Если каталог уже существует или вместо каталога существует файл с таким же именем, то будет выведено сообщение об ошибке.

Удаление каталогов

Для удаления файлов и каталогов в операционной системе Linux используются команды **rm** и **rmdir**. С помощью команды **rmdir** проще удалить одиночный каталог, причем он должен быть пустым. Если в каталоге имеются элементы, отличные от `.` и `..`, то команда **rmdir** такой каталог не удаляет. Синтаксис этой команды таков:

```
rmdir [-p][-s] каталог
```

Команда **rmdir** имеет две опции:

- **-p** – позволяет удалить пустой каталог вместе с его родительскими каталогами, отображая сообщение об успешном или неуспешном выполнении операции;
- **-s** – подавляет выдачу сообщений при использовании опции **-p**.

Команда **rm** работает несколько иначе. С ее помощью можно удалить указанные файлы, при этом каталоги по умолчанию не удаляются.

Если для команды заданы опции **-r** или **-R**, то будет удаляться все дерево каталогов ниже заданного каталога, включая и сам каталог, причем не накладывается никаких ограничений на глубину дерева. Если последний компонент файла – это `.` или `..`, то выдается со-

общение об ошибке (это сделано для того, чтобы избежать неприятных сюрпризов при выполнении команды **rm -r .*** или ей подобных).

Команда **rm** поддерживает стандарты POSIX и GNU. Вот опции POSIX:

- **-f** – не запрашивается подтверждение операции и не выдаются диагностические сообщения. При завершении команды с ошибками код ошибки не возвращается, если ошибки вызваны отсутствием файлов;
- **-I** – выводится запрос на подтверждение удаления (при указании опций **-f** и **-i** одновременно используется последняя);
- **-r** или **-R** – позволяет рекурсивно удалять дерево каталогов.

Следует учитывать потенциальную опасность использования команды **rm**, поскольку после ее выполнения восстановить удаленные файлы невозможно. Вот некоторые примеры неправильного использования команды **rm**.

Например, командная строка

```
$ rm * /tmp
```

должна была по замыслу удалить все файлы в каталоге /tmp. На самом деле были удалены сначала все файлы в текущем каталоге, а затем осуществлена попытка удалить /tmp, которая закончилась неудачно, поскольку tmp – это каталог.

Правильно следовало бы ввести команду

```
$ rm /tmp/*
```

Поиск файлов и каталогов

Поиск файлов и каталогов также относится к довольно часто выполняемым операциям. Поиск файлов и каталогов, удовлетворяющих определенным критериям, как известно, выполняется стандартными средствами операционной системы. Файловая система Linux содержит тысячи файлов, поэтому для быстрого поиска нужны очень эффективные средства, одним из которых является команда **find** с соответствующими опциями, представляющая собой очень мощный инструмент для выполнения подобных операций.

Команда имеет следующий синтаксис:

```
find каталог ... выражение
```

Утилита просматривает иерархии каталогов в поисках файлов, удовлетворяющих критерию, задаваемому выражением выражение.

Выражения строятся из элементов с помощью следующих конструкций:

- **-name шаблон** – условие истинно, если имя файла соответствует шаблону. При использовании метасимволов необходимо маскировать шаблоны от командного интерпретатора;
- **-type тип** – условие истинно, если файл имеет указанный тип. Типы файлов задаются символами **b**, **c**, **d**, **f**, **l**, **p** и **s**, обозначающими, соответственно, специальное блочное устройство, специальное символьное устройство, каталог, обычный файл, символическую ссылку, именованный канал и сокет;
- **-user пользователь** – условие истинно, если файл принадлежит пользователю, указанному по идентификатору или регистрационному имени;
- **-group группа** – условие истинно, если файл принадлежит группе, указанной по идентификатору или имени;
- **-perm [-] права** – если дефис не задан, то условие истинно, только если права доступа в точности соответствуют указанным (как в команде **chmod**). Если задан дефис, то условие истинно, если в правах доступа файла, как минимум, установлены те же биты, что и в указанных правах;
- **-size [+|-|=]n[c]** – условие истинно, если файл имеет длину n блоков (блок – 512 байтов) или символов (если указан суффикс c). Перед размером можно указывать префикс **+** (не меньше), **-** (не больше) или **=** (в точности равен);
- **-atime [+|-|=]n** – условие истинно, если к файлу последний раз обращались **n** дней назад. Перед **n** в элементах **-atime**, **-ctime** и **-mtime** можно указывать префикс **+** (не позже), **-** (не ранее) или **=** (ровно);
- **-ctime n** – условие истинно, если файл создан **n** дней назад;
- **-mtime n** – условие истинно, если файл был изменен **n** дней назад;
- **-newer файл** – условие истинно, если файл является более новым, чем указанный;
- **-ls** – условие истинно всегда (выдает информацию о файле, аналогичную длинному листингу);
- **-print** – условие истинно всегда (выдает полное имя файла в стандартный выходной поток);
- **-exec** команда **{} \;** – условие истинно, если выполненная команда имеет код возврата 0. Команда заканчивается замаскированной точкой с запятой. В команде можно использовать

конструкцию **{}**, заменяемую полным именем рассматриваемого файла;

- **-ok** команда **{} \;** – аналогично exec, но полученная после подстановки имени файла вместо **{}** команда выдается с вопросительным знаком и выполняется, если пользователь ввел символ **y**;
- **-depth** – условие истинно всегда – требует так обходить иерархию каталогов, чтобы файлы любого каталога всегда обрабатывались раньше, чем сам каталог (обход «в глубину»);
- **-prune** – условие истинно всегда – требует не проверять файлы в каталоге, путевое имя которого присутствует в предыдущем выражении. Не действует, если ранее указан элемент **-depth**.

В различных версиях операционной системы Linux могут поддерживаться и другие компоненты выражений в команде **find**. Следует сказать, что если командная строка сформирована неправильно, команда немедленно завершает работу.

Рассмотрим несколько примеров использования команды **find**.

Вначале – простой пример поиска файлов в каком-либо каталоге и вывод содержимого на консоль. Например, чтобы получить содержимое рабочего каталога программы (в нашем примере это /home/yury), достаточно выполнить команду

```
$ find `pwd` -print
```

На экране дисплея будет отображено содержимое текущего каталога:

```
/home/yury
/home/yury/developer
/home/yury/developer/test.c
```

Для получения содержимого другого каталога, например /home/yury/developer, нужно выполнить команду

```
$ find /home/yury/developer -print
/home/yury/developer
/home/yury/developer/MISC
/home/yury/developer/MISC/test
/home/yury/developer/MISC/misc.c
/home/yury/developer/MISC/stat_name.c
```

Следующая команда выводит на консоль содержимое текущего каталога, откуда была запущена команда **find**:

```
$ find . -print
.
./developer
./developer/test.c
```

Вызванная с параметрами, указанными далее, команда **find** ничего не выдает, поскольку файла с именем tmp в текущем каталоге нет:

```
$ find . -name tmp -print
```

Следующий пример отличается от предыдущего шаблоном имени файла. Пусть необходимо найти файлы, чьи имена заканчиваются на tmp. Если текущий каталог содержит, например, такие файлы:

```
$ ls -l
total 3
-rw-r--r--  1 YURY  Отсутств   5   Feb  3 18:08 test1
-rw-r--r--  1 YURY  Отсутств   199 Feb  3 18:08 test1.c
-rw-r--r--  1 YURY  Отсутств   9   Feb  3 18:09 test3.tmp
```

то поиск легко выполнить при помощи командной строки, указанной далее (здесь же выводится и результат поиска):

```
$ find . -name '*tmp' -print
./test3.tmp
```

В следующем примере с помощью команды **find** выполняется поиск файлов с расширением tmp или c, находящихся в текущем каталоге:

```
$ find . \( -name '*.tmp' -o -name '*.c' \) -print
./test1.c
./test3.tmp
```

В команде **find** можно задавать временны́е критерии поиска файлов, причем в самых различных комбинациях. Следующий пример демонстрирует это: в нем используется опция **-atime [+|-|=]n**. Условие является истинным, если время последнего доступа к файлу больше/меньше, чем n*24. Например, команда

```
$ find . \( -name '*.tmp' -o -name '*.pl' \) -atime +3 -print
```

выполняет поиск файлов с указанными шаблонами, к которым не было обращения больше трех суток.

Нередко требуется найти файлы, принадлежащие определенному пользователю. Например, следующая команда выполняет поиск

файлов в каталоге DIR, владельцем которых является суперпользователь root:

```
$ find DIR -user root –print
```

Если критерием поиска является размер файла, то можно использовать следующую опцию: **-size [+|-|=]n[c]**. Условие, задаваемое этой опцией, истинно, если размер файла больше/меньше n. При этом различают два случая: если присутствует опция c, то размер файла предполагается заданным в байтах, если опция с отсутствует – то в блоках по 512 байтов.

Следующая команда выполняет поиск файлов, размер которых превышает 256 байтов, в каталоге DIR:

```
$ find DIR -size +256c -print
```

Особую гибкость команде **find** дает еще одна возможность – выполнение команды или группы команд, принимающих в качестве параметра результат поиска файлов. Для реализации такой возможности служит опция **-exec**. В этом случае команда должна заканчиваться пробелом и символами **\;**.

Рассмотрим следующий пример. Пусть в каталоге DIR необходимо удалить все файлы, размер которых не превышает 100 байтов. Подобную операцию можно выполнить при помощи командной строки

```
$ find DIR -size -100c -print -exec rm {} \;
```

Еще один пример использования опции **-exec**:

```
$ find TEST -name 't*' -exec ls -l {} \;
```

Здесь на консоль выводятся атрибуты всех файлов (команда **ls -l**), удовлетворяющих шаблону t*.

Можно расширить возможности команды **find**, перенаправив ее вывод не на стандартное устройство вывода, а в программный канал. В этом случае значительно расширяются возможности поиска и обработки файлов.

Следующая команда

```
$ find DIR -name 't*' -print|grep tmp
```

выполняет поиск файлов в каталоге DIR, удовлетворяющих шаблону **t***, в имени которых присутствует tmp.

Очень часто конвейер программ применяется в операциях копирования, перемещения и создания резервных копий файловых систем, когда вывод команды **find** служит вводом для команды **cpio**.

Команда **find** часто используется в командных файлах, когда нужно выполнить определенные однотипные действия над объектами файловой системы, удовлетворяющими определенным критериям.

Например, можно разработать командные файлы, позволяющие убрать мусор – ставшие ненужными временные файлы или файлы, имеющие нулевой размер. Такие файлы могут создаваться как самой операционной системой, так и работающими приложениями. Накапливаясь, подобные файлы уменьшают свободное дисковое пространство и замедляют работу файловой системы в целом. Хочу сказать, что часть файлов, имеющих нулевой размер, является файлами устройств или системными, поэтому нужно осторожно подходить к каким-либо манипуляциям с такими файлами.

Рассмотрим несколько примеров командных файлов с использованием команды **find**. Следующий командный файл (назовем его find_zero) обнаруживает в текущем каталоге файлы, имеющие нулевой размер, и удаляет их:

```
if [ $# -ne 1 ]
  then
    echo "Usage: $0 [file]"
  else
    dir=`pwd`
    list=`find "$dir" -name "$1"`
    echo Searching zero-length files in current directory...
    for file in $list
      do
        if [ -f $file -a   ! -s $file ]
          then
            echo Found "$file"
            rm $file
            echo Successfully deleted.
        fi
    done
fi
echo Done.
```

Здесь переменной **dir** присваивается строка, представляющая путь к текущему каталогу, в котором выполняется командный файл:

```
dir=`pwd`
```

В качестве единственного параметра программа принимает шаблон имени файла **$1**. Для поиска файлов здесь используется команда **find**. Оператор

```
if [ -f $file -a  ! -s $file ]
```

выполняет проверку найденного объекта на принадлежность к файлам (опция **-f**) и на равенство его размера нулю (опция **! -s**).

Если задать командную строку, например в виде

```
$ ./find_zero "*"
```

то будут выполнены поиск и удаление всех файлов нулевого размера.

Если задать командную строку в таком виде

```
$ ./find_zero "t*"
```

то ищутся и удаляются только файлы, начинающиеся с символа **t**.

Архивирование данных в Linux

В процессе функционирования в операционной системе Linux происходит постоянное увеличение как количества файлов, так и их размера. Это рано или поздно приводит к нескольким неприятным вещам. Во-первых, заполняется дисковое пространство, и какова бы ни была емкость дисков, наступает момент времени, когда ее оказывается недостаточно. Во-вторых, замедляется работа самой операционной системы: чем больше файлов, тем больше времени требуется для выполнения файловых операций, что связано с произвольным расположением кластеров на диске.

Как показывают наблюдения, доступ к большинству файлов происходит довольно редко, особенно это касается программ, управляющих базами данных. В таких случаях имеет смысл хранить редко используемые файлы в виде архивов. Еще один важный момент – архивы данных нужны при авариях системы, когда требуется восстановить данные за определенный период времени.

Для операций архивирования и обратных им операций восстановления в операционной системе Linux предусмотрен целый ряд команд, в их числе и уже знакомая нам команда **find**.

Команда **find** очень широко используется при операциях архивирования и перемещения файлов и целых файловых систем. Чаще

всего в таких операциях она применяется в паре с командой **cpio**, позволяющей копировать файлы в архив или извлекать их из архива. Опции команды **-i**, **-o** и **-p** задают требуемую операцию.

Команда **cpio -i** (от англ. *copy in*) извлекает файлы из стандартного входного потока, предположительно сформированного предыдущей командой **cpio -o**. Выбираются только имена файлов, соответствующие шаблону. Извлекаемые файлы копируются в текущее дерево каталогов в соответствии с дополнительными опциями. Права доступа для созданных файлов соответствуют тем, которые были во время создания архива командой **cpio -o**. При этом атрибуты пользователя и группы-владельца устанавливаются в соответствии с теми, которые имеются у текущего пользователя, если только это не суперпользователь root – в этом случае владельцы будут такие же, как и при выполнении соответствующей команды **cpio -o**.

Используемая с опцией **-i** команда имеет синтаксис:

```
cpio -i   [ bBcdfkmPrsStuvV6 ] [ -C размер_буфера ]
          [ -E файл ] [ -H формат ] [ -I файл [ -M сообщение ] ]
          [ -R id ] [ шаблон ... ]
```

Смысл дополнительных опций (не только для **-i**) мы рассмотрим немного позже. Должен заметить, что при попытке создания файла, который уже существует, с той же датой изменения или более новой, команда **cpio -i** выдает предупреждающее сообщение и не перезаписывает существующий файл. Если необходимо безусловно перезаписать существующие файлы, следует задать опцию **-u**.

Команда **cpio -o** (от англ. *copy out*) читает список файлов из входного потока и копирует эти файлы в стандартный выходной поток вместе с полным путевым именем и информацией о состоянии. Результат по умолчанию выравнивается по 8192-байтовой границе или до указанного пользователем (с помощью опций **-B** или **-C**) размера блока, или, при необходимости, до специфического размера блока устройства. Команда **cpio -o** имеет ряд дополнительных опций, представленных далее, смысл которых мы рассмотрим позже:

```
cpio -o  [ aABcLPvV ]  [ -C размер_буфера ]
         [ -H формат ] [ -O файл [ -M сообщение ] ]
```

Команда **cpio -o** использует стандартный входной поток, причем, если он формируется и направляется по программному каналу команде **cpio -o**, утилита группирует файлы так, что они могут быть перенаправлены (**>**) в один файл архива. Опция **-c** (как и опция **-H**)

гарантирует, что файл архива будет переносим на другие машины. Вместо **ls** можно использовать **find**, **echo**, **cat** и любые другие команды, формирующие в стандартном выходном потоке список имен файлов для **cpio**. Результат можно перенаправить на устройство, а не в обычный файл.

Еще один режим работы команды **cpio** – режим передачи. В этом режиме **cpio -p** (от англ. *pass*) читает список файлов, которые при необходимости, в зависимости от описанных далее опций, создаются и копируются в указанное целевое дерево каталогов.

cpio -p [adlLmPuvV] [-R id] ***каталог***

Проанализируем более детально смысл некоторых дополнительных опций, которые можно в любой последовательности указывать вместе с опциями **-o**, **-i** или **-p**:

- **-a** – выполняет сброс времени обращения к исходным файлам после их копирования. При этом время обращения не сбрасывается для связанных файлов при указании опций **cpio -pla**;
- **-A** – добавляет файлы в архив, требуя обязательного указания опции **-O**. Использование данной опции допустимо только для архивов в файлах, на дискетах или на разделах жесткого диска;
- **-b** – изменяет порядок байтов в каждом слове и используется исключительно с опцией **-i**;
- **-B** – осуществляет ввод/вывод в виде блоков по 5120 байтов. Если эта опция и опция **-C** не указаны, то используется стандартный размер блока – 8192 байта. Опцию **-B** нельзя использовать в режиме передачи, кроме того, она имеет смысл только при обмене данными со специальным символьным устройством, например /dev/rmt/0m;
- **-C размер_буфера** – объединяет ввод/вывод в записи указанного размера (размер задается положительным целым числом). Стандартный размер буфера, если не задана эта опция и опция **-B**, равен 8192 байта. Опция **-C** имеет смысл, только если происходит обмен данными со специальным символьным устройством, например /dev/rmt/0m;
- **-d** – создает каталоги при необходимости;
- **-E файл** – задает входной файл, содержащий список имен файлов, которые необходимо извлечь из архива (по одному имени в строке).

Если при записи на специальное символьное устройство (**-o**) или при чтении со специального символьного устройства (**-i**) утилита

cpio обнаруживает конец носителя (например, конец дискеты) и при этом не заданы опции **-O** и **-I**, выдается такое сообщение:

```
To continue, type device/file name when ready.
(Для продолжения введите имя устройства/файла.)
```

Для продолжения необходимо заменить носитель, ввести имя специального символьного устройства (например, /dev/rdiskette) и нажать клавишу **Enter**. Можно для продолжения архивирования задать **cpio** для другого устройства. Например, при наличии двух дисководов можно переключать архивирование с одного на другой, чтобы оно продолжалось, пока вы меняете дискету. (Если просто нажать клавишу **Enter**, процесс **cpio** завершает работу.)

Кроме дополнительных опций, команда **cpio** поддерживает следующие операнды:

- **каталог** – путь к существующему целевому каталогу для команды **cpio -p**;
- **шаблон** – выражение с теми же метасимволами сопоставления с образцом, что и шаблоны имен файлов командного интерпретатора:
  - ✧ **?** – соответствует любому одиночному символу;
  - ✧ **[...]** – соответствует любому из указанных в квадратных скобках символов. Пара символов, разделенных дефисом (**-**), соответствует любому из символов диапазона (включительно с указанными), определяемых по стандартной кодовой таблице. Если сразу за открывающей скобкой (**[**) идет восклицательный знак (**!**), то результаты будут неопределенными;
  - ✧ **!** – отрицание (например, **!abc*** исключает из обработки все файлы, имена которых начинаются с abc).

В шаблонах метасимволы **?**, ***** и **[...]** сопоставляются с символом косой черты (**/**), а символ обратной косой черты (****) является маскирующим. Можно задавать несколько шаблонов, а если шаблон не задан, предполагается шаблон ***** (то есть выбираются все файлы). Каждый шаблон необходимо брать в двойные кавычки, в противном случае командный интерпретатор может подставить имена файлов текущего каталога.

Рассмотрим несколько примеров использования утилиты cpio для архивирования и восстановления файлов.

Пусть требуется создать архив файлов текущего каталога и записать его в файл arch. Предположим, что текущий каталог содержит такие файлы:

```
$ ls -l
-rw-r--r--   1 yury       yury       10 Feb   5 17:32 memo1
-rw-r--r--   1 yury       yury       14 Feb   5 17:32 memo2
-rw-r--r--   1 yury       yury       17 Feb   5 17:32 test1
```

Для архивирования файлов следует воспользоваться командой

```
$ ls | cpio -oc > arch
```

Извлечь все файлы из созданного архива можно с помощью команды

```
$ cat arch | cpio -icd
```

Если необходимо восстановить из архива только некоторые файлы, то следует их перечислить в команде **cpio**:

```
$ cat arch | cpio -icd "memo1" "memo2"
```

Здесь файлы memo1 и memo2 извлекаются из ранее созданного архива arch. Опция **-c** используется, если архив был создан с переносимым заголовком. Если путевые имена восстанавливаемых файлов содержат одинаковые символы, можно воспользоваться шаблоном

```
$ cat arch | cpio -icd "m*"
```

В следующем примере выполняется копирование файлов в другой каталог. Для этого применяется команда **cpio -p**, которая принимает из стандартного входного потока список файлов и копирует или создает на них ссылки (опция **-l**) в другом каталоге (NEWDIR в данном случае). Опция **-d** требует создания каталогов при необходимости, а опция **-m** запрещает модификацию времени изменения файла.

Для генерации списка полных путевых имен файлов для **cpio** в команде **find** нужно задать опцию **-depth**. Это позволяет создавать файлы в каталогах, доступных только для чтения. Замечу, что целевой каталог NEWDIR должен существовать к моменту выполнения команды. Вот полная командная строка для выполнения операции копирования:

```
$ find . -depth -print | cpio -pdlmv NEWDIR
```

В следующем примере выполняется копирование файлов, находящихся в текущем каталоге и удовлетворяющих шаблону **t***, в архивный файл TARCH:

```
$ find . -name 't*' -print|cpio -o   > TARCH
1 block
```

Восстановить файлы из архива TARCH можно при помощи командной строки

```
$ cpio -iuvB < TARCH
```

Для извлечения отдельных файлов из архива, созданного командой **cpio**, в командной строке следует указать имена файлов:

```
$ cpio -iuvB < TARCH "test2" "test3"
```

Здесь из архива TARCH извлекаются только файлы test2 и test3.

Используя опцию **-t**, можно просмотреть содержимое архива:

```
$ cpio -it < TARCH
```

Команда **cpio** по завершении формирует статус выхода. При этом значению 0 соответствует успешное завершение, если же значение статуса больше 0, это значит, что произошла ошибка.

Еще одной, очень популярной утилитой, позволяющей архивировать и восстанавливать файлы, является **tar**. Команда **tar** позволяет сохранять файлы на архивном носителе и/или восстанавливать их с данного носителя. Операции, выполняемые командой, определяются строкой символов, содержащей одну опцию (**c**, **r**, **t**, **u** или **x**) и, возможно, один или несколько модификаторов (**v**, **w**, **f**, **b**, **L**, **k**, **F**, **X**, **h**, **i**, **e**, **n**, **A**, **l**, **m**, **o**, **p** и **num**).

Остальные аргументы команды – имена файлов (или каталогов), указывающие, какие файлы необходимо заархивировать или извлечь из архива. Во всех случаях указание имени каталога означает ссылку на все файлы и (рекурсивно) подкаталоги этого каталога.

Команда принимает множество опций, наиболее часто используемыми из которых являются:

- **-c** – позволяет создавать новый архив, при этом запись начинается с начала архива;
- **-r** – позволяет добавлять указанные файлы в конец существующего архива;
- **-t** – имена и другая информация об указанных файлах выдаются для каждого их вхождения в архив (задан модификатор **v**). При отсутствии модификатора **v** выдаются только имена файлов в формате, аналогичном формату команды **ls -l**, если же файлы не заданы, то выдается информация обо всех файлах в архиве;
- **-u** – позволяет добавить указанные файлы в архив, если их еще нет или если они изменились с момента последней записи в данный архив;

- **-x** – позволяет извлекать указанные файлы из архива, при этом если приведенное имя соответствует имени каталога, содержимое которого было записано в архив, то извлекается (рекурсивно) этот каталог. Желательно использовать относительный путь к файлу или каталогу, иначе **tar** не найдет его. Восстанавливаются также атрибуты файла (владелец, дата изменения и права доступа к файлу, если это возможно). Если файлы не указаны, извлекается все содержимое архива.

В процессе работы команда может выдавать сообщения об ошибках чтения/записи или о нехватке свободной памяти для размещения таблиц связей. Кроме того, существует ограничение на длину путевого имени файла, которая не должна превышать 100 символов.

Недостатком команды **tar** является и низкая скорость работы в режиме **u**, **a** опция **b** не может быть использована при работе с архивом, который должен обновляться. Если архив находится в дисковом файле, то опцию **b** нельзя применять ни в коем случае, потому что обновление архива, расположенного на диске, может разрушить его. Кроме того, команда **tar** не копирует пустые каталоги и специальные файлы.

Что же касается совместимости форматов архивов, созданных различными программами, то, например, утилита **cpio** распознает архивы, созданные при помощи **tar**, поэтому ее можно применять для чтения таких архивов.

Рассмотрим несколько примеров использования команды **tar**.

Предположим, что в текущем каталоге находятся такие файлы:

```
-rw-r--r--      1  root        root       10  Feb   8  01:09  test1
-rw-r--r--      1  root        root       19  Feb   8  01:09  test2
-rw-r--r--      1  root        root        0  Feb   8  01:10  test3
```

Для создания архива archive.tar, содержащего файлы test1, test2 и test3, нужно выполнить команду

```
$ tar -cf archive.tar test1 test2 test3
```

Просмотреть содержимое созданного архива можно при помощи команды

```
$ tar -tvf archive.tar
-rw-r--r-- root/root            10  2006-02-08  01:09:50  test1
-rw-r--r-- root/root            19  2006-02-08  01:09:58  test2
-rw-r--r-- root/root             0  2006-02-08  01:10:08  test3
```

Чтобы извлечь файлы из архива, нужно выполнить команду

```
$ tar -xf archive.tar
```

Пусть необходимо скопировать содержимое текущего каталога в каталог ARCHTAR, который еще не создан. Особенностью команды **tar** является то, что, заданная с модификатором **f** и опцией «**-**», она может прочитать стандартный ввод и использоваться в конвейере команд. Следующая последовательность команд позволяет скопировать содержимое рабочего каталога в целевой каталог ARCHTAR:

```
$ tar cf - .|(mkdir ARCHTAR; cd ARCHTAR; tar xf -)
```

Для копирования нескольких каталогов в целевой каталог можно создать несложный командный файл:

```
for x in $*
 do
  tar cf - $x|(cd `pwd`/DEST; tar xf -)
 done
```

Здесь содержимое каталогов, заданных в качестве аргументов командной строки, копируется в каталог DEST, который к моменту копирования уже должен существовать.

Таковы особенности функционирования операционных систем Linux. Raspbian OS является одним из клонов Linux, а именно Debian Linux, поэтому все вышеизложенное относится и к этой операционной системе. В следующих главах будут обсуждаться специфичные особенности Raspbian OS.

4 Особенности функционирования Raspbian OS в Raspberry Pi

После загрузки операционной системы Raspbian OS пользователь увидит примерно такой графический интерфейс (рис. 4.1). На данном рисунке показан рабочий стол с настройками под конкретного пользователя, с дополнительными пиктограммами, но в целом картинка будет похожа на ту, что появляется при первой загрузке операционной системы.

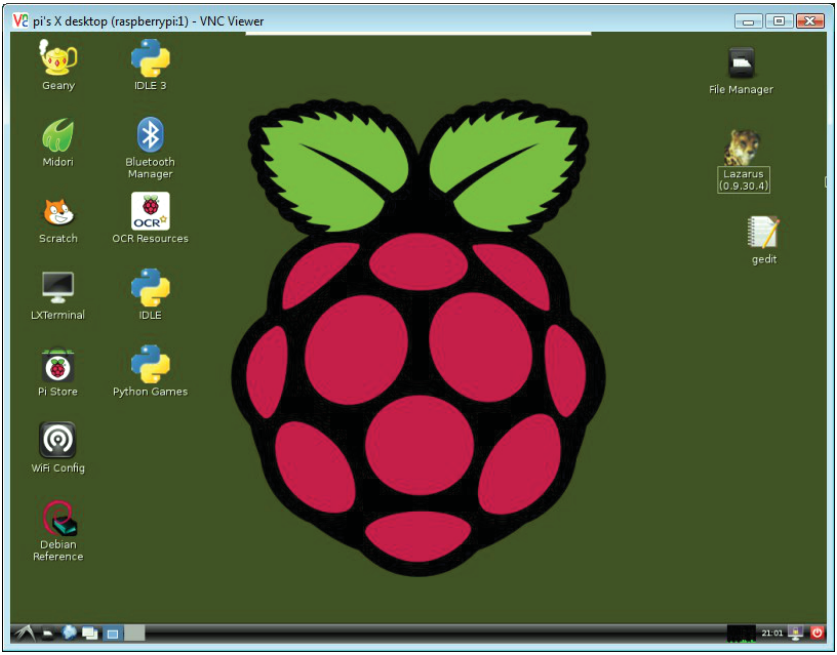

Рис. 4.1

По умолчанию в Raspbian OS используется графическая оболочка, известная под названием Lightweight X11 Desktop Environment (LXDE). LXDE, в свою очередь, базируется на X Window System и обеспечивает простой и интуитивно-понятный графический интерфейс.

В некоторых дистрибутивах графическая оболочка не загружается по умолчанию, поэтому после входа в систему нужно набрать команду **`startx`**. Дистрибутив Debian Linux содержит массу приложений с графическим интерфейсом, которые можно условно разбить на несколько групп. Для просмотра этих групп и их содержи-

мого нужно щелкнуть левой кнопкой мыши на иконке меню в левом нижнем углу экрана.

Пользователь может избрать любой из двух методов работы с операционной средой Raspbian OS. В первом случае можно использовать текстовую консоль, с которой можно набирать команды Linux. Можно воспользоваться и более дружественным графическим интерфейсом, который обычно предпочитает большинство пользователей.

Установка и обновление программ

Дистрибутив Raspbian OS поставляется с многочисленными предустановленными программами. Пользователь может установить также необходимые программы, используя один из двух вариантов. Первый метод – традиционный для Debian Linux операционных систем, когда загрузка и инсталляция программного обеспечения осуществляются с удаленного сервера, который называется репозитарием. Для этого в командной строке можно выполнить команды:

```
$ sudo apt-get update
$ sudo apt-get install <имя_пакета_программ>
```

С начала выпуска Raspberry Pi появилась еще одна возможность для установки программного обеспечения, которая использует репозиторий Pi Store, в котором собрано программное обеспечение, специально разработанное для компьютера Raspberry Pi. Для загрузки и инсталляции программ с Pi Store нужно набрать следующую команду:

```
$ sudo apt-get update && sudo apt-get install pistore
```

Программное обеспечение на Pi Store (http://store.raspberrypi.com) оптимизировано для применения с Raspberryy Pi, при этом все требуемые предварительные настройки уже выполнены, что обеспечивает легкий процесс инсталляции. Главная страница этого сайта показана на рис. 4.2.

Программирование в Raspbian OS

Одной из основных задач, которые ставили перед собой разработчики Raspberry Pi, было предоставление возможности для создания программных приложений широкой аудиторией пользователей, включая

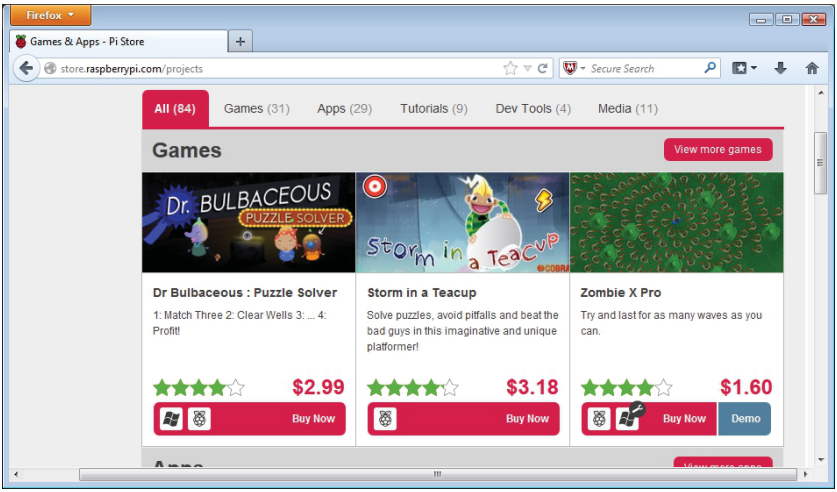

Рис. 4.2

школьников и студентов. Процесс программирования в Raspberry Pi является относительно легким благодаря наличию различных инструментов разработки, включенных в дистрибутив Raspbian OS.

Одним из основных языков программирования, поддерживаемых в Raspberry Pi, является Python. Если пользователь хоть немного знаком с основами программирования на языке Python, то программирование в RPi будет для него несложной задачей, поскольку эта среда программирования очень хорошо поддерживается в Raspbian OS. Кроме базовых модулей, Python включает модуль Pygame, с помощью которого можно программировать игровые приложения, что может заинтересовать многих начинающих программистов.

Поскольку Python является кросс-платформенным языком программирования, разработчик имеет возможность создавать приложения на других платформах, например в Windows или MacOS, а затем легко адаптировать код в Raspbian OS. Так как Python очень широко распространен, то в Интернете имеется очень много документации по программированию на этом языке, к тому же разработаны многочисленные модули с библиотечными функциями для работы с различными аппаратными интерфейсами. Для новичков можно порекомендовать сайт http://docs.python.org/3/, где имеется масса полезной документации по программированию на языке Python.

Несмотря на то что Python рассматривается как базовый язык программирования в Raspberry Pi, в дистрибутив Raspbian OS включены и другие популярные средства разработки, такие как Java, PHP и Scratch. Язык Java для Raspberry Pi в настоящее время находится на последнем этапе тестирования, проводимого фирмой Oracle; PHP можно использовать для многих интересных проектов, связанных с использованием мини-компьютера Raspberry Pi в качестве веб-сервера.

Среда программирования на языке Scratch также включена в дистрибутив Raspbian OS. Scratch использует графические блоки и технологию «drag-and-drop», что позволяет обучить основам программирования даже детей. С помощью Scratch можно легко моделировать игровые ситуации и создавать довольно сложные игры с различными персонажами, включенными в базу спрайтов.

На рис. 4.3 показан общий вид интерфейса программирования Scratch.

Язык Scratch является интуитивно-понятным, тем не менее имеется хорошая документация по программированию, а также различные проекты на сайте http://scratch.mit.edu/.

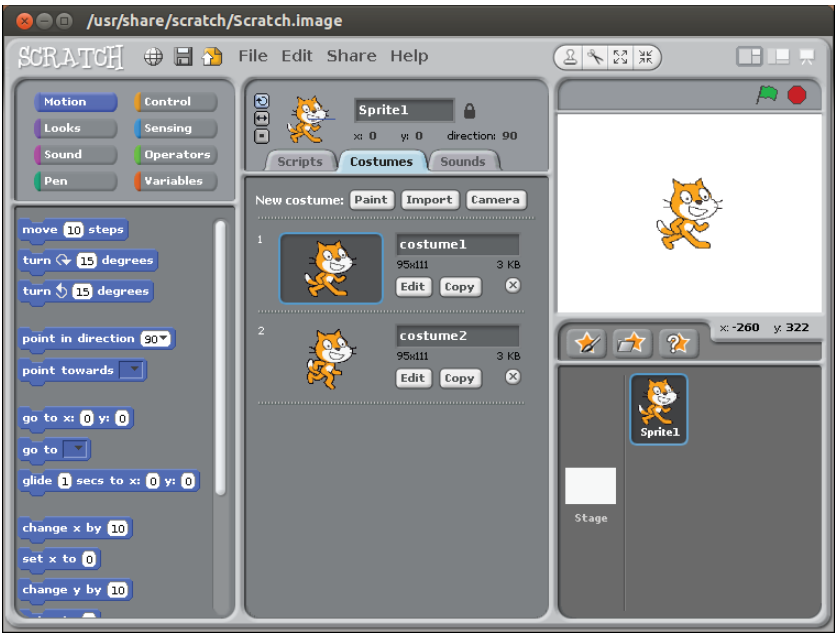

Рис. 4.3

5 Сетевые настройки Raspbian OS

В большинстве случаев подключение RPi к сети выполняется просто – нужно просто подключить сетевой кабель к порту Ethernet на плате. При этом система сама выполнит все необходимые настройки сети автоматически. Подобная легкость, однако, возможна только в том случае, если в вашей сети функционирует сервис автоматического присвоения сетевых адресов (DHCP), а в самой операционной системе Raspbian OS работает клиентское программное обеспечение DHCP.

Если сервис DHCP по каким-либо причинам недоступен, то настройку сети Raspberry Pi придется выполнить вручную.

При ручной настройке сети вначале нужно найти файл, где описаны сетевые интерфейсы. Этот файл называется interfaces и находится в каталоге /etc/network. Данный файл может редактировать только пользователь с правами суперпользователя root. Необходимо помнить, что удаление сетевого интерфейса из списка интерфейсов вызовет остановку работы этого интерфейса.

Для редактирования файла /etc/network/interfaces необходимо из терминальной программы вызвать текстовый редактор, например nano, с помощью команды:

```
pi@raspberrypi / $ sudo nano /etc/network/interfaces
```

Текстовый редактор nano – это простой и достаточно мощный текстовый редактор (рис. 5.1).

Навигация по открытому документу осуществляется клавишами перемещения. Для сохранения изменений в документе необходимо нажать комбинацию **CTRL**+**Q**, а для выхода из программы нужно нажать **CTRL**+**X**.

В этом файле необходимо отредактировать строку, которая начинается с `iface eth0 inet`. В этой строке нужно удалить `dhcp` и вместо него набрать `static`. После этого следует перейти к новой строке, нажав клавишу **Enter**, и ввести строки в следующем формате:

```
address xxx.xxx.xxx.xxx
netmask xxx.xxx.xxx.xxx
gateway xxx.xxx.xxx.xxx
```

Каждой из этих строк должно предшествовать нажатие клавиши **Tab**. Символы x в каждой строке должны представлять статические сетевые IP-адреса, которые пользователь должен ввести вручную. В строке с `netmask` необходимо ввести маску сети. Для домаш-

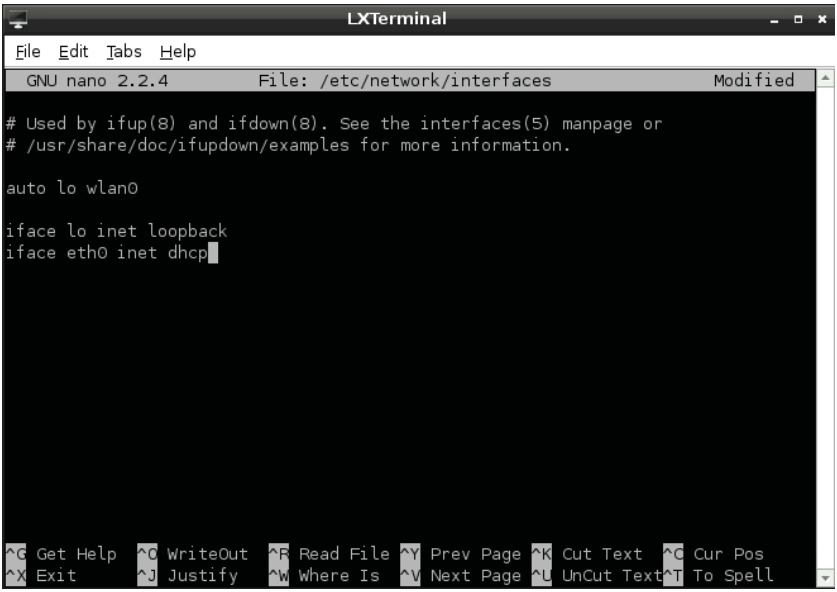

Рис. 5.1

ней сети маска сети набирается как 255.255.255.0. В строке **gateway** необходимо указать IP-адрес маршрутизатора или кабельного модема.

Ниже приводится пример конфигурирования домашней сети со статическими IP-адресами:

```
iface eth0 inet static
[Tab] address 192.168.1.4
[Tab] netmask 255.255.255.0
[Tab] gateway 192.168.1.1
```

По окончании редактирования файла нужно сохранить изменения, нажав комбинацию **CTRL**+**O**, после чего можно выйти из редактора nano по **CTRL**+**X**. Новые настройки будут работать после перезапуска сетевых сервисов, поэтому нужно выполнить команду

```
pi@raspberrypi / $ sudo /etc/init.d/networking restart
```

Если понадобится вновь вернуться к DHCP, то нужно отредактировать файл interfaces, удалив при этом строки с IP-адресами, и заменить **static** на **dhcp** в соответствующей строке.

Установки IP-адреса может оказаться недостаточно для связи RPi с внешним миром, поскольку компьютеры в глобальных сетях, в частности в Интернете, помимо IP-адреса, имеют так называемые «доменные имена». Доменные имена существенно облегчают пользователям поиск компьютеров в Интернете. Например, намного легче запомнить и написать www.raspberrypi.org в адресном поле браузера вместо IP-адреса 93.93.128.176.

Для идентификации компьютеров в сети по их доменным именам используется сервис DNS (Domain Name Service), обычно работающий на специально выделенном сервере. DNS-сервер осуществляет преобразование доменных имен в их IP-адреса, что необходимо для доступа к сетевому ресурсу. Перед тем как использовать данный сервис, пользователь должен указать операционной системе, какой DNS-сервер использовать. Список серверов DNS, которые часто называют «серверы имен», находится в файле /etc/resolv.conf. Когда используется DHCP-сервис, то этот файл заполняется автоматически.

При ручном конфигурировании IP-адресов пользователю следует указать IP-адрес сервера имен; обычно сервер имен располагается на маршрутизаторе. Для конфигурирования серверов имен нужно вначале открыть файл resolv.conf с помощью команды

```
pi@raspberrypi / $ sudo nano /etc/resolv.conf
```

Необходимо добавить каждый из DNS-серверов в отдельной строке. Например, для сети, в которой используются серверы имен Google, содержимое файла resolv.conf может выглядеть следующим образом:

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

После внесения всех изменений необходимо сохранить файл, нажав комбинацию **CTRL**+**O**, затем покинуть редактор.

Для активации сетевых настроек следует перезапустить сетевые службы:

```
pi@raspberrypi / $ sudo /etc/init.d/networking restart
```

Для тестирования сетевых настроек можно воспользоваться либо браузером, либо набрать команду

```
pi@raspberrypi / $ ping -c 1 www.raspberrypi.org
```

Вот так может выглядеть терминальное окно работающей команды ping (рис. 5.2):

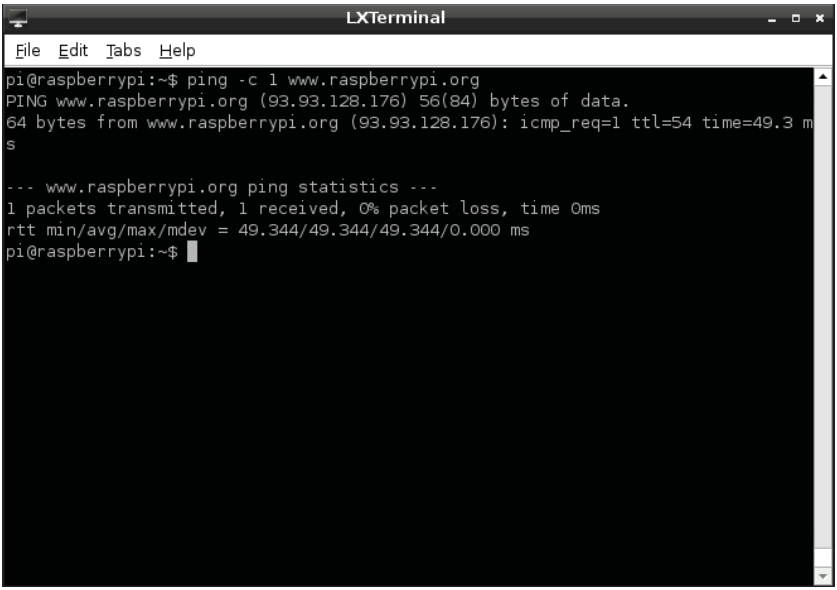

Рис. 5.2

Настройка беспроводной сети в Raspberry Pi

Несмотря на то что на плате Raspberry Pi отсутствует аппаратный модуль для поддержки беспроводной сети Wi-Fi, такую возможность можно обеспечить путем подключения простого и недорогого адаптера Wi-Fi. Для этого, помимо установки адаптера, потребуется сконфигурировать программную часть беспроводной сети.

Следует учитывать то, что адаптеры USB Wi-Fi потребляют значительную мощность, поэтому если вы подключите адаптер непосредственно к USB-порту на плате RPi, то, скорее всего, система перестанет работать. Для подключения Wi-Fi-адаптера нужно использовать USB-хаб с отдельным питанием.

Перед тем как приступить к конфигурированию беспроводного интерфейса, необходимо определить SSID (Service Set Identifier),

или, по-другому, сетевое имя беспроводного маршрутизатора, к которому осуществляется подключение. Кроме того, следует знать тип шифрования, пароль и протокол сети, поддерживаемый маршрутизатором. Wi-Fi-адаптер, поддерживающий протокол 802.11a, не будет работать с другим сетевым протоколом, например 802.11g.

Для того чтобы операционная система могла обращаться к адаптеру USB Wi-Fi, необходимо установить соответствующее программное обеспечение. Некоторые дистрибутивы включают программное обеспечение для работы со стандартным Wi-Fi-устройством, в некоторых для экономии места программные файлы для поддержки Wi-Fi отсутствуют. В большинстве случаев программное обеспечение для Wi-Fi следует устанавливать вручную.

Для установки корректной версии ПО для данного адаптера Wi-Fi необходимо знать тип адаптера. Большинство производителей Wi-Fi-адаптеров выпускают устройства на одних и тех же базовых компонентах, поэтому маркировка на устройстве может дать мало информации о том, какое программное обеспечение нужно установить. Для определения типа базового кристалла, установленного в Wi-Fi-адаптере, нужно подсоединить устройство к Raspberry Pi и проверить кольцевой буфер ядра на наличие сообщений об ошибках.

Кольцевой буфер ядра – это область памяти, используемая ядром Linux для хранения информации в удобном для чтения формате. Это очень важная особенность операционной системы, поскольку при загрузке Linux сообщения в текстовом формате проходят слишком быстро для того, чтобы успеть их проанализировать. Это можно сделать позже, прочитав содержимое кольцевого буфера.

Если Wi-Fi-адаптер подсоединен к системе, но драйверы и утилиты для данного устройства не установлены, то ядро выдаст серию сообщений об ошибках, которые будут помещены в кольцевой буфер. Для чтения этих сообщений на экране можно воспользоваться командой **dmesg**, которую можно запустить из любой терминальной программы.

Результатом выполнения команды **dmesg** будет все содержимое кольцевого буфера, отображающее все сообщения системы. Поскольку нам необходимо ограничить этот список только сообщениями, касающимися нашего адаптера, то нужно воспользоваться следующей командой:

```
pi@raspberrypi / $ dmesg | grep usb
```

На рис. 5.3 показано содержимое экрана после выполнения данной команды. Легко определить, что в системе используется USB Wi-Fi-адаптер Zyxel NWD2015.

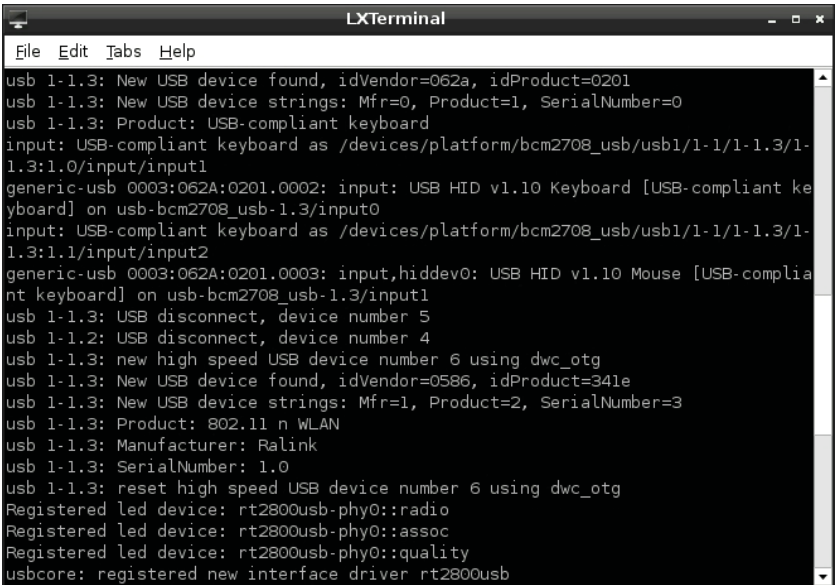

Рис. 5.3

В выведенной на экран информации можно заметить строку, в которой указан производитель устройства Zyxel NWD2105, в данном случае это Ralink, компания, которая производит базовый чип для адаптера Zyxel NWD2105. Естественно, что нормальное функционирование данного Wi-Fi-адаптера возможно только с драйверами этой компании.

В случае если по ключу **usb** ничего обнаружить не удалось, можно воспользоваться командой **lsusb**, которая выведет на экран список USB-устройств. Для поиска программного обеспечения адаптера можно воспользоваться командой **apt-cache**. В данном случае (для адаптера Zyxel NWD2015) командная строка будет выглядеть так:

```
pi@raspberrypi / $ apt-cache search ralink
```

Если с помощью этой команды не удалось найти подходящие драйверы, то можно попробовать установить один из пакетов, которые присутствуют в дистрибутиве Debian Linux:

- atmel-firmware – подходит для устройств на базе чипсета Atmel AT76C50X;
- firmware-atheros – подходит для устройств на базе чипсета Atheros;
- firmware-brcm80211 – подходит для устройств на базе чипсета Broadcom;
- firmware-intelwimax – подходит для устройств на базе чипсета Intel's WiMAX;
- firmware-ipw2x00 – подходит для устройств на базе чипсета Intel Pro Wireless (включая версии 2100, 2200 и 2915);
- firmware-iwlwifi – подходит для других устройств на базе чипсета Intel (включая версии 3945, 4965 и 5000);
- firmware-ralink – подходит для устройств на базе чипсета Ralink;
- firmware-realtek – подходит для устройств на базе чипсета Realtek;
- zd1211-firmware – подходит для устройств на базе чипсета ZyDAS 1211.

Если даже вы установите программное обеспечение, которое не будет работать с данным адаптером, не следует беспокоиться – его можно легко удалить с помощью команды **apt-get remove**.

После того как программное обеспечение для данного Wi-Fi-адаптера установлено, настройка сетевых соединений не составляет труда. Прежде всего нужно убедиться, что USB Wi-Fi-адаптер работает, используя команду **iwlist**.

Эта команда позволяет осуществить сканирование близлежащих точек доступа (если таковые имеются):

```
pi@raspberrypi / $ sudo iwlist scan | less
```

Данная команда возвращает список беспроводных сетей, доступных для Raspberry Pi, а также детальное описание параметров каждой из обнаруженных сетей (рис. 5.4).

Текущее состояние беспроводной сети также можно определить с помощью команды **iwconfig**. Как и команда **ifconfig**, **iwconfig** позволяет определить состояние беспроводной сети и, при необходимости, настроить параметры такой сети. В отличие от **ifconfig**, команда

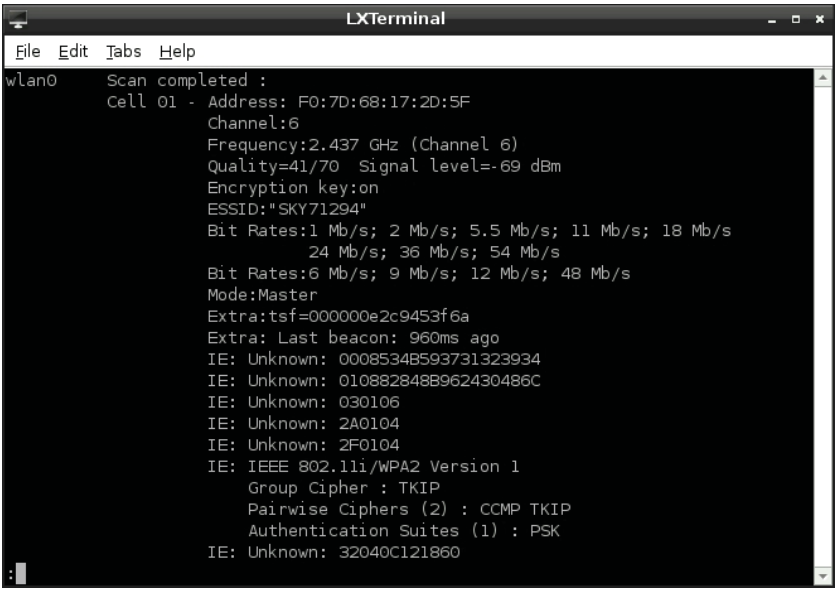

Рис. 5.4

iwconfig специально была разработана для управления беспроводными сетями. Если набрать команду без параметров

```
pi@raspberrypi / $ iwconfig
```

то на экран консоли будет выведена текущая конфигурация сети (рис. 5.5).

Смысл полученной информации объясняется ниже:

- Interface Name – любому беспроводному адаптеру присваивается уникальное имя, как это имеет место в проводной сети. Имя по умолчанию для Raspberry Pi – wlan0.
- Standard – стандарт беспроводной передачи данных IEEE 802.11 включает целое подмножество, состоящее из различных типов, имена которых отличаются только наличием суффикса.
- ESSID – идентификатор сети (SSID), к которой адаптер подключен. Если адаптер не подключен к беспроводной сети, в этом поле будет off.
- Mode – режим работы адаптера. Беспроводной USB-адаптер может функционировать в одном из следующих режимов:

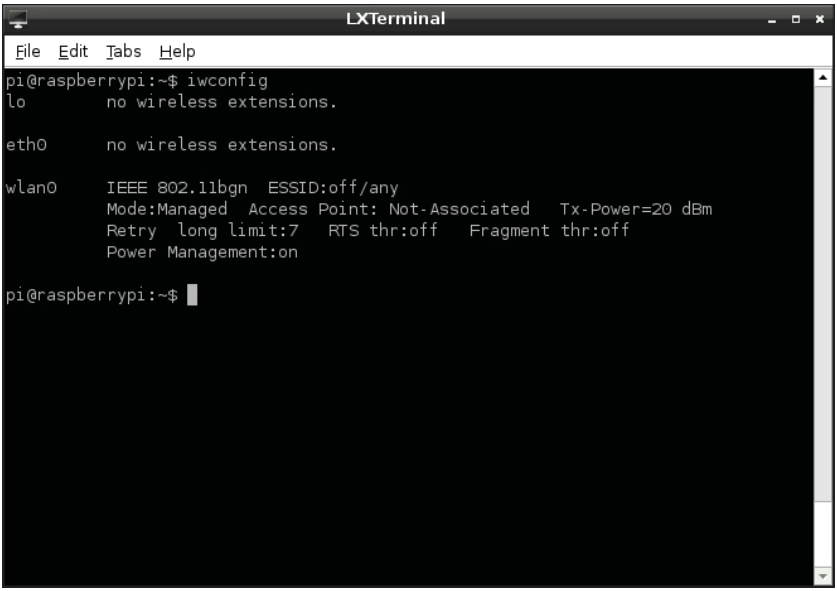

Рис. 5.5

- ✧ Managed – стандартный режим работы беспроводной сети, когда клиент подсоединен к точке доступа (access point). Этот режим используется в большинстве домашних сетей;
- ✧ Ad-Hoc – соединение «устройство – устройство», без управления через точку доступа;
- ✧ Monitor – специальный режим, в котором адаптер прослушивает весь трафик независимо от адресации. Этот режим используется при настройке и диагностике беспроводных сетей;
- ✧ Repeater – специальный режим, в котором адаптер выполняет роль репитера (усилителя) для передачи усиленных сигналов всем остальным сетевым клиентам;
- ✧ Secondary – разновидность режима Repeater, в котором адаптер выполняет роль резервного репитера.

- Access Point – адрес точки доступа, к которой подключается беспроводной адаптер. Если адаптер не подключен ни к одной точке доступа, то он идентифицируется как Not-Associated.

- Tx-Power – мощность сигнала передатчика беспроводного адаптера, выраженная в числовой форме. Бóльшая мощность характеризуется бóльшим значением.
- Retry – количество повторных попыток передачи данных. Обычно это значение не требует ручной настройки, а некоторые адаптеры вообще не позволяют изменять заводские настройки.
- RTS – текущие настройки адаптера для Ready-To-Send и Clear-To-Send (RTS/CTS) сигналов, используемые в загруженных сетях для предотвращения коллизий. Этот параметр обычно устанавливается на стороне точки доступа.
- Fragment – максимальная длина фрагмента пакета данных, обычно используется в загруженных сетях для передачи пакета посредством нескольких фрагментов. Этот параметр чаще всего настраивается на стороне точки доступа.
- Power Management – определяет текущее состояние энергопотребления адаптера и позволяет уменьшить энергопотребление адаптера при отсутствии сетевого трафика. Этот параметр не оказывает большого влияния в системах с Raspberry Pi.

Для подсоединения Raspberry Pi к беспроводной сети необходимо откорректировать несколько строк в файле /etc/network/interfaces. Первое – откроем этот файл для редактирования в текстовом редакторе nano:

```
pi@raspberrypi / $ sudo nano /etc/network/interfaces
```

За последней строкой файла нужно создать запись для конфигурирования беспроводного USB-адаптера (рис. 5.6):

```
auto wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa.conf
```

Сохраним изменения в файле, нажав комбинацию **CTRL**+**O**, затем выйдем из nano, нажав **CTRL**+**X**.

Идентификатор (ID) устройства для wlan0 будет корректен, если это первое устройство в беспроводной сети с Raspberry Pi. Если это не так, то ID будет другим. Для определения списка беспроводных устройств нужно выполнить утилиту **iwconfig**, после чего откорректировать соответствующую запись из предыдущего примера.

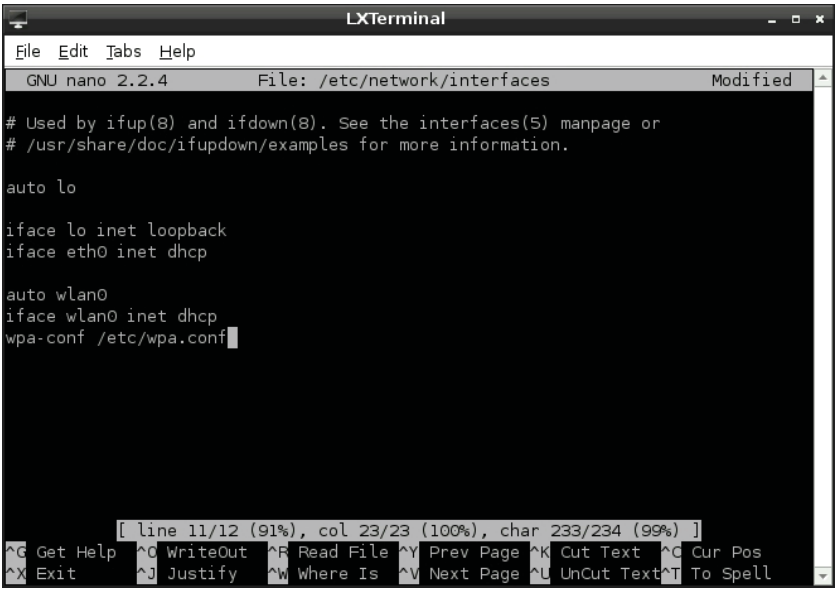

Рис. 5.6

В последней строке файла interfaces дается ссылка на файл конфигурации, wpa.conf, который не существует. Этот файл используется утилитой **wpasupplicant**, которая существенно облегчает процесс конфигурирования беспроводных сетей в Linux и обеспечивает доступ к сети, где используется метод защиты Wireless Protected Access (WPA).

С помощью wpasupplicant можно подключить RPi практически к любой беспроводной сети, независимо от того, какой алгоритм защиты (WPA или более новый, WPA2) используется для доступа.

Утилита **wpasupplicant** хранит данные конфигурации в файле wpa.conf, расположенном в каталоге /etc. Чтобы настроить доступ RPi к беспроводной сети, вначале следует открыть пустой файл для последующего редактирования с помощью редактора nano:

```
pi@raspberrypi / $ sudo nano /etc/wpa.conf
```

Затем нужно набрать следующие строки (рис. 5.7):

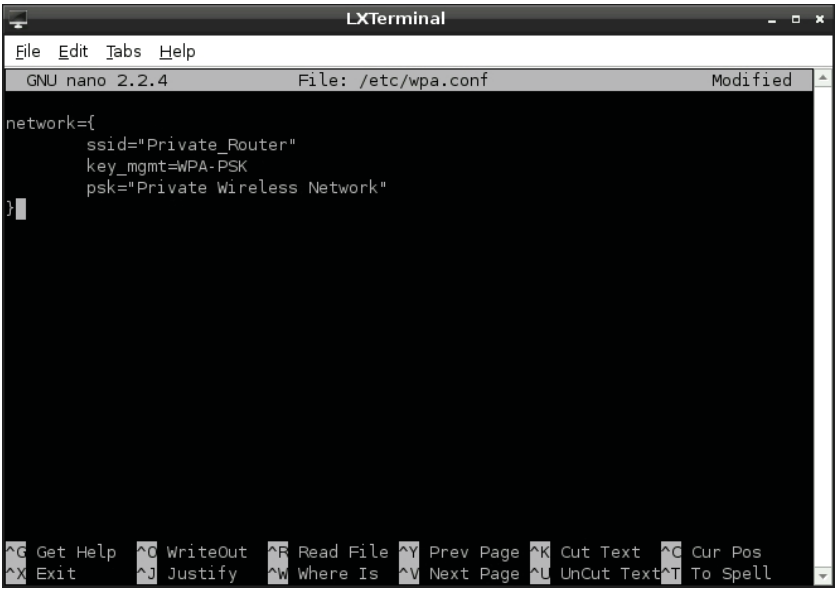

Рис. 5.7

Здесь в строке с SSID нужно указать идентификатор беспроводной сети, к которой выполняется подсоединение:

```
network={
[Tab]  ssid="Your_SSID"
```

Далее необходимо указать параметры кодирования.

Если кодирование не используется (No Encryption), следует ввести строки:

```
[Tab]  key_mgmt=NONE
}
```

и затем сохранить файл (**CTRL**+**O**), после чего выйти из редактора nano (**CTRL**+**X**).

Если используется кодирование (WEP Encryption), то в wpa.conf file следует набрать:

```
[Tab]  key_mgmt=NONE
[Tab]  wep_key0="Your_WEP_Key"
}
```

Здесь нужно вместо "**Your_WEP_Key**" указать ключ в ASCII-формате, используемый для WEP-кодирования. После этого сохраняем изменения (**CTRL**+**O**) и выходим из редактора nano (**CTRL**+**X**).

Нужно отметить, что WEP-кодирование очень слабо защищает сеть, поэтому соответствующие программы взлома легко преодолеют такую защиту в течение нескольких минут. Если все же приходится иметь дело с WEP-защитой, то для лучшей защиты нужно использовать WPA- или WPA2-кодирование.

Рассмотрим конфигурирование файла wpa.conf при использовании WPA/WPA2-кодирования. В этом случае в конец файла добавляются следующие записи:

```
[Tab]  key_mgmt=WPA-PSK
[Tab]  psk="Your_WPA_Key"
}
```

Здесь нужно заменить "**Your_WPA_Key**" на кодовую фразу для данной беспроводной сети. На рис. 5.7 показан пример конфигурирования доступа к беспроводной сети с SSID "**Private_Router**" и WPA кодовой фразой "**Private Wireless Network**". Как всегда, редактирование файла и сохранение его содержимого завершается комбинациями **CTRL**+**O** и **CTRL**+**X**.

После выполнения настроек беспроводной сети следует перезагрузить систему, затем набрать следующую команду:

```
pi@raspberrypi / $ sudo ifup wlan0
```

Чтобы убедиться, что беспроводная сеть работоспособна, нужно отсоединить сетевой кабель от разъема Ethernet и выполнить следующую команду:

```
pi@raspberrypi / $ ping -c 1 www.raspberrypi.org
```

Если после установки USB Wi-Fi-адаптера возникают проблемы с USB-устройствами (чаще всего с USB-клавиатурой), то за решением этой проблемы нужно обратиться по ссылке http://www.element14.com/community/docs/DOC-44703/l/raspberry-pi-wifi-adapter-testing или eLinux wiki at http://elinux.org/RPi_VerifiedPeripherals#Working_USB_Wifi_Adapters.

Доступ к сетевым ресурсам из Raspbian OS

Очень часто возникает необходимость в доступе к ресурсам компьютеров, расположенных в одной и той же сети. Существенную помощь в этом может оказать пакет программ, разработанный для UNIX-совместимых операционных систем, который называется Samba. Этот пакет включает программное обеспечение для системы «клиент-сервер», которая позволяет осуществлять доступ к распределенным ресурсам в сетях UNIX и Windows. С помощью Samba файлы и принтеры компьютеров с UNIX становятся доступными для клиентов Windows, и наоборот. Пакет программ Samba поддерживает протокол SMB (Session Message Block), который является составной частью сетевого программного обеспечения всех компьютеров с установленными операционными системами Windows. В Windows-компьютерах поддержка SMB осуществляется на уровне протокола NetBIOS.

Если сервер Samba запущен в Linux, то клиенты Windows могут получить доступ к файловой системе Linux. Точно так же клиент Samba в Linux может получить доступ к ресурсам на компьютере с Windows, используя UNC-имена. Пакет Samba может быть применен для решения различных задач. В локальных сетях (LAN) пользователь может задействовать эту программу для копирования/перемещения файлов между сервером Linux и клиентом Windows. Другим примером использования Samba является удаленное редактирование содержимого веб-сайта на веб-сервере Apache. Кроме обычных операций копирования/перемещения, клиенты SMB могут выполнять редактирование удаленных файлов.

Для доступа к ресурсам Linux-компьютера с работающим сервером Samba клиенту Windows нужно поискать сетевой ресурс в папке **Сетевое окружение (Network Neighborhood)** (рис. 5.8).

Для доступа к распределенному ресурсу (пусть это будет, например, диск D) на Windows-компьютере с именем localpc в командной строке Linux-хоста нужно набрать команду **smbclient** с соответствующими параметрами. Результат выполнения команды показан ниже:

```
pi@raspberrypi / $ smbclient \\\\localpc\\d$ -U admin
Enter admin's password:
Domain=[LOCALPC] OS=[Windows Vista (TM) Ultimate 6002 Service
```

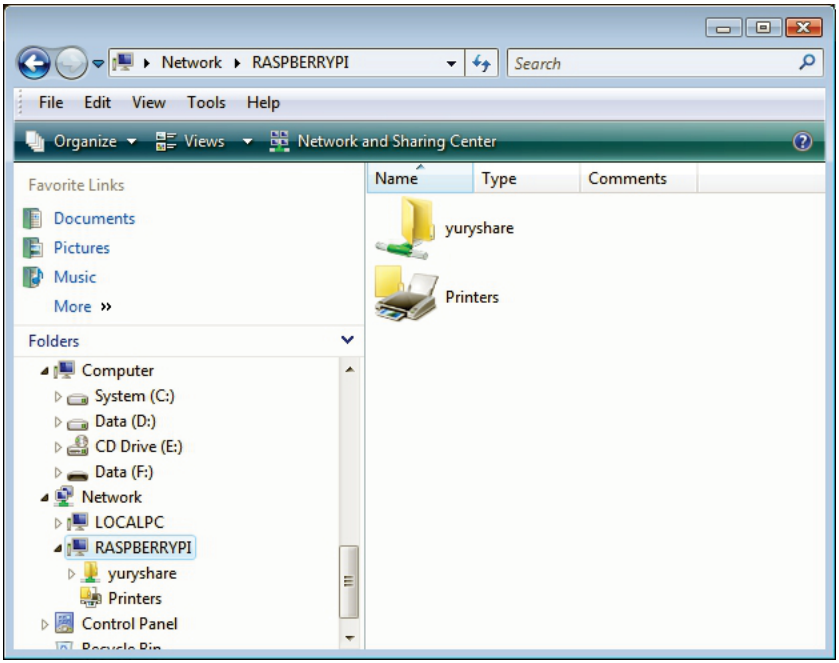

Рис. 5.8

```
Pack 2] Server=[Windows Vista (TM) Ultimate 6.0]
smb: \>
```

В качестве параметра U этой команде вводится имя пользователя Windows-компьютера (в данном случае admin). При выполнении этой команды от пользователя потребуется ввести пароль пользователя Windows (admin), под именем которого выполняется доступ к ресурсу.

Доступ к ресурсам Windows можно осуществить из графического интерфейса пользователя Raspbian OS. Для этого подойдет программа менеджера файлов (File Manager). В меню **Go** нужно выбрать опцию **Network Drives** (рис. 5.9).

Затем из раскрывшегося списка сетевых хостов выбрать Windows-компьютер (в данном примере это LOCALPC) (рис. 5.10).

Если сделать двойной щелчок на LOCALPC, то система потребует идентификацию пользователя на Windows-хосте (рис. 5.11).

В нашем примере таким пользователем является admin, поэтому в поле **Username** следует ввести это имя, а в поле **Password** – па-

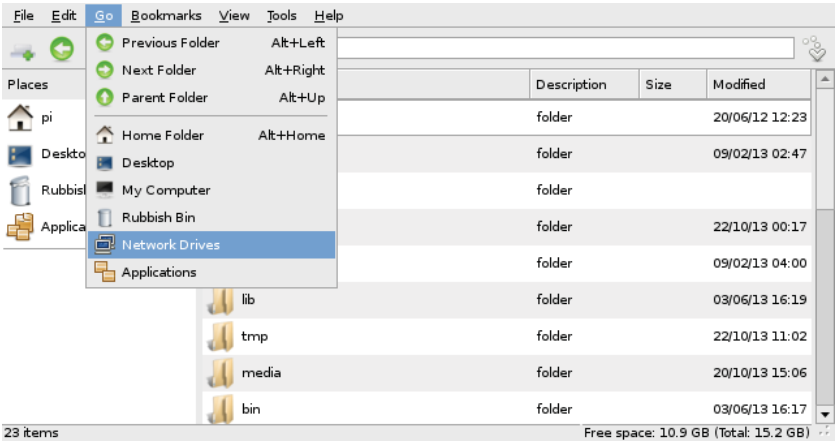

Рис. 5.9

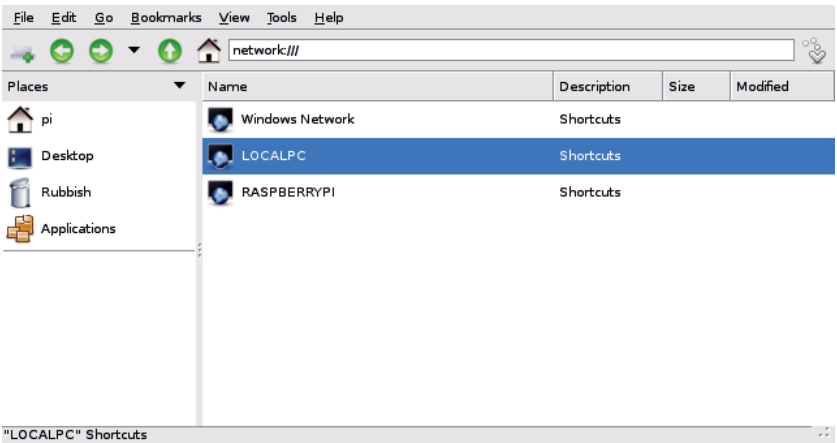

Рис. 5.10

роль на вход данного пользователя. После успешной идентификации в окне менеджера файлов появится список доступных ресурсов на Windows-компьютере (рис. 5.12).

Для доступа к файлам на любом из показанных дисков (C$, D$ или ADMIN$) также потребуется аутентификация пользователя с вводом пароля.

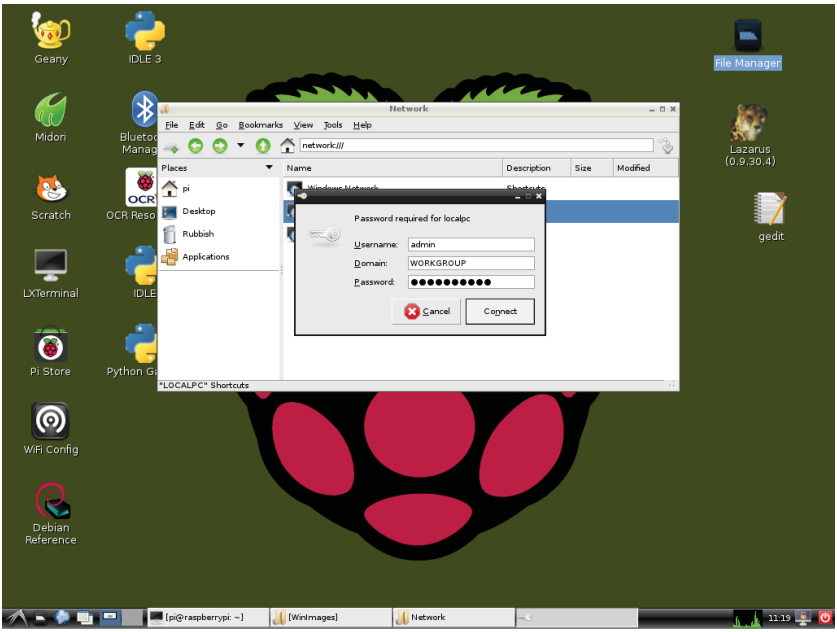

Рис. 5.11

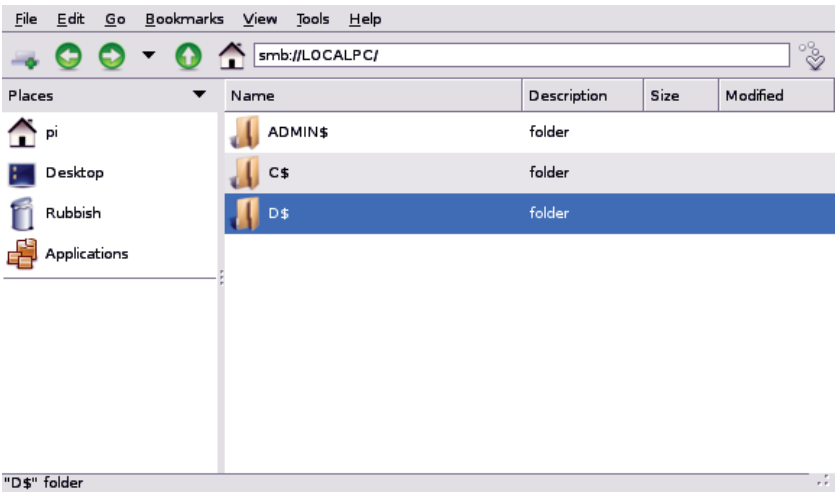

Рис. 5.12

Теперь рассмотрим более подробно, каким образом нужно сконфигурировать программное обеспечение Samba в операционной системе Raspbian OS.

На первом шаге нужно установить (если еще не установлено) пакет программ Samba. Для этого следует выполнить команду

```
$ sudo apt-get install samba samba-common-bin
```

Эта команда, помимо клиент-серверных программ, также установит дополнительные программы и инструменты настройки.

На втором шаге укажем Samba, что аутентификация пользователя осуществляется без проверки пароля:

```
$ sudo smbpasswd -an nobody
```

Следует помнить, что это действие ослабляет защиту системы, поэтому в реальной жизни аутентификация обязательно должна включать проверку пароля.

На шаге 3 нужно выполнить конфигурирование сервиса **smbd**, для чего нужно внести некоторые изменения в файл конфигурации /etc/samba/smb.conf. В этом файле содержатся настройки, которые определяют, какие файлы, каталоги и принтеры должны быть доступны, а также атрибуты доступа.

Для редактирования файла /etc/samba/smb.conf его нужно открыть текстовым редактором, например gedit или nano с правами суперпользователя root (для этого можно использовать команду **sudo**).

```
$ sudo nano /etc/samba/smb.conf
```

Затем в секции **[global]** нужно найти раздел **Authentication** и установить следующие настройки:

```
[global]
security = user
guest account = nobody
map to guest = bad password
```

Вот так может выглядеть отредактированный раздел **[global]** после редактирования:

```
[global]
...

####### Authentication #######

# "security = user" is always a good idea. This will
```

```
# require a Unix account in this server for every user
# accessing the server. See /usr/share/doc/samba-doc/
# htmldocs/Samba3-HOWTO/ServerType.html in the samba-doc
# package for details.
#    security = user

# You may wish to use password encryption.   See the
# section on 'encrypt passwords' in the smb.conf(5) manpage
# before enabling.
    encrypt passwords = true

# If you are using encrypted passwords, Samba will need to
# know what password database type you are using.
    passdb backend = tdbsam

    obey pam restrictions = yes

# This boolean parameter controls whether Samba attempts to
# sync the Unix password with the SMB password when the
# encrypted SMB password in the passdb is changed.
    unix password sync = yes

   guest account = nobody

# This option controls how unsuccessful authentication
# attempts are mapped to anonymous connections
    map to guest = bad user
```

На шаге 4 необходимо указать ресурс, к которому разрешается доступ. В данном случае соответствующая секция будет выглядеть так:

```
[yuryshare]
   comment = HOME Share
   browseable = yes
   path = /home/pi
   public = yes
   writable = yes
   guest ok = yes
```

Здесь предоставляется доступ по записи/чтению к каталогу /home/pi. Это означает, что пользователь сети может создавать и редактировать файлы в этом каталоге.

После внесенных изменений нужно сохранить содержимое файла /etc/samba/smb.conf и выполнить перезапуск сервиса Samba:

```
pi@raspberrypi ~ $ sudo service samba stop
[ ok ] Stopping Samba daemons: nmbd smbd.
pi@raspberrypi ~ $ sudo service samba start
[ ok ] Starting Samba daemons: nmbd smbd.
```

6 Программирование на языке Scratch в Raspberry Pi

Для программирования с RPi можно использовать самые разные языки программирования в зависимости от квалификации и опыта пользователя. В этой главе рассматривается очень популярный язык программирования, разработанный для начинающих, который называется Scratch. Этот язык позволяет быстро разрабатывать графические приложения, включая различные игры, используя в качестве программных конструкций набор графических блоков.

Каждый такой блок является визуальным представлением операторов, присутствующих в языках высокого уровня. Использование такого подхода к изучению программирования обладает целым рядом неоспоримых преимуществ. В частности, новичкам он существенно упрощает изучение основных концепций программирования (операторов, логических конструкций и циклов) и позволяет быстро создавать довольно сложные приложения с графическим интерфейсом.

Пиктограмму Scratch можно легко обнаружить на рабочем столе операционной системы Raspbian, поэтому пользователю не нужно устанавливать эту программу; для начала работы со Scratch достаточно щелкнуть мышью на пиктограмме. Перед тем как создать наше собственное приложение на языке Scratch, вкратце ознакомимся с элементами языка.

Программы на Scratch состоят из графических блоков, подписи к которым зависят от выбранного для интерфейса языка (поддерживаются 50 языков интерфейса, включая русский). Основными компонентами Scratch-программы являются объекты-спрайты. Спрайт состоит из графического представления – набора кадров-костюмов (costume) и скрипта. Редактирование костюмов спрайтов в Scratch осуществляется с помощью встроенного графического редактора (Paint Editor). Программа выполняется в специально выделенной области (stage) размером 480×360 пикселей с центром координат в середине области.

Для создания скриптов (программ) в языке Scratch нужно перетащить необходимые блоки из окна блоков в область скриптов. Функционально блоки разделены на 8 групп, каждая из которых обозначается своим собственным цветом. В табл. 6.1 даются расшифровка групп и цвет выделения.

Многие блоки имеют редактируемое белое поле для ввода параметров. Блоки делятся на три вида: стека, заголовков и ссылок.

На графическом представлении блоков стека (Stack Blocks) сверху имеется выемка, а внизу – выступ, что позволяет объединять блоки в группу блоков, именуемую стеком. Стеки можно копировать

Таблица 6.1. Группы блоков и их расшифровка

Обозначение	Описание	Цвет	Примечание
Movement	Перемещение	Синий	
Looks	Вид	Сиреневый	
Sound	Звуковые эффекты	Розовый	
Pen	Рисование	Зеленый	
Control	Управляющие структуры	Желтый	Управляющие конструкции, заголовки обработчиков событий
Sensing	Чтение ввода данных	Голубой	
Variables	Переменные	Оранжевый	

и перемещать как единый блок. Одной из разновидностей блоков стека являются управляющие конструкции, которые называются циклами. Графически цикл представлен в форме литеры С. В такой С-образной форме могут содержаться вложенные блоки.

В блоках заголовков (Hats) присутствует выпуклый верхний край и выступ для объединения снизу – это заголовки стеков. К таким блокам можно отнести блоки when (если) из группы Control, позволяющие организовать обработку сообщений от клавиатуры и мыши, а также для событий, инициируемых самими спрайтами.

Блоки ссылок (Reporters) используются для заполнения внутренних полей других блоков числами, текстовыми строками, логическими значениями, а также списками в форме динамических массивов.

Несколько слов об интерфейсе пользователя языка Scratch. Главная форма разделена на три секции. Все необходимые блоки для создания приложений находятся в секции слева, в центральной секции, собственно, и происходит создание приложения, а в правой секции программа запускается на выполнение (рис. 6.1).

Как видно по рисунку, в левом верхнем углу размещены группы блоков (**Motion**, **Looks**, **Sound**, **Pen**, **Control**, **Sensing**, **Operators** и **Variables**), которые будут использоваться для создания демонстрационного приложения на Scratch.

В нашем демонстрационном приложении кот будет охотиться за мышью, поэтому нам понадобятся как минимум два спрайта, представляющие этих персонажей. Разработаем шаг за шагом наше игровое приложение на языке Scratch.

На первом шаге создадим спрайт мыши. Для этого просто заменим спрайт кошки, выбрав рисунок мыши из **Costumes** > **Import** > **Animals** > **Mouse 1** (рис. 6.2).

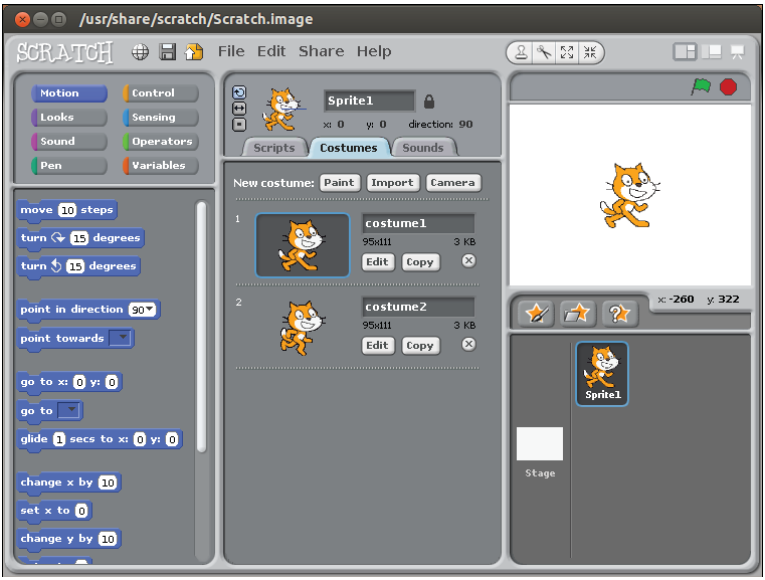
/usr/share/scratch/Scratch.image
SCRATCH
File Edit Share Help
Motion
Control
Looks
Sensing
Sound
Operators
Pen
Variables
move 10 steps
turn 15 degrees
turn 15 degrees
point in direction 90
point towards
go to x: 0 y: 0
go to
glide 1 secs to x: 0 y: 0
change x by 10
set x to 0
change y by 10
Sprite1
x: 0 y: 0 direction: 90
Scripts
Costumes
Sounds
New costume: Paint Import Camera
costume1
95x111 3 KB
Edit Copy
costume2
95x111 3 KB
Edit Copy
x: -260 y: 322
Sprite1
Stage

Рис. 6.1

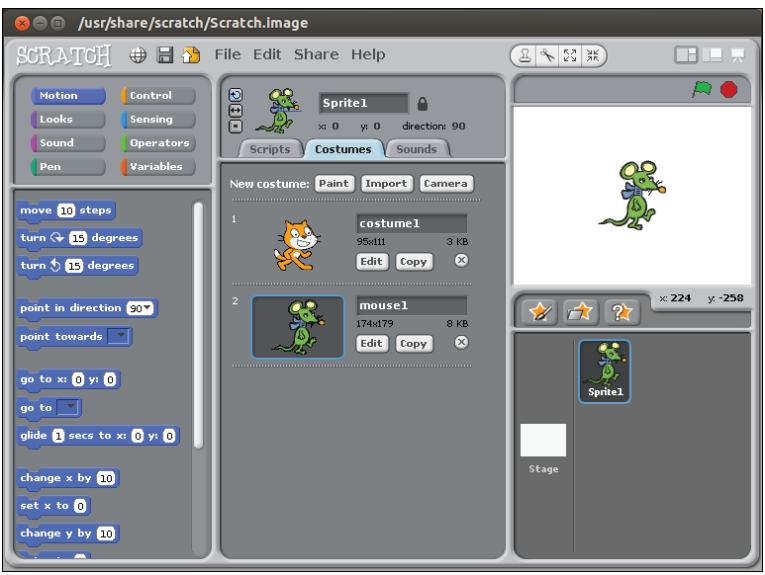
/usr/share/scratch/Scratch.image
SCRATCH
File Edit Share Help
Motion
Control
Looks
Sensing
Sound
Operators
Pen
Variables
move 10 steps
turn 15 degrees
turn 15 degrees
point in direction 90
point towards
go to x: 0 y: 0
go to
glide 1 secs to x: 0 y: 0
change x by 10
set x to 0
change y by 10
Sprite1
x: 0 y: 0 direction: 90
Scripts
Costumes
Sounds
New costume: Paint Import Camera
costume1
95x111 3 KB
Edit Copy
mouse1
174x179 8 KB
Edit Copy
x: 224 y: -258
Sprite1
Stage

Рис. 6.2

Теперь можно уменьшить размер спрайта, нажав на иконку **Shrink sprite** (в форме круга).

Далее выполним настройки клавиатуры (в нашей игре будут использоваться клавиши ↑ и ↓ для навигации во время игры, а также клавиша **r** для запуска новой игры).

Перейдем в секцию **Scripts** и заменим **When Right Arrow Key Pressed** на **When r Key Pressed** – это позволит запускать новую игру при нажатии клавиши **r** (рис. 6.3).

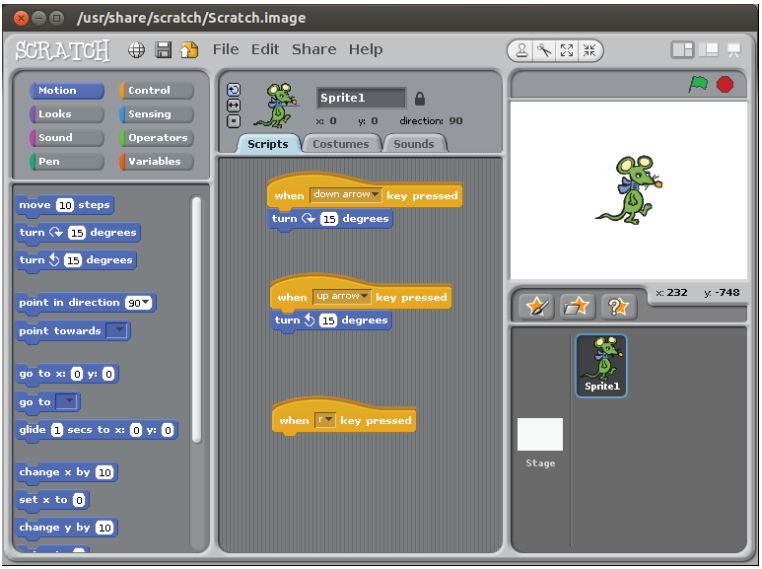

Рис. 6.3

На следующем этапе нам необходимо будет создать несколько переменных. Для этого щелкнем на группе **Variables** в левом верхнем углу, затем выберем **Make a variable**, после чего присвоим переменной имя **score** (рис. 6.4). Затем точно так же создадим переменную **over**.

Для сброса результата игры и обновления экрана добавим под скриптом **When r Key Presses** строку **show** (группа **Looks**), **Go To X:100, Y:100** (группа **Motion**), **Set Score To 0** and **Set Over to 0** (**Variables**) (рис. 6.5).

На следующем шаге добавим сообщение о начале игры – это позволит другому спрайту определить момент начала игры. Для этого


/usr/share/scratch/Scratch.image
SCRATCH
File Edit Share Help
Motion
Control
Looks
Sensing
Sound
Operators
Pen
Variables
Make a variable
Make a list
Sprite1
x: 0 y: 0 direction: 90
Scripts
Costumes
Sounds
when down arrow key pressed
turn 15 degrees
when r key pressed
?
Variable name?
score
For all sprites
For this sprite only
OK
Cancel
x: 236 y: 74
Sprite1
Stage

Рис. 6.4

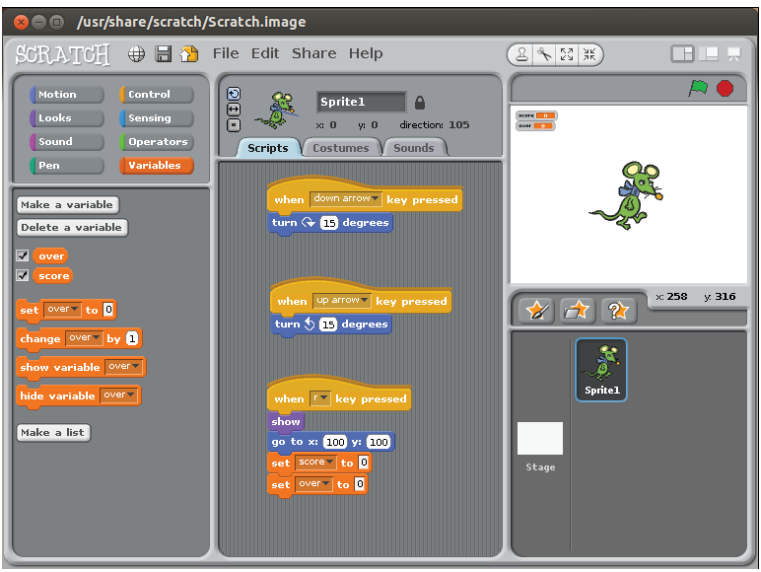
/usr/share/scratch/Scratch.image
SCRATCH
File Edit Share Help
Motion
Control
Looks
Sensing
Sound
Operators
Pen
Variables
Make a variable
Delete a variable
over
score
set over to 0
change over by 1
show variable over
hide variable over
Make a list
Sprite1
x: 0 y: 0 direction: 105
Scripts
Costumes
Sounds
when down arrow key pressed
turn 15 degrees
when up arrow key pressed
turn 15 degrees
when r key pressed
show
go to x: 100 y: 100
set score to 0
set over to 0
x: 258 y: 316
Sprite1
Stage

Рис. 6.5

перетащим блок **Broadcast...** вниз под скрипт **When r Key Pressed**, а затем в выпадающем меню выберем **New...** и присвоим сообщению имя «**start**» (рис. 6.6).

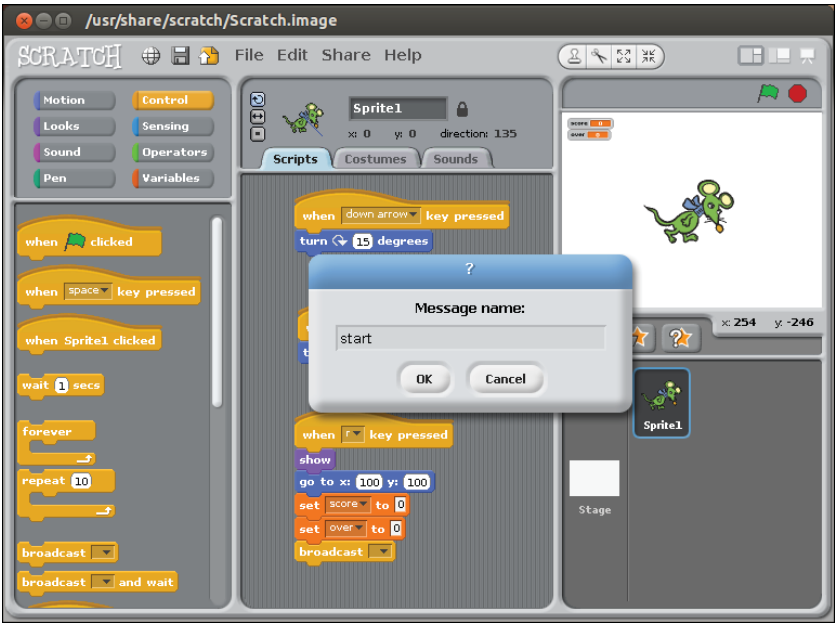

Рис. 6.6

Для повторного выполнения приложения, равно как и отдельных его частей, необходимо организовывать циклы, которые позволяют выполнять один и тот же код многократно. Модифицируем наш скрипт, добавив в него блок **Repeat Until...** (группа **Control**), и определим для него условия выполнения. Для этого перетащим оператор присвоения (равенства) **=** (группа **Operators**), после чего поместим **Over** (**Variables**) слева от знака равенства. Справа от **=** введем значение 1 (рис. 6.7).

Далее поместим операторы цикла внутрь тела цикла **Repeat Until Over = 1**. Добавим блок **Change score By 1** (группа **Variables**), блок **Move 7 Steps** (группа **Motion**) и **If On Edge, Bounce** (группа **Motion**). Эти три блока кода будут выполняться до тех пор, пока переменная **over** не станет равной 1 (рис. 6.8).

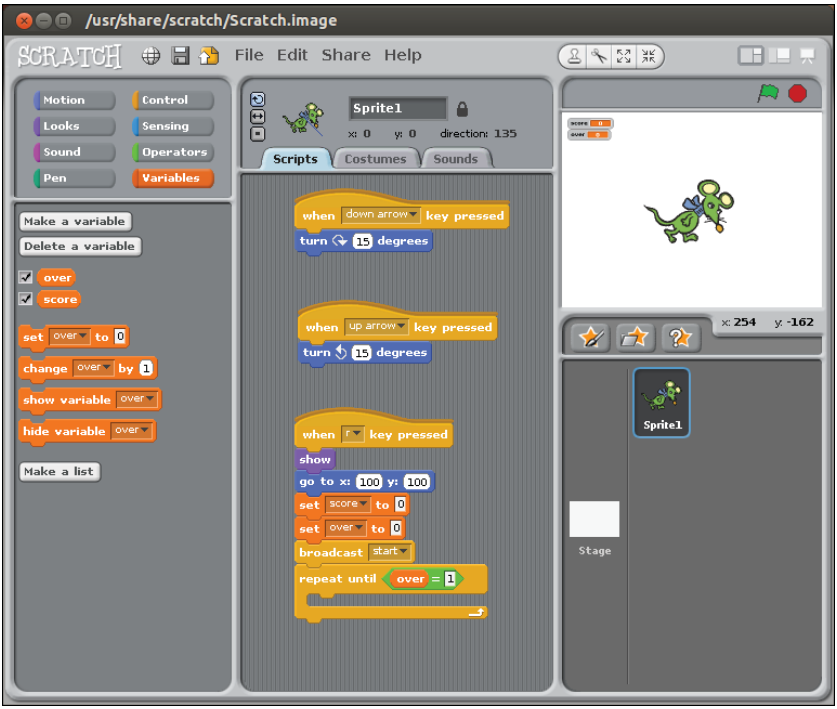

Рис. 6.7

Как только игра закончилась (кошка поймала мышь), цикл **Repeat Until** также заканчивается, выполнение программы будет продолжено внизу цикла. Перетащим блок **Hide** (группа **Looks**) и поместим его под циклом. Этот блок позволяет скрыть спрайт мыши по окончании игры (рис. 6.9).

Теперь займемся другим персонажем нашей игры, кошкой. Вначале выберем для нее подходящий спрайт (**Select Choose** > **New Sprite From File** > **Cat 4**) (рис. 6.10). Размеры этого (как и любого) спрайта также можно уменьшить или увеличить, как это мы сделали ранее для мыши. Каждый спрайт ассоциируется со своим собственным скриптом, поэтому при необходимости можно переходить от одного спрайта к другому. Для выбора спрайта нужно просто щелкнуть на нем мышью.

Наша кошка должна двигаться в поисках мыши, поэтому предоставим ей такую возможность. Создадим новый скрипт, добавив

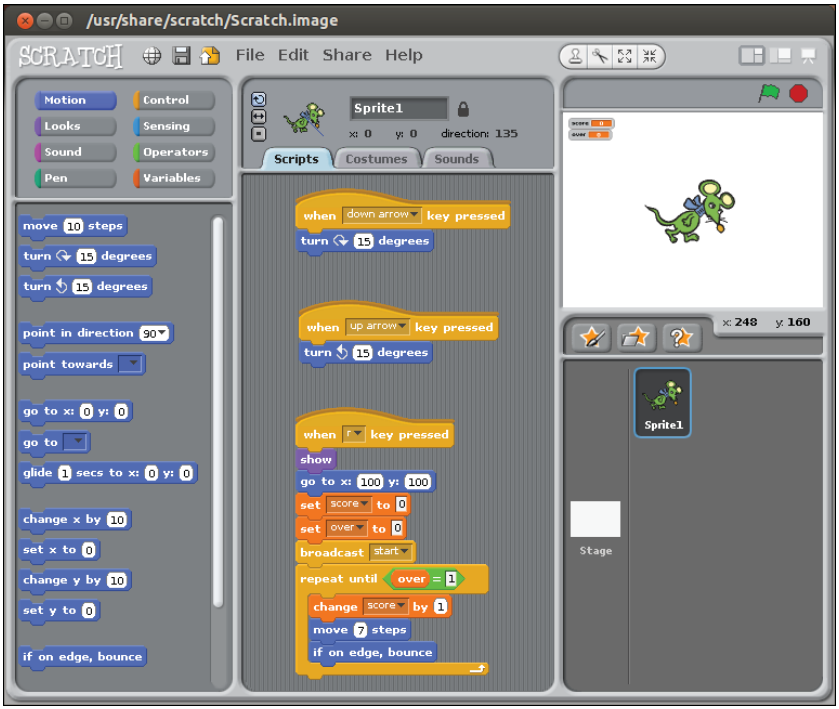

Рис. 6.8

в него блоки **When I Receive start** (группа **Control**) и **Go To X:-100 Y:-100** (рис. 6.11). Таким образом, в исходной позиции кошка будет находиться в противоположном от мыши углу экрана (вспомним, что координаты (0,0) соответствуют центру экрана).

Наша кошка должна непрерывно двигаться в поиске мыши, поэтому нам понадобится организовать цикл. Добавим блок **Repeat Until** (группа **Control**) и затем в пустое поле условия добавим блок **Touching Sprite 1** (группа **Sensing**). Таким образом, кошка (спрайт 2) будет перемещаться до тех пор, пока не поймает мышь (спрайт 1) (рис. 6.12).

В нашей игре мы можем также задавать уровень сложности. Для этого внутрь цикла **Repeat Until** добавим блок **Point Towards Sprite 1** (группа **Motion**) и **Move 4 Steps (Motion)** (рис. 6.13). Величина перемещения кошки и мыши в каждом из циклов определяет сложность достижения результата. Экспериментально установлено, что лучше всего значения шага будут находиться между 4 и 7.

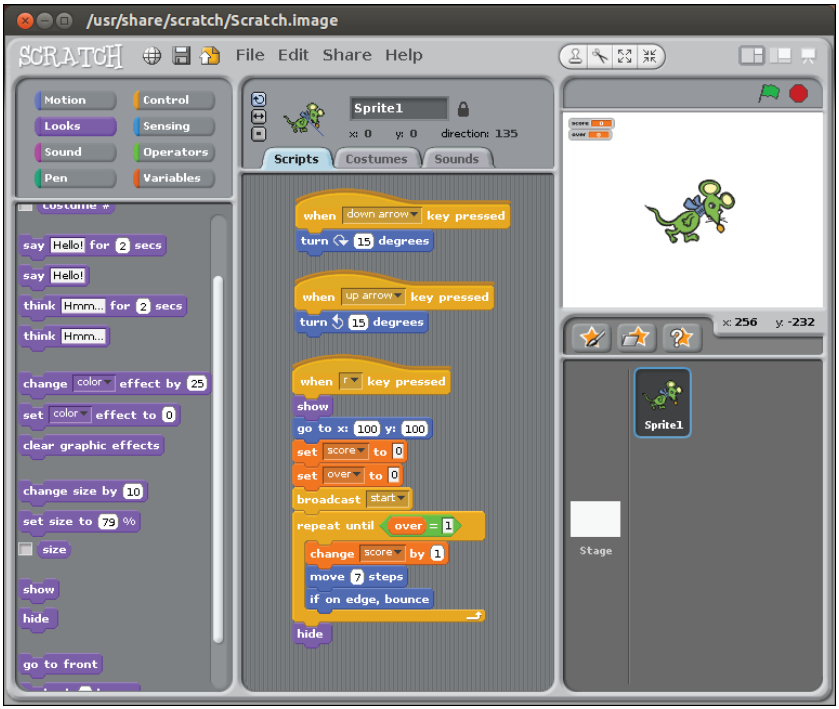

Рис. 6.9

Цикл будет завершен, когда кошка поймала мышь. В этом случае игра прекращается, поэтому выполнение скрипта для спрайта 1 также должно быть остановлено. Это легко можно сделать, добавив блок **Set over To 1** (группа **Variables**) под блок **Repeat Until** (рис. 6.14).

Кроме того, игрок также должен узнать об окончании игры. Это можно сделать, послав звуковое и текстовое уведомления, для чего следует добавить блок **Play Drum 1 for 1 Beats** (группа **Sound**), затем добавить **Say Mmmm Tasty For 1 Secs** (группа **Looks**) (рис. 6.15).

Наконец, наше приложение должно отобразить результат игры. У нас имеется переменная **score**, которая увеличивается на 1 каждый раз после прохождения очередного цикла, поэтому можно организовать вывод результата, добавив блок **Say You Scores ... For 1 Secs**, затем перетащить еще один блок **Say ... for 1 Secs**, в пустое поле которого поместить значение переменной **score** (группа **Variables**) (рис. 6.16).

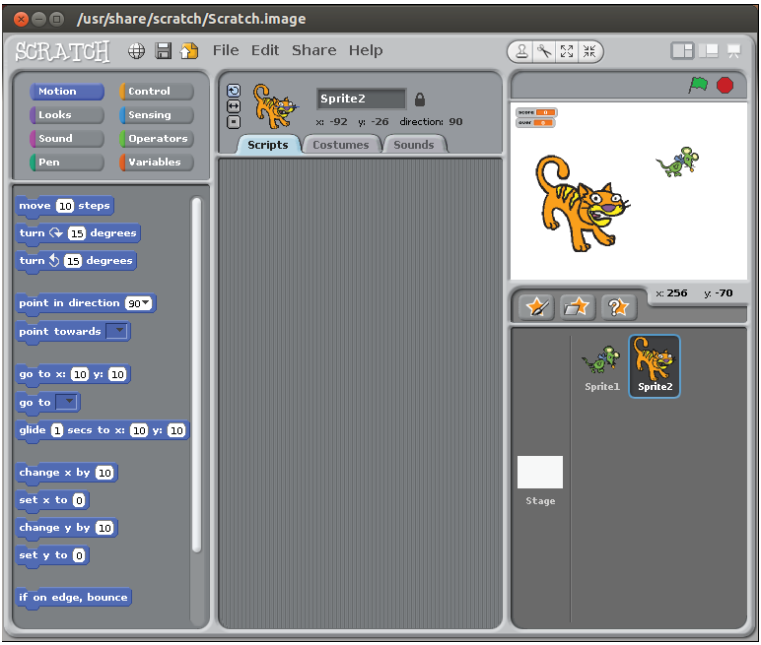

Рис. 6.10

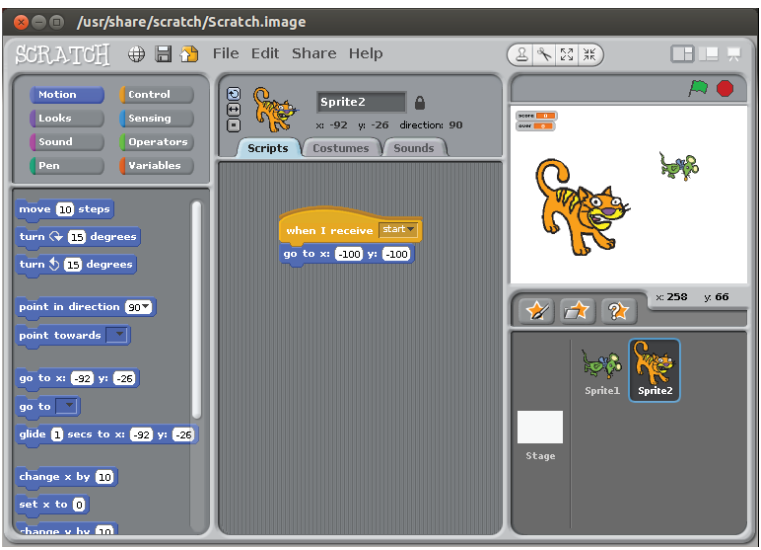

Рис. 6.11

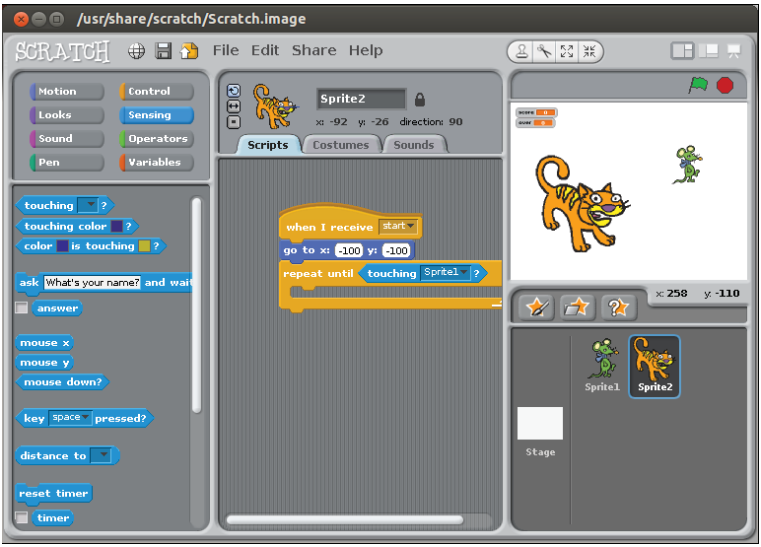

Рис. 6.12

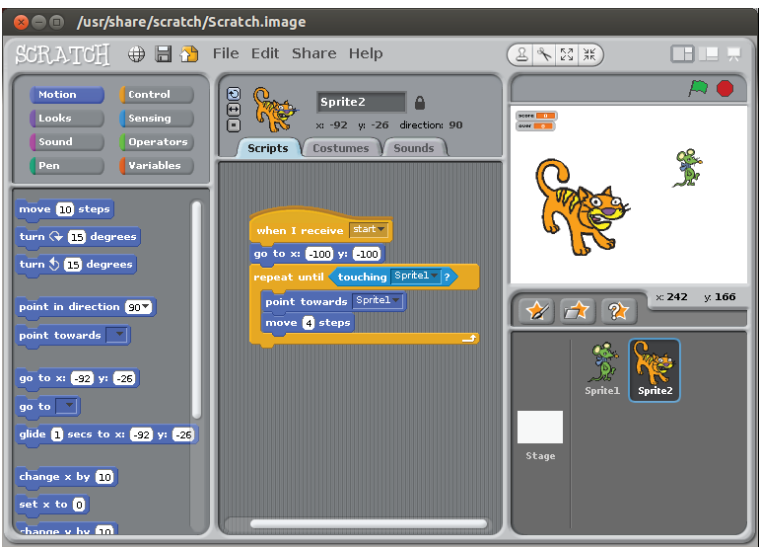

Рис. 6.13

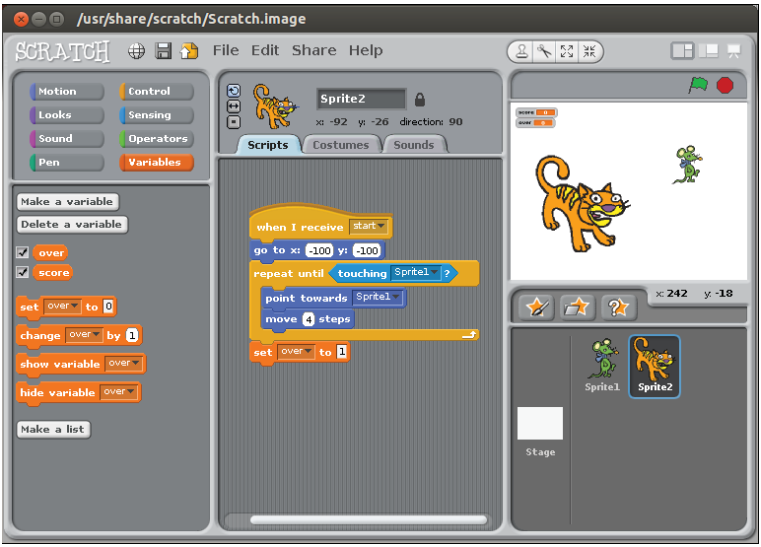

Рис. 6.14

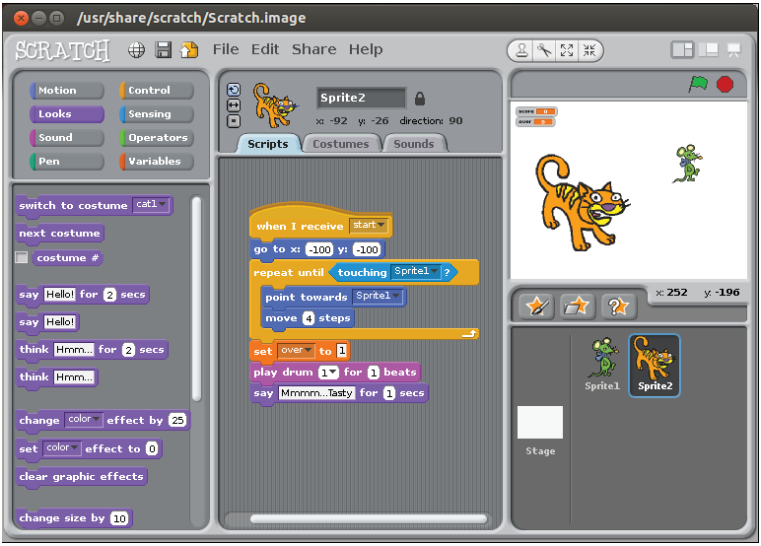

Рис. 6.15

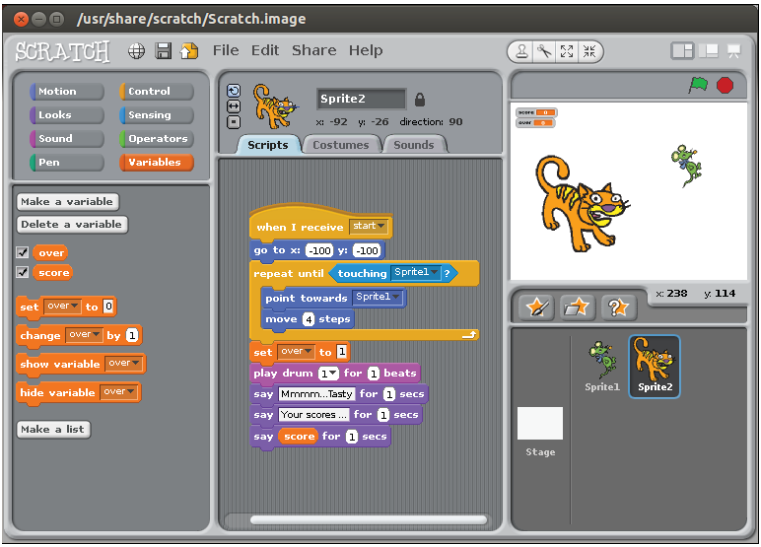

Рис. 6.16

Теперь можно запустить наше игровое приложение, нажав клавишу **r**. Для перемещения мыши нужно использовать клавиши ↑ и ↓. Фрагмент игры показан на рис. 6.17.

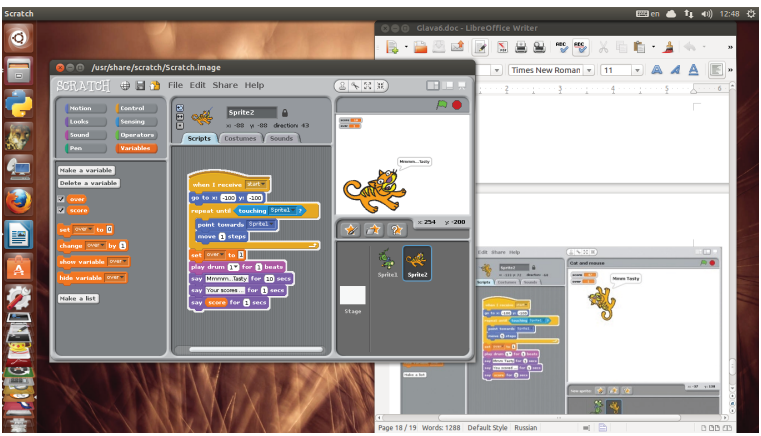

Рис. 6.17

7 Программирование приложений на языке Python в Raspbian OS

В этой главе рассматриваются основы программирования на языке Python, который является одним из наиболее популярных в настоящее время языков программирования. Python широко используется как для разработки самостоятельных программ, так и для создания прикладных сценариев в самых разных областях применения.

Язык Python является мощным, переносимым, простым в использовании и свободно распространяемым языком. Python включен в дистрибутив Raspbian OS для Raspberry Pi в двух версиях – 2.7.3 и 3.2.3. Версии 3.x.x языка имеют больше возможностей и содержат ряд дополнительных функций, по сравнению с версиями 2.x.x. Версии 3.x.x представляют новую ступень развития языка и во многом несовместимы с прежним программным кодом. В этой главе мы будем рассматривать базовые принципы, лежащие в основе версии 3.2.3 языка Python.

Для разработки программного кода примеров используется приложение IDLE 3, пиктограмма которого находится на рабочем столе Raspbian OS (рис. 7.1).

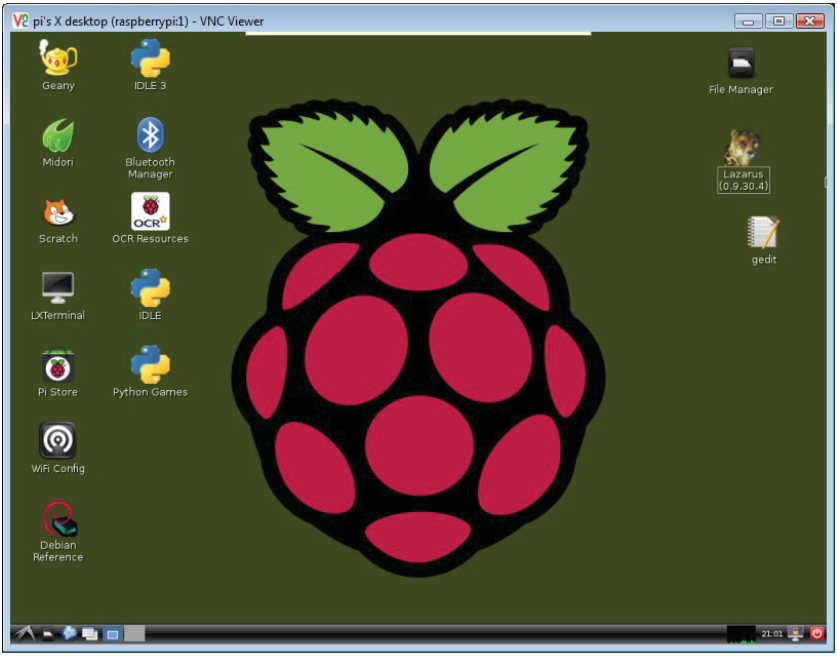

Рис. 7.1

При запуске приложения открывается окно редактирования, в верхней части которого отображаются номер версии и дата выпуска релиза (рис. 7.2).

Рис. 7.2

В IDLE 3 можно выполнять операторы языка пошагово, как показано на рис. 7.3.

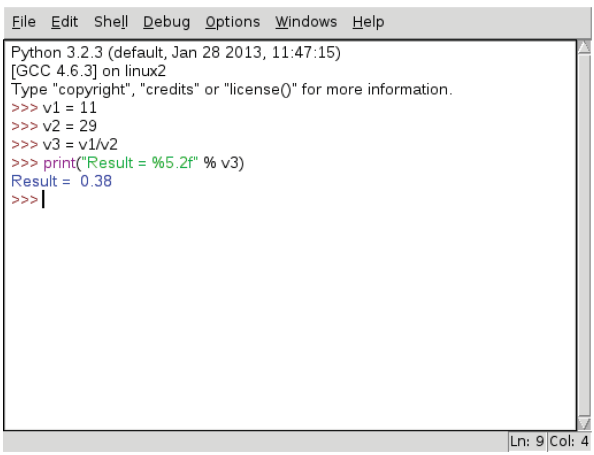

Рис. 7.3

Можно сохранить последовательность команд в отдельном файле с расширением .py и затем выполнить программу. Чтобы это сделать, нужно в меню **File** выбрать опцию **New Window** (рис. 7.4).

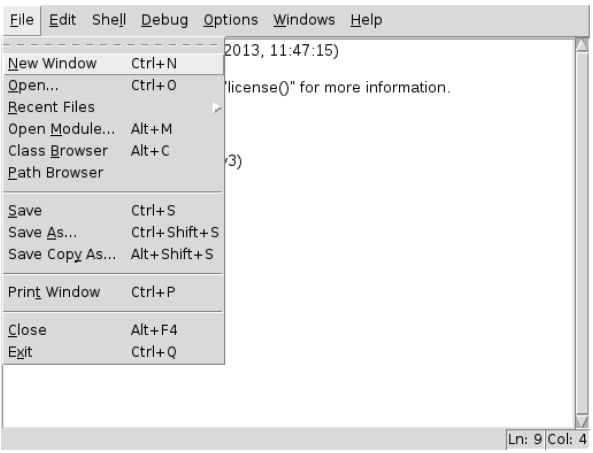

Рис. 7.4

В открывшемся окне редактирования можно набрать текст нашего приложения (рис. 7.5).

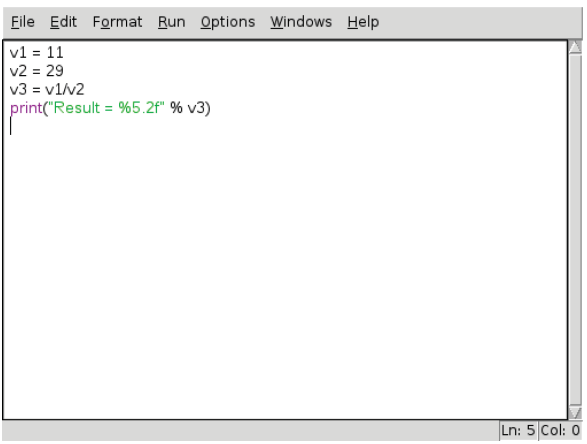

Рис. 7.5

Затем следует сохранить исходный текст программы в файле, например EXAMPLE1.py. Для выполнения нашего приложения нужно в меню **Run** выбрать опцию **Run Module** или просто нажать клавишу **F5** (рис. 7.6).

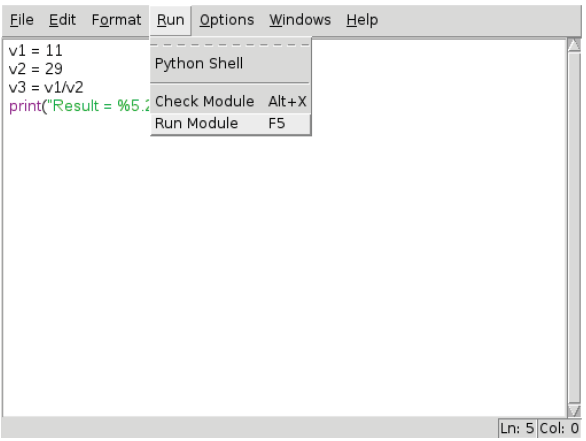

Рис. 7.6

Результат выполнения программы будет выведен в отдельном окне, как это иллюстрирует рис. 7.7.

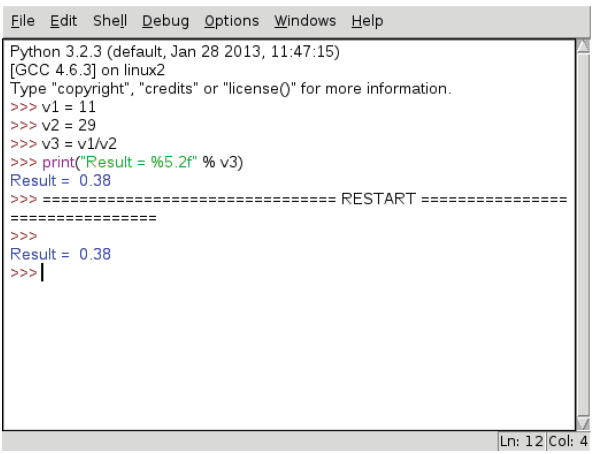

Рис. 7.7

Мы ознакомились со средой программирования IDLE 3. Далее вкратце рассмотрим базовые конструкции языка Python версии 3.x.x. Начнем с типов данных, используемых в программах.

Базовые типы данных языка Python представлены в табл. 7.1.

Таблица 7.1. Базовые типы данных языка Python

Тип объекта	Пример использования
Числа	1234, 3.1415, 3+4j, Decimal, Fraction
Строки	'spam', "guido's" , b'a\x01c'
Списки	[1, [2, 'three'], 4]
Словари	{'food': 'spam', 'taste': 'yum'}
Кортежи	(1,'spam', 4, 'U')
Файлы	myfile = open('eggs', 'r')
Множества	set('abc'), {'a', 'b', 'c'}
Прочие базовые типы	Сами типы, None, логические значения
Типы структурных элементов программ	Функции, модули, классы
Типы, имеющие отношение к реализации	Компилированный программный код, стек вызовов

Как видно из содержимого таблицы, в ней представлены некоторые встроенные типы объектов языка Python и некоторые синтаксические конструкции использования этих объектов в виде литералов – то есть выражения, которые генерируют данные объекты. Некоторые из этих типов, скорее всего, уже вам знакомы, особенно если ранее вам приходилось работать с другими языками программирования. Так, например, числа и строки представляют числовые и текстовые значения соответственно, а файлы обеспечивают интерфейс для работы с файлами, хранящимися в компьютере.

Типы объектов, показанные в табл. 7.1, обычно называют базовыми, потому что они встроены непосредственно в язык Python, то есть для создания большинства из них используется вполне определенный синтаксис. Например, когда выполняется следующий программный код:

```
>>> 'spam'
```

то выполняется выражение-литерал, которое генерирует и возвращает новый строковый объект. Подобным образом выражение, заключенное в квадратные скобки, создает список; выражение, заключенное в фигурные скобки, – словарь и т. д. Хотя в языке Python и отсутствует конструкция объявления типа, сам синтаксис

выполняемых выражений задает типы создаваемых и используемых объектов.

Как только создается объект, он ассоциируется со своим собственным набором операций на протяжении всего времени существования. Это означает, например, что над строками можно будет выполнять только строковые операции, над списками – только операции, применимые к спискам. В языке Python используется динамическая типизация, когда типы данных определяются автоматически и их не требуется объявлять в программном коде. Тем не менее Python является языком со строгой типизацией, поскольку над объектом можно выполнять только те операции, которые применимы к его типу.

Рассмотрим более подробно типы данных языка и начнем с чисел. Базовый набор объектов языка Python включает в себя целые числа (числа без дробной части), вещественные числа (числа с десятичной точкой) и более сложные типы (комплексные числа с мнимой частью, числа с фиксированной точностью, рациональные числа и множества).

Числовые данные в языке Python поддерживают набор стандартных математических операций, таких как сложение (символ «плюс» (+)), умножение (символ «звездочка» (*)), деление (наклонный слеш вправо (/)), возведение в степень (два символа «звездочка» (**)).

Вот несколько примеров операций над числами:

```
>>> 123 + 222 # Целочисленное сложение
345
>>> 1.5 * 4 # Умножение вещественных чисел
6.0
>>> 2 ** 100 # 2 в степени 100
1267650600228229401496703205376
```

Обратите внимание на результат последней операции: в языке Python версии 3.x.x целые числа автоматически обеспечивают неограниченную точность для представления больших значений (в отличие от Python 2.x.x, где для представления больших целых чисел имеется отдельный тип длинных целых чисел).

Помимо выражений для выполнения операций с числами, в составе Python имеется несколько полезных модулей, например **math**:

```
>>> import math
>>> math.pi
3.1415926535897931
```

```
>>> math.sqrt(85)
9.2195444572928871
>>> math.pow  (3,  2)
9.0
>>> math.sin(math.pi/4)
0.7071067811865475
```

Модуль **math** содержит более сложные математические функции, а модуль **random** реализует генератор случайных чисел и функцию случайного выбора, в данном случае из списка:

```
>>> import  random
>>> random.random()
0.4575646056534337
>>> random.choice([7,  -9,  12,  3])
-9
```

Теперь перейдем к строкам. Строки используются для записи текстовой информации, включая произвольные последовательности байтов. Последовательности поддерживают порядок размещения своих элементов слева направо. При этом элементы последовательности обрабатываются в соответствии с их позициями (индексами). Строки являются последовательностями односимвольных строк.

Строки поддерживают операции в соответствии с порядком позиционирования элементов. Если, например, имеется строка

```
>>> str  =  'String1'
```

то с помощью встроенной функции **len** можно определить ее длину, а отдельные элементы строки извлечь с помощью выражений индексирования:

```
>>> len(str)
7
>>> str[1]
't'
```

В языке Python индексы реализованы в виде смещений от начала, и индексация начинается с 0, поэтому первый элемент имеет индекс 0, второй – 1 и т. д.

Обратите внимание, как в данном примере выполняется присваивание строки переменной с именем **str**. В языке Python не требуется объявлять переменные заранее.

Переменная создается в тот момент, когда ей присваивается значение, при этом переменной можно присвоить значение любого

типа, а при использовании внутри выражения имя переменой замещается ее фактическим значением. Для обращения к переменной ей предварительно должно быть присвоено какое-либо значение.

В языке Python предусмотрена возможность индексации в обратном порядке, от конца к началу – положительные индексы отсчитываются от левого конца последовательности, а отрицательные – от правого:

```
>>> str[-1]
'1'
>>> str[-3]
'n'
```

В дополнение к простой возможности индексирования по номеру позиции последовательности поддерживают более общую форму индексирования, позволяющую получить часть строки (подстроку), как в следующем примере:

```
>>> str[2:5]
'rin'
```

Обратите внимание, что в качестве границ подстроки можно использовать отрицательные индексы, например:

```
>>> str[-3:-1]
'ng'
```

Строки поддерживают операцию конкатенации (соединения), которая обозначается знаком + (плюс). Вот несколько примеров операции конкатенации:

```
>>> str1 = 'My'
>>> str1 + str
'MyString1'
>>> str1
'My'
>>> str
'String1'
>>> str2 = str1 + str
>>> str2
'MyString1'
>>> str2 * 2
'MyString1MyString1'
```

Здесь знак плюс (+) имеет иной смысл, чем для операций с числами. Это свойство языка Python называется полиморфизмом и озна-

чает, что фактически выполняемая операция зависит от объектов, которые в ней задействованы.

Следует обратить внимание на то, что во всех предыдущих примерах ни одна из использованных операций не изменяла оригинальную строку. Все операции над строками в результате создают новую строку, потому что строки в языке Python являются неизменяемыми. Однажды созданную строку нельзя изменить. Это означает, что нельзя изменить строку присвоением значения одной из ее позиций, хотя всегда можно создать новую строку и присвоить ей то же самое имя.

Далее мы рассмотрим очень важный тип объектов языка Python – файлы. Объекты-файлы – это основной интерфейс между программным кодом на языке Python и дисковыми файлами, хранящимися на компьютере. Файлы являются одним из базовых типов, которые создаются специфичным способом.

Для создания объекта файла необходимо вызвать встроенную функцию **open**, передав ей имя внешнего файла и строку режима доступа к файлу. Следующий пример иллюстрирует эту концепцию. Здесь в текущем каталоге вначале создается файл для вывода данных, при этом в качестве одного из параметров указывается режим обработки файла (запись '**w**'). Еще одним параметром является имя файла, которое может содержать полный путь к каталогу, если требуется получить доступ к файлу, находящемуся в другом месте.

```
>>> fd = open('Hello.txt', 'w')
>>> str = 'Hello, world'
>>> fd.write(str)
>>> fd.close()
```

Для чтения содержимого только что записанного файла нужно открыть его в режиме чтения ('**r**') (этот режим используется по умолчанию, если данный параметр не указан). Далее содержимое текстового файла присваивается строке, которая затем отображается на экране. В следующем примере демонстрируются чтение файла и отображение данных:

```
>>> fd = open('Hello.txt', 'r')
>>> txt = fd.read()
>>> txt
'Hello, world'
```

На этом краткое знакомство с типами данных и основными методами их обработки мы закончим. Более подробную и полную ин-

формацию по данной теме можно найти в многочисленных книгах по программированию на языке Python.

Теперь мы ознакомимся с инструкциями (операторами) языка Python. Вначале рассмотрим общее представление синтаксиса инструкций, затем – конкретные инструкции детально.

Синтаксис языка Python базируется на инструкциях и выражениях. С помощью выражений обрабатываются объекты, а сами выражения встраиваются в инструкции. Инструкции представляют собой более крупные логические блоки программы – они напрямую используют выражения для обработки объектов. Кроме того, инструкции сами создают объекты (например, инструкции присваивания).

Основные инструкции языка Python показаны в табл. 7.2.

Таблица 7.2. Основные инструкции языка Python

Инструкция	Действие	Пример
Присваивание	Создание ссылок	a, *b = 'good', 'bad', 'ugly'
Вызовы и другие выражения	Запуск функций	log.write("spam, ham")
Вызов функции print	Вывод объектов	print('The Killer', joke)
if/elif/else	Операция выбора	if "python" in text: print(text)
for/else	Обход последовательности в цикле	for x in mylist: print(x)
while/else	Циклы общего назначения	while X > Y: print('hello')
pass	Пустая инструкция-заполнитель	while True: pass
break	Выход из цикла	while True: if exittest(): break
continue	Переход в начало цикла	while True: if skiptest(): continue
def	Создание функций и методов	def f(a, b, c=1, *d): print(a+b+c+d[0])
return	Возврат результата	def f(a, b, c=1, *d): return a+b+c+d[0]
yield	Функции-генераторы	def gen(n): for i in n: yield i*2
global	Пространства имен	x = 'old' def function(): global x, y; x = 'new'

nonlocal	Пространства имен (3.0+)	def outer():
		x = 'old'
		def function():
		nonlocal x; x = 'new'

Таблица 7.2 (окончание)

Инструкция	Действие	Пример
import	Доступ к модулям	import sys
from	Доступ к атрибутам модуля	from sys import stdin
class	Создание объектов	class Subclass(Superclass): staticData = [] def method(self): pass
try/except/finally	Обработка исключений	try: action() except: print('action error')
raise	Возбуждение исключений	raise endSearch(location)
assert	Отладочные проверки	assert X > Y, 'X too small'
with/as	Менеджеры контекста (2.6+)	with open('data') as myfile: process(myfile)
del	Удаление ссылок	del data[k] del data[i:j] del obj.attr del variable

Одним из новых синтаксических элементов в языке Python является символ двоеточия (`:`). Все составные инструкции в языке Python, то есть инструкции, которые включают вложенные в них инструкции, записываются в соответствии с одним и тем же общим шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, обычно с отступом под строкой основной инструкции.

Двоеточие является обязательным, а его отсутствие является самой распространенной ошибкой, которую допускают начинающие программисты. В большинстве текстовых редакторов с выделением синтаксиса эту ошибку легко заметить, хотя с приобретением опыта такая ошибка исчезает сама по себе.

В языке Python круглые скобки не обязательны, хотя для C-подобных языков они являются обязательным атрибутом операторов. Например, в инструкции

```
if (x < y)
```

языка C круглые скобки являются обязательными. В языке Python скобки можно опустить:

```
if x < y
```

Тем не менее каждое выражение может быть заключено в скобки, но их присутствие не будет противоречить синтаксису языка Python, и они не будут считаться ошибкой. Стиль языка Python характеризуется тем, что скобки вообще опускаются в подобных инструкциях.

Другой особенностью языка Python является то, что конец строки является концом инструкции.

Еще один, более важный синтаксический элемент, который пользователь не найдет в программном коде на языке Python, – это точка с запятой. В языке Python не требуется завершать инструкции точкой с запятой, как это делается в C-подобных языках, например в инструкции:

```
x = 1;
```

Общее правило в языке Python гласит, что конец строки автоматически считается концом инструкции, находящейся в этой строке. Иными словами, точку с запятой после инструкции можно опустить:

```
x = 1
```

В языке Python не нужно вставлять **begin**/**end**, **then**/**endif** или фигурные скобки вокруг вложенных блоков, как это делается в C-подобных языках:

```
if (x > y)
{
x = 1;
y = 2;
}
```

Для этих целей в языке Python используются отступы, когда все инструкции в одном и том же вложенном блоке оформляются с одинаковыми отступами от левого края. По ширине отступа интерпретатор определяет, где находится начало блока и где конец:

```
if x > y:
    x = 1
    y = 2
```

Под отступами в данном примере подразумевается пустое пространство слева от двух вложенных инструкций. Интерпретатор не накладывает ограничений на то, как выполняются отступы (можно использовать как символы пробела, так и символы табуляции), и на величину отступов (допускается использовать любое число пробелов или символов табуляции).

При этом ширина отступа для одного вложенного блока может существенно отличаться от ширины отступа для другого блока. По правилам синтаксиса требуется, чтобы все инструкции в пределах одного блока имели один и тот же отступ от левого края. Если это требование не выполняется, то интерпретатор генерирует синтаксическую ошибку, и программный код не будет работать, пока отступ не будет исправлен.

В программах на языке Python на каждой строке обычно располагается одна инструкция, хотя вполне возможно для большей компактности записать несколько инструкций в одной строке, разделив их точками с запятой, как в следующем примере:

```
a = 1; b = 2; print(a + b)
```

Это единственный случай, когда в языке Python необходимо использовать точки с запятой в качестве разделителей инструкций. Однако такой подход не применим к составным инструкциям. Это означает, что в одной строке можно размещать только простые инструкции наподобие присваивания, **print** и вызовы функций. Составные инструкции должны размещаться в отдельной строке.

Согласно другому правилу, применяемому к инструкциям, допускается записывать одну инструкцию в нескольких строках. Для этого достаточно заключить часть инструкции в пару круглых, квадратных или фигурных скобок. Любой программный код, заключенный в одну из этих конструкций, может располагаться на нескольких строках. Инструкция не будет считаться законченной, пока интерпретатор Python не достигнет строки с закрывающей скобкой.

Например, литерал списка можно записать так:

```
slist = [000,
111,
222]
```

Поскольку программный код заключен в пару квадратных скобок, интерпретатор всякий раз переходит на следующую строку, пока не обнаружит закрывающую скобку.

Как и в любом языке высокого уровня, в языке Python существуют операторы, позволяющие реализовать сложные алгоритмы вычисления. К этим операторам относятся логические конструкции **if**, а также операторы для реализации циклических вычислений. Рассмотрим вкратце использование таких операторов и начнем с инструкции **if**.

В языке Python инструкция **if** выбирает, какое действие следует выполнить. Она является основным инструментом выбора в Python, который отражает большую часть логики программы. Подобно всем составным инструкциям языка Python, инструкция **if** может содержать другие инструкции, в том числе другие условные инструкции **if**. Фактически Python позволяет комбинировать инструкции в программные последовательности (для их последовательного выполнения) и в произвольно вложенные конструкции (которые выполняются только при соблюдении определенных условий).

Условная инструкция **if** в языке Python – это типичная условная инструкция, которая присутствует в большинстве процедурных языков программирования. Синтаксически сначала записывается часть **if** с условным выражением, затем могут следовать одна или более необязательных частей **elif** (**"else if"**) с условными выражениями и, наконец, необязательная часть **else**.

Условные выражения и часть **else** имеют ассоциированные с ними блоки вложенных инструкций, с отступом относительно основной инструкции. Во время выполнения условной инструкции **if** интерпретатор выполняет блок инструкций, ассоциированный с первым условным выражением, только если оно возвращает истину, в противном случае выполняется блок инструкций **else**.

Общая форма записи условной инструкции **if** выглядит следующим образом:

```
if <test1>:        # Инструкция if с условным выражением test1
    <statements1>  # Ассоциированный блок
elif <test2>:      # Необязательные части elif
    <statements2>
else:              # Необязательный блок else
    <statements3>
```

Рассмотрим несколько простых примеров использования инструкции **if**. Следующий пример является простейшим случаем применения оператора **if**:

```
>>> if 1:
...         print 'true'
...
true
```

Обратите внимание, что приглашение к вводу изменяется на ... для строк продолжения в базовом интерфейсе командной строки, используемом здесь (в IDLE 3 текстовый курсор просто перемеща-

ется на следующую строку уже с отступом, а нажатие на клавишу **Backspace** возвращает на строку вверх).

Ввод пустой строки (двойным нажатием клавиши **Enter**) завершает инструкцию и приводит к ее выполнению. Поскольку число 1 является логической истиной, поэтому данная проверка всегда будет успешной.

Для обработки ложного результата нужно усложнить инструкцию, добавив часть else:

```
>>> if not 1:
... print 'true'
... else:
... print 'false'
...
false
```

Для организации циклов в программах на языке Python используются две основные логические конструкции, **while** и **for**. Первая из них, инструкция **while**, обеспечивает способ организации универсальных циклов. Вторая, инструкция **for**, предназначена для обхода элементов в последовательностях и выполнения блока программного кода для каждого из них.

В языке Python существуют и другие способы организации циклов, но инструкции **while** и **for** являются основными синтаксическими элементами, предоставляющими возможность для программирования повторяющихся действий.

Инструкция **while** является самой универсальной конструкцией для организации повторных вычислений (итераций) в языке Python. Эта инструкция выполняет блок инструкций (обычно с отступами) до тех пор, пока условное выражение истинно. Подобная конструкция называется «циклом», поскольку управление циклически возвращается к началу инструкции, пока условное выражение не станет ложным.

Если в результате проверки условия будет получено ложное значение, то управление немедленно передается первой инструкции, следующей за вложенным блоком тела цикла **while**. Тело цикла продолжает повторно выполняться, пока условное выражение возвращает истину; в случае если условное выражение при входе в цикл сразу же возвращает ложное значение, то тело цикла никогда не будет выполнено.

Наиболее сложная форма – инструкция **while** – состоит из строки заголовка с условным выражением, тела цикла, содержащего одну

или более инструкций с отступами, и необязательной части else, которая выполняется, когда управление передается за пределы цикла без использования инструкции **break**.

Интерпретатор продолжает вычислять условное выражение в строке заголовка и выполнять вложенные инструкции в теле цикла, пока условное выражение не вернет ложное значение:

```
while <test>:       # Условное выражение test
      <statements1> # Тело цикла
else:               # Необязательная часть else
      <statements2> # Выполняется, если выход из цикла
                    #   производится не инструкцией break
```

Рассмотрим несколько простых примеров с циклом **while**. Первый, который содержит инструкцию **print**, вложенную в цикл **while**, просто выводит сообщение до бесконечности. Следует помнить, что логическая константа True соответствует числу 1, поэтому результатом этого условного выражения всегда будет истина, и интерпретатор бесконечно будет выполнять тело цикла, пока пользователь не прервет его выполнение.

```
>>> while True:
... print('Type Ctrl-C to stop me!')
```

Следующий фрагмент кода периодически удаляет из строки первый символ, пока она не опустеет и условие не станет ложным.

```
>>> x = 'spam'
>>> while x:        # Пока x содержит элементы
... print(x, end=' ')
... x = x[1:]      # Вырезать первый символ из x
...
spam pam am m
```

Здесь аргумент **end=' '** обеспечивает вывод значений в одну строку через пробел.

Следующий фрагмент обрабатывает значения от **a** до **b**, исключая само значение **b**:

```
>>> a=0; b=10
>>> while a < b:
... print(a, end=' ')
... a += 1 # Или, a = a + 1

...
0 1 2 3 4 5 6 7 8 9
```

В языке Python отсутствует цикл **do... until**, который присутствует в других языках программирования. Однако его можно симулировать, добавив в конец тела цикла условную инструкцию и инструкцию **break**:

```
while True:
...тело цикла...
if exitTest():
       break
```

При организации циклов очень полезными могут оказаться еще две инструкции языка Python, **break** и **continue**. В языке Python инструкция **break** выполняет переход за пределы цикла, а инструкция continue выполняет безусловный переход в начало цикла. Инструкция **pass** в языке Python является пустой и используется для временных задержек.

Общий формат цикла **while** с учетом инструкций **break** и **continue** будет выглядеть так:

```
while <test1>:
   <statements1>
   if <test2>:
        break    # Выйти из цикла, пропустив часть else
   if <test3>:
        continue # Перейти в начало цикла, к выражению test1
   else:
        <statements2> # Выполняется, если не была использована
                      # инструкция 'break'
```

Инструкции **break** и **continue** могут находиться в любом месте внутри тела цикла **while** (или **for**), но, как правило, они используются в условных инструкциях **if** для выполнения необходимого действия в ответ на некоторое условие.

Инструкция **continue** вызывает немедленный переход в начало цикла. С ее помощью иногда можно избежать использования вложенных инструкций. В следующем примере инструкция **continue** используется для пропуска нечетных чисел.

В этом фрагменте кода выводятся четные числа < 10 и больше или равные 0. Здесь нужно вспомнить, что число 0 означает ложь, а оператор **%** вычисляет остаток от деления, поэтому данный цикл выводит числа в обратном порядке, пропуская значения, не кратные 2, то есть 8, 6, 4, 2, 0:

```
x = 10
while x:
```

```
    x = x-1                        # Или, x -= 1
    if x % 2 != 0: continue        # если нечетное,
                                   # то пропустить вывод
    print(x, end=' ')
```

Так как инструкция continue выполняет переход в начало цикла, то нет необходимости вкладывать инструкцию **print** в инструкцию **if** – она будет задействована только в том случае, если инструкция continue не будет выполнена. По своей сути эта инструкция частично выполняет функции оператора «**goto**», присутствующего в других языках программирования, но отсутствующего в языке Python.

Инструкция **break** приводит к немедленному выходу из цикла. Поскольку программный код, следующий в цикле за этой инструкцией, не будет выполнен, то с помощью **break** можно избежать вложения циклов.

Далее приводится пример простого интерактивного цикла, в котором производится ввод данных с помощью функции **input**. Выход из цикла происходит в случае, если в ответ на запрос имени будет введена строка «**stop**»:

```
>>> while 1:
... name = input('Enter name:')
... if name == 'stop': break
... age = input('Enter age: ')
... print('Hello', name, '=>', int(age) ** 2)
...
```

Еще одной полезной логической конструкцией является **for**. Цикл **for** является универсальным итератором последовательностей в языке Python; с его помощью можно выполнять обработку элементов в любых упорядоченных группах объектов. Инструкция **for** используется для обработки строк, списков и других встроенных объектов, поддерживающих возможность выполнения итераций.

Циклы **for** в языке Python начинаются со строки заголовка, где указывается переменная для присваивания, а также объект, обход которого будет выполнен. Вслед за заголовком следует блок (обычно с отступами) инструкций, которые требуется выполнить:

```
for <target> in <object>:  # Связывает элементы объекта с
                           # переменной цикла
       <statements>  # Повторяющееся тело цикла: использует
                     #переменную цикла
else:
       <statements>  # Если не попали на инструкцию 'break'
```

Когда интерпретатор выполняет цикл **for**, он поочередно, один за другим, присваивает элементы объекта последовательности переменной цикла и выполняет тело цикла для каждого из них. Для обращения к текущему элементу последовательности в теле цикла обычно используется переменная цикла.

Имя, используемое в качестве переменной цикла (возможно, новой), которое указывается в заголовке цикла **for**, обычно находится в области видимости самой инструкции **for**.

О ней почти нечего сказать; хотя она может быть изменена в теле цикла, тем не менее ей автоматически будет присвоен следующий элемент последовательности, когда управление вернется в начало цикла. После выхода из цикла эта переменная обычно ссылается на последний элемент последовательности, если только цикл не был завершен инструкцией **break**.

Инструкция **for** также поддерживает необязательную часть **else**, которая работает точно так же, как и в циклах **while**, – она выполняется, если выход из цикла производится не инструкцией **break** (то есть если в цикле были обработаны все элементы последовательности). Инструкции **break** и **continue**, представленные выше, в циклах **for** работают точно так же, как и в циклах **while**.

Полная форма цикла **for** имеет следующий вид:

```
for <target> in <object>:  # Присваивает элементы объекта с
                           # переменной цикла
      <statements>
      if <test>: break     # Выход из цикла, минуя блок else
      if <test>: continue  # Переход в начало цикла
      else:
         <statements> # Если не была вызвана инструкция 'break'
```

Рассмотрим несколько типичных примеров построения интерактивных циклов **for**.

Как уже обсуждалось, цикл **for** может выполнять обработку элементов в любых объектах последовательностей.

В первом примере переменной **x** поочередно присваивается слева направо каждый из трех элементов списка и выводится каждый из них с помощью инструкции **print**. Внутри инструкции **print** (в теле цикла) имя **x** ссылается на текущий элемент списка:

```
>>> for x in ["spam", "eggs", "ham"]:
... print(x, end=' ')
...
spam eggs ham
```

В следующих двух примерах вычисляются сумма и произведение всех элементов в списке.

```
>>> sum = 0
>>> for x in [1, 2, 3, 4]:
... sum = sum + x
...
>>> sum
10
>>> prod = 1
>>> for item in [1, 2, 3, 4]: prod *= item
...
>>> prod
24
```

Цикл **for**, будучи универсальным инструментом, может применяться к любым последовательностям, как в этом примере:

```
>>> S = "lumberjack"
>>> T = ("and", "I'm", "okay")
>>> for x in S: print(x, end=' ') # Обход строки

l u m b e r j a c k
>>> for x in T: print(x, end=' ') # Обход элементов кортежа
...
and I'm okay
```

Фактически, как будет показано чуть ниже, циклы **for** могут применяться даже к объектам, которые вообще не являются последовательностями, например к файлам.

8 Порт GPIO в измерительных системах

Миниатюрный компьютер Raspberry Pi может служить базисом для разработки систем управления и контроля, управляемых с помощью сигналов порта ввода-вывода (GPIO). Внешние электронные схемы могут подключаться к выводам GPIO через разъем P1 на плате Raspberry Pi.

Любой из выводов порта GPIO является цифровым и может работать как двунаправленный (ввод или вывод), поэтому к нему можно подключать либо нагрузку (вывод), либо считывать сигнал с цифрового датчика. Направление передачи сигнала можно указывать программным способом из приложения на языке C или Python.

Поскольку Raspbian OS – это полнофункциональная Linux-система, то измерительная или управляющая система, контролируемая GPIO, не сможет работать в реальном времени. Для работы с временными интервалами порядка единиц и десятков микросекунд необходимо разработать или иметь готовый драйвер устройства. Тем не менее многие системы не требуют столь коротких интервалов времени для обработки сигналов, поэтому для относительно медленных измерительных систем можно разработать программное обеспечение на Python или C, используя готовые библиотеки функций.

Перед тем как перейти к практике, вкратце ознакомимся с основами функционирования порта ввода-вывода GPIO. Выводы порта располагаются на 26-пиновом двухрядном разъеме на плате, который на схемах обозначен как P1. К выводам порта имеется возможность подключить 8 цифровых входов-выходов, а также интерфейсы I^2C, SPI и UART. Здесь же присутствуют линии питания +3,3 В, +5 В и земля (GND). Каждый бит GPIO порта может быть сконфигурирован либо как вход, либо как выход.

Важное замечание

Все выводы порта GPIO работают с напряжением 3,3 В, поэтому уровни сигналов, приходящие с внешних цепей, не должны превышать этого значения. В противном случае возможно повреждение платы.

При питании внешних электронных цепей от источника питания +3,3 В на плате Raspberry Pi потребляемый ток не может превышать 50 мА. Если схема потребляет больший ток, то необходимо использовать отдельный источник питания. При подключении сигнальных линий к плате длина проводников, соединяющих разъем P1 и схему, не должна превышать 10–12 см, поскольку при большей протяжен-

ности цифровой сигнал искажается, что может привести к сбоям в работе системы.

В этой главе мы разработаем несколько простых измерительных систем, которые будут управляться из приложений на языке Python.

Важное замечание

Для программирования приложений этой главы был использован Python 2.7.3, поэтому при переносе кода на версии 3.x необходимо учитывать отличия между этими версиями и соответственно модифицировать код в случае необходимости.

Рассмотрим вначале, как программировать GPIO-порт на языке Python.

Первое, что необходимо сделать, – установить в системе библиотеку **RPi.GPIO**. В Raspbian OS это можно сделать, набрав следующую команду в LXTerminal (можно использовать любое другое консольное приложение):

```
$ wget http://pypi.python.org/packages/source/R/RPi.GPIO/RPi.
GPIO-0.1.0.tar.gz
$ tar zxf RPi.GPIO-0.1.0.tar.gz
$ cd RPi.GPIO-0.1.0
$ sudo python setup.py install
```

После успешной инсталляции библиотеки можно использовать включенные в нее функции для управления линиями порта GPIO. Следующая последовательность демонстрирует, как можно получить доступ к выводу (биту) GPIO 18.

Вызовем интерпретатор Python:

```
$ sudo python
```

и подключим библиотеку **RPi.GPIO** к приложению, используя директиву **import**:

```
>>> import RPi.GPIO as GPIO
```

На следующем шаге нужно привязать вывод GPIO к плате Raspberry Pi с помощью следующей команды:

```
>>> GPIO.setmode(GPIO.BCM)
```

Далее устанавливаем направление выполнения операции (ввод или вывод):

```
GPIO.setup(18, GPIO.OUT)
```

После выполнения этой команды вывод 18 GPIO будет работать на вывод данных. Далее можно записать данные в этот бит. Поскольку порт цифровой, то допустимо одно из двух значений – 0 или 1. При этом для записи 0 (логический уровень «0») используется ключевое слово **`False`**, а для записи 1 (логический уровень «1») – ключевое слово **`True`**.

Следующая команда выполняет запись 0 в вывод GPIO 18:

```
GPIO.output(18, False)
```

Для чтения данных с вывода GPIO вначале нужно сконфигурировать этот вывод для приема (чтения) данных. Для вывода GPIO 18 это можно сделать, например, следующей командой:

```
GPIO.setup(18, GPIO.IN)
```

Теперь можно выполнять чтение данных с вывода GPIO 18. Следующая команда выполняет чтение данных и сохраняет полученные данные в переменной **`inpVal`**:

```
inpVal = GPIO.input(18)
```

При работе на большую нагрузку на выводе GPIO следует устанавливать буферные элементы (транзистор, операционный усилитель или цифровой буфер), поскольку максимально допустимый выходной ток на выводе GPIO ограничен единицами миллиампер. Пример подключения нагрузки к GPIO 18 показан на рис. 8.1.

Здесь для управления светодиодом используется вывод GPIO 18, сигнал с которого поступает на буферный элемент микросхемы 74HC14. 74HC14 включает в себя триггеры Шмитта, которые инвертируют входной сигнал, поэтому сигнал на выходе буфера (вывод 2 микросхемы 74HC14) будет инвертирован по отношению к выходу GPIO 18.

Буферные элементы могут запитываться как от источника +3,3 В на плате RPi, так и использовать автономный источник напряжения. Если необходимо управлять высоковольтными нагрузками, то можно применить изолирующий буфер, например оптопару 4N357-4N37, подключив входной диод оптопары к выводу GPIO через резистор.

Простейшее приложение на языке Python, исходный текст которого показан в листинге 8.1, будет переключать светодиод каждую секунду 10 раз.

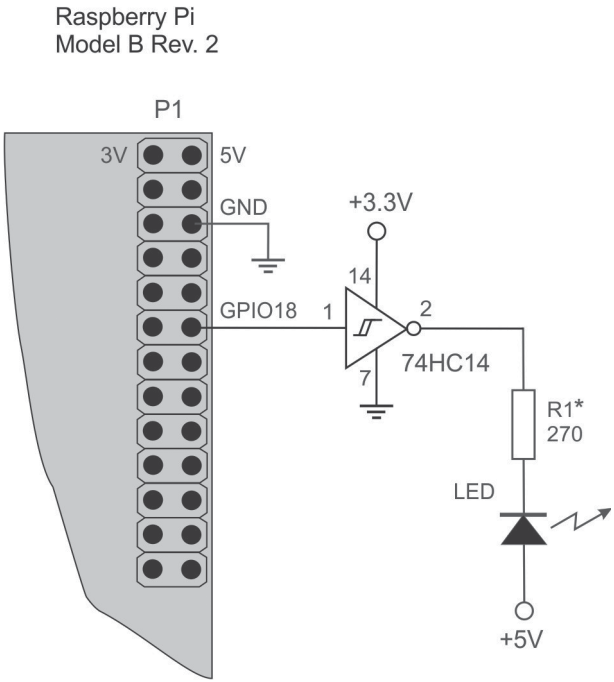

Рис. 8.1

Листинг 8.1

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
counter = 0
while (counter < 10):
        GPIO.output(18, False)
        sleep(1)
        GPIO.output(18, True)
        sleep(1)
        counter += 1
GPIO.output(18, False)
```

Предположим, что исходный текст листинга сохранен в файле gpio18_test.py. Тогда для выполнения программы необходимо набрать команду:

```
sudo python gpio18_test.py
```

Команда **sudo** нужна, поскольку непривилегированный пользователь (и его процесс) не может получить непосредственный доступ к аппаратным средствам Linux. В общей форме подобные программы должны предваряться командой **sudo**.

В следующем примере показано, как прочитать цифровой сигнал на выводе GPIO 23 и вывести управляющий сигнал через вывод GPIO 18. Аппаратная часть нашего проекта показана на рис. 8.2.

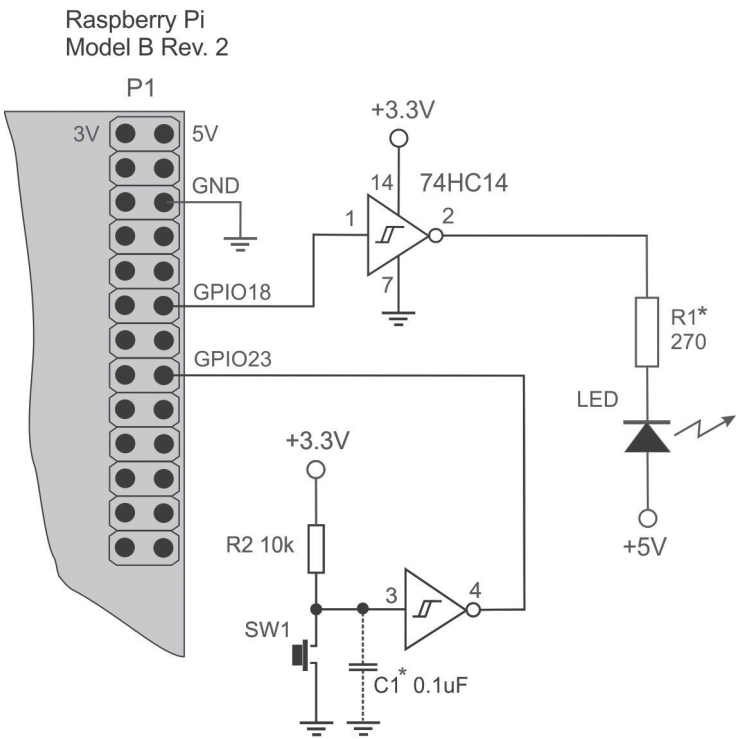

Рис. 8.2

Программа считывает состояние переключателя SW1 на выводе 3 буфера 74HC14 каждые 2 секунды. Если SW1 нажат, то светодиод LED на выводе 2 буфера загорается. Наоборот, когда SW1 отжат, то LED гаснет.

Поскольку в этом проекте используется механический переключатель, то для устранения так называемого «дребезга» контактов желательно установить конденсатор небольшой емкости (C1, показан пунктиром на схеме). Вместе с резистором R2 конденсатор C1 образует простейший фильтр нижних частот с постоянной времени, приблизительно равной 0,1 с.

Исходный текст программы на языке Python показан в листинге 8.2.

Листинг 8.2

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.IN)

while (True):
    bVal = GPIO.input(23)
    if (bVal != 0x0):
       print('SW1 is pressed. LED is turned ON')
       GPIO.output(18, True)
    else:
       print('SW1 is released. LED is turned OFF')
       GPIO.output(18, False)
    sleep(2)
```

Практические примеры простых систем управления

В этом разделе приводятся несколько демонстрационных систем управления с датчиками аналоговых сигналов на входе. В первом проекте система анализирует сигналы, поступающие с фоторезистора, и в зависимости от установленного порога светочувствительности включает/выключает светодиод. В качестве датчика светового потока используется фоторезистор с номинальным сопротивлением 20 К. Аппаратная часть системы представлена на рис. 8.3.

На этой схеме фоторезистор LDR и резистор R3 соединяются последовательно, образуя делитель напряжения. Напряжение с этого

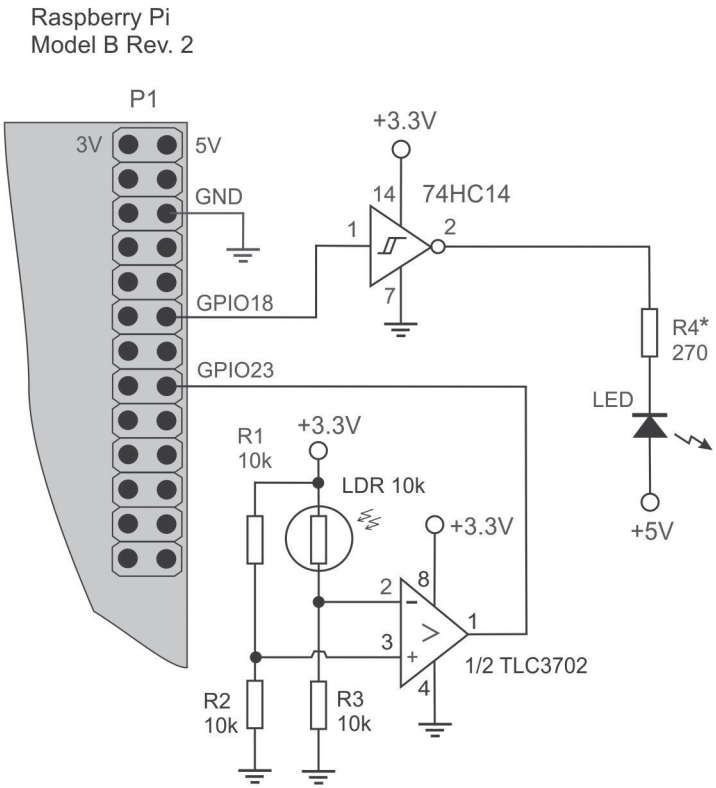

Рис. 8.3

делителя подается на инвертирующий вход компаратора напряжения на микросхеме TLC3702. На неинвертирующий вход этого компаратора подается образцовое напряжение с делителя напряжения R1–R2, которое определяет момент переключения выходного напряжения компаратора. В данном случае образцовое напряжение устанавливается равным половине напряжения питания, то есть 1,65 В.

Световой поток, попадающий на фоторезистор, вызывает уменьшение его сопротивления. Как только сопротивление становится меньше 10 К, напряжение на инвертирующем входе компаратора становится больше 1,65 В и на выходе устройства устанавливается уровень лог. «0». Если интенсивность светового потока уменьшается, то сопротивление фоторезистора увеличивается, что приводит к переключению выходного напряжения компаратора в лог. «1».

Такую систему можно использовать, например, для включения/выключения освещения в зависимости от времени суток. В данном случае в качестве исполнительного устройства используется обычный светодиод, сигнализирующий об изменении освещенности.

Программное обеспечение периодически считывает состояние входа на выводе GPIO 23. Если на GPIO 23 присутствует высокий уровень напряжения (лог. «1»), то светодиод включается. Выключение индикации происходит при низком уровне напряжения на входе GPIO 23.

Исходный текст программы на Python показан в листинге 8.3.

Листинг 8.3

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.IN)

while (True):
    bVal = GPIO.input(23)
    if (bVal == 0x0):
       print('Light intensity increased. LED is ON')
       GPIO.output(18, False)
    else:
       print('Shadows fell.LED is OFF')
       GPIO.output(18, True)
    sleep(2)
```

Вместо датчика светового потока в таком проекте можно применить любой другой аналоговый датчик, выходное напряжение которого изменяется в зависимости от какой-либо физической величины (температуры, давления, влажности и т. д.).

На рис. 8.4 показана аппаратная часть системы управления с датчиком температуры LM35. Точка срабатывания такой системы устанавливается пользователем с помощью резистивного делителя R1-R2.

В данном проекте точка срабатывания выбрана равной 0,25 В, что соответствует температуре окружающей среды 25 °С. Как только температура превысит указанный порог, напряжение на выходе датчика температуры оказывается выше 0,25 В и выход компаратора

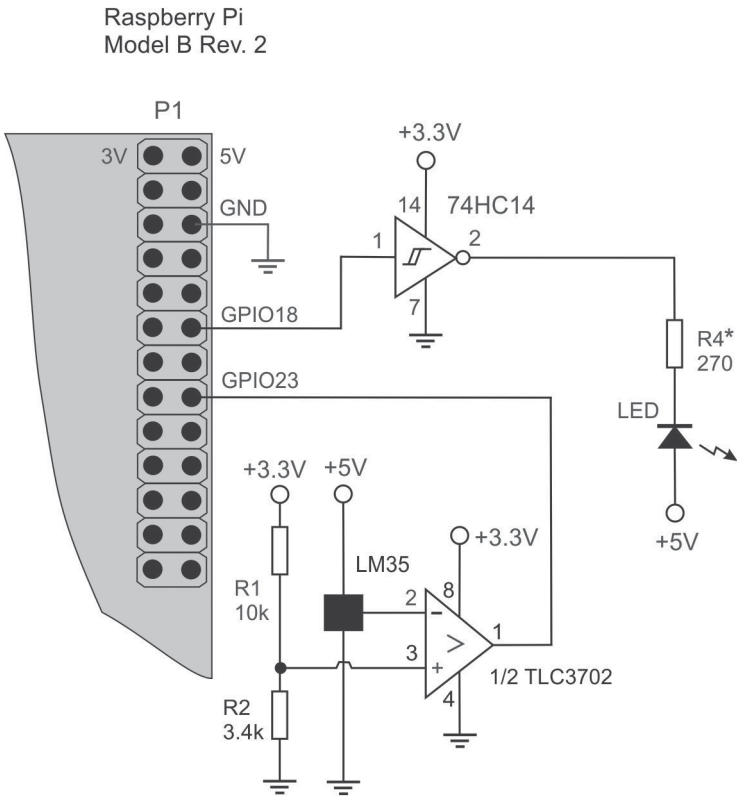

Рис. 8.4

переходит в состояние лог. «0». Если температура опускается ниже 25 °С, напряжение на выходе датчика падает ниже 0,25 В, что, в свою очередь, приводит к переключению выхода компаратора в лог. «1». Как и в предыдущем примере, сигнал компаратора может быть использован для управления каким-либо исполнительным устройством (электродвигателем, реле и т. д.).

В простейшем случае подключение мощного исполнительного устройства можно осуществить, применив мощный биполярный транзистор, работающий в ключевом режиме, и электромагнитное реле, как показано на рис. 8.5.

Как видно по рисунку, транзисторная сборка Дарлингтона (TIP122) управляется с вывода GPIO 18, а нагрузка LOAD включена в коллекторную цепь транзистора. Диод D1 необходим для защиты тран-

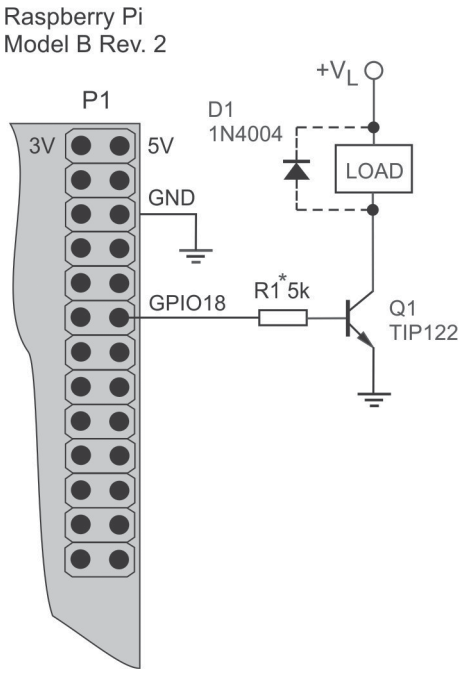

Рис. 8.5

зистора при работе с индуктивной нагрузкой. Можно полностью изолировать выход GPIO от нагрузки, применив оптоэлектронный ключ на одной из популярных микросхем, например 4N35–4N37.

Расширение порта GPIO с помощью интерфейса I²C

Плата Raspberry Pi имеет относительно небольшое количество сигнальных линий порта GPIO, что может оказаться недостаточным для создания комплексных систем управления и измерения. В этом случае можно расширить количество выводов управления, подключив к порту GPIO один из модулей расширения, работающих с интерфейсом SPI или I²C.

В дальнейшем мы будем использовать интерфейс I²C, поскольку у него есть целый ряд преимуществ, по сравнению с другими интер-

фейсами, в частности он позволяет подключать несколько устройств к одной и той же шине.

Упрощенная схема подключения устройств к шине I²C показана на рис. 8.6.

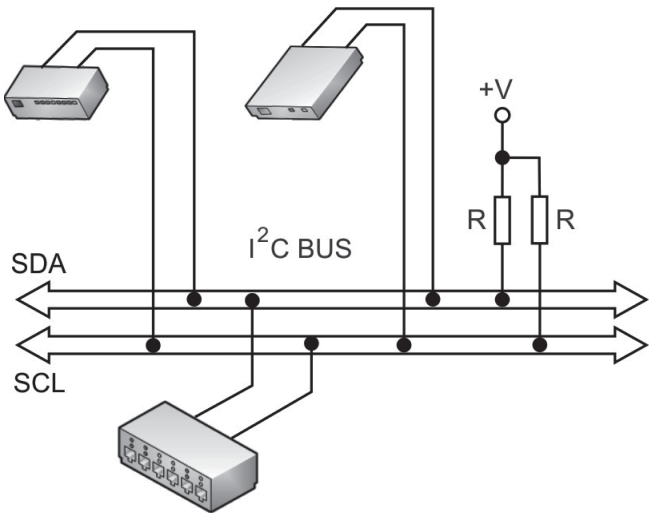

Рис. 8.6

Коммуникации по шине I²C осуществляются по двум сигнальным линиям, которые называются SCL и SDA. По линии SCL передаются сигналы синхронизации для передачи данных, в то время как линия SDA служит для передачи данных. Все устройства I²C подключаются к этим линиям при помощи «подтягивающих» резисторов, номинал которых может изменяться приблизительно от 1,5 до 20 К, в зависимости от того, какая скорость передачи данных выбрана. Для больших номиналов резисторов скорость передачи данных по шине будет меньше, хотя и потребляемый ток в этом случае также будет меньше.

На шине I²C достаточно иметь только два подтягивающих резистора независимо от количества подключенных устройств. Любое устройство на шине I²C может выполнять функцию либо ведущего («master»), либо ведомого («slave»). Ведущий управляет шиной, передавая сигналы синхронизации по линии SCL всем ведомым устройствам. Устройство, выполняющее функцию ведущего, иници-

рует передачу данных по шине; в системе может быть только один ведущий, все остальные устройства должны быть ведомыми. В системах с Raspberry Pi в роли ведущего выступает сам модуль RPi.

Передача по интерфейсу I²C может выполняться как ведущим, так и ведомым, хотя сигналы управления подаются на шину только ведущим устройством. На рис. 8.7 показана упрощенная временная диаграмма интерфейса I²C.

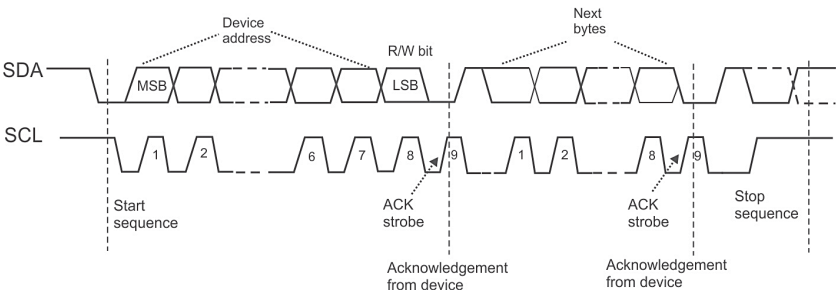

Рис. 8.7

При работе с интерфейсом I²C должны соблюдаться два следующих основных правила. Первое: инициализация передачи данных может осуществляться, только когда шина свободна, то есть никакой текущей передачи данных нет. Второе: при передаче данных данные на линии SDA не должны изменяться при высоком уровне сигнала на линии SCL. Любые изменения на линии данных при высоком уровне сигнала на SCL интерпретируются как сигналы управления и обрабатываются соответствующим образом. Таким образом, каждый бит данных может защелкиваться только при переходе 0–1 на линии SCL.

Передача данных по интерфейсу синхронизируется несколькими управляющими последовательностями сигналов. Одна из них называется «стартовая последовательность», а другая – «стоповая последовательность». Любая передача данных начинается со стартовой последовательности и заканчивается стоповой последовательностью. Обе последовательности инициируются ведущим устройством.

При стартовой последовательности ведущий устанавливает низкий уровень на линии SDA, сохраняя высокий уровень на SCL (см. рис. 8.7). Стоповая последовательность инициируется, когда ведущий изменяет уровень линии SDA с низкого на высокий, со-

храняя высокий уровень на SCL. При передаче данных данные на линии SDA должны быть стабильны при высоком уровне на линии SCL.

Различные устройства с интерфейсом I²C могут осуществлять различные транзакции на шине, поскольку количество передаваемых байтов данных между стартовой и стоповой последовательностями не ограничено. В любом случае каждая транзакция начинается с передачи байта адреса, который должен содержать адрес одного из ведомых устройств и код операции (запись или чтение).

Адреса устройств на шине I²C могут содержать от 7 до 10 бит. Большинство устройств имеют 7-битовый адрес, что позволяет адресовать до 128 устройств на шине I²C. Кроме семи адресных битов, ведущий устанавливает также и 8-й бит, который указывает на код операции. Если этот бит равен 0, то ведущий будет выполнять запись данных в ведомое устройство. Если бит равен 1, то будет выполняться чтение данных с ведомого устройства. Первые 7 бит адреса являются старшими битами, а бит операции (R/W) является младшим битом в этой последовательности.

Кроме того, по завершении каждой транзакции ведущий выставляет бит подтверждения выполнения («Acknowledge Bit») или неподтверждения выполнения («Non-Acknowledge Bit»).

Для использования интерфейса I²C в системах с Raspberry Pi необходимо выполнить некоторые подготовительные шаги. Первое – отредактируем файл конфигурации /etc/modprobe.d/raspi-blacklist.conf с помощью текстового редактора:

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

Затем найдем в этом файле следующие строки:

```
# blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
blacklist i2c-bcm2708
```

Теперь нужно закомментировать строку, где упоминается I²C. Эти строки должны выглядеть так:

```
# blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Второе – следует включить модуль I²C в список загружаемых модулей ядра, для чего нужно отредактировать файл /etc/modules с помощью текстового редактора (nano в данном случае):

```
sudo nano /etc/modules
```

Мы должны увидеть примерно следующее:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should
# be loaded
# at the boot time, one per line. Lines beginning with «#»
# are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
```

Здесь мы должны добавить **i2c-dev** в конец файла. Окончательный вариант файла должен выглядеть так:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should
# be loaded
# at boot time, one per line. Lines beginning with «#» are
# ignored.
# Parameters can be specified after the module name.

snd-bcm2835
i2c-dev
```

На следующем шаге нужно проинсталлировать несколько пакетов, требуемых для работы интерфейса I²C. Первый из них, **i2c-tools**, может быть установлен следующей командой:

```
$ sudo apt-get install i2c-tools
```

Поскольку мы собираемся писать приложения на языке Python, нам необходимо установить пакет **python-smbus** с помощью команды:

```
$ sudo apt-get install python-smbus
```

Теперь следует включить пользователя pi в группу I²C для доступа к устройствам:

```
$ sudo adduser pi i2c
```

На последнем шаге нужно перезагрузить систему командой

```
$ sudo reboot
```

Теперь мы сможем использовать интерфейс I²C для подключения электронных схем к Raspberry Pi. В нашем первом проекте рассмотрим подключение расширителя интерфейса PCF8574 к RPi.

Применение расширителя ввода-вывода PCF8574

Увеличить количество линий ввода-вывода в системах с Raspberry Pi можно путем использования специальных микросхем. Одной из таких микросхем является PCF8574, управление которой осуществляется по интерфейсу I²C. Микросхема PCF8574 имеет 8-битовый квазидвунаправленный порт (обозначается P0–P7), каждый из битов которого включает триггер-защелку по выходу. Кроме того, выходы порта могут непосредственно управлять относительно большими нагрузками, например маломощными реле. Любой из выводов порта может использоваться либо как вход, либо как выход, причем сигналы переключения направления передачи не требуются.

Функциональная схема устройства PCF8574 показана на рис. 8.8.

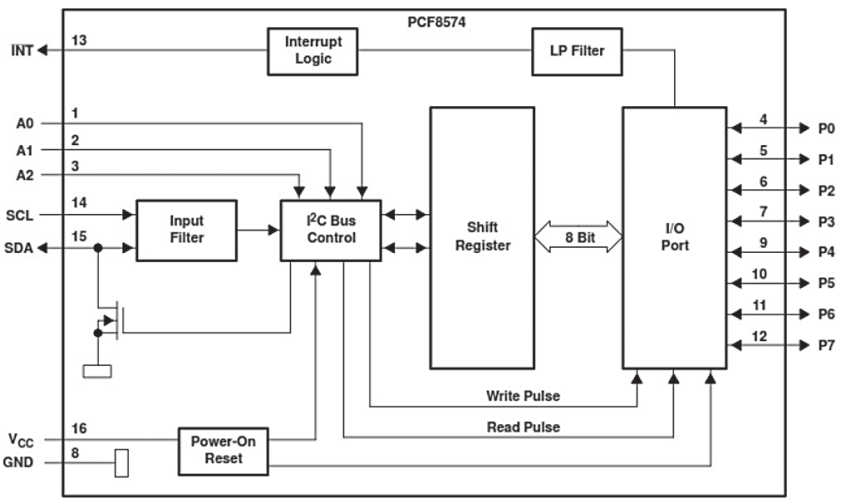

Рис. 8.8

При подаче питания на микросхему все линии порта переключаются на ввод данных. Формат данных для порта ввода-вывода показан на рис. 8.9.

Interface Definition

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I²C slave address	L	H	L	L	A2	A1	A0	R/$\overline{W}$
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0

Рис. 8.9

Следующий проект демонстрирует применение расширителя PCF8574 с модулем RPi. Аппаратная часть проекта показана на рис. 8.10.

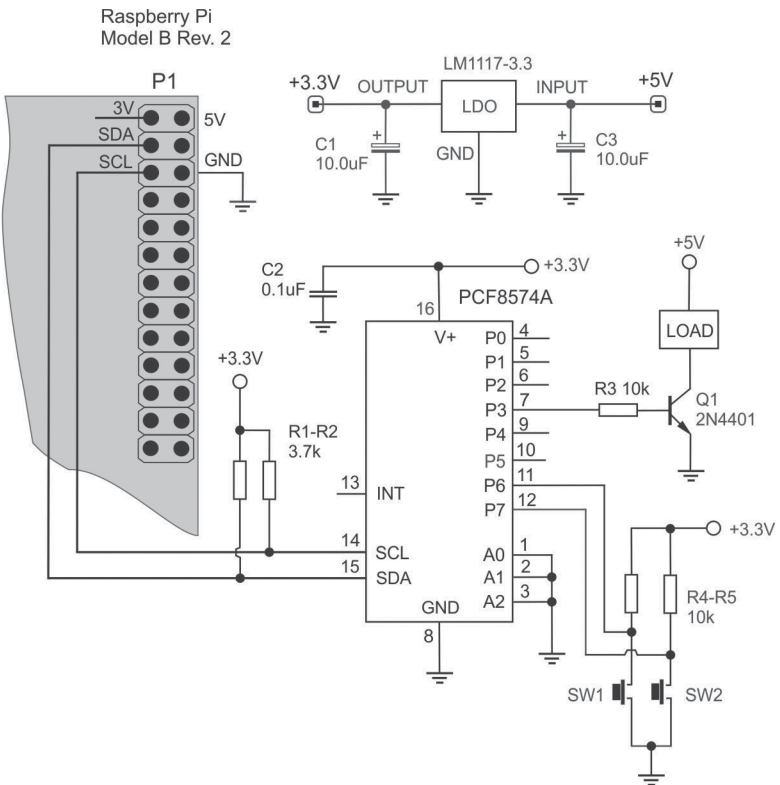

Рис. 8.10

В этой схеме линии SCL (вывод 14) и SDA (вывод 15) микросхемы PCF8574A подключаются к соответствующим выводам на разъеме P1 платы Raspberry Pi.

Общий провод платы Raspberry Pi (вывод «GND») соединяется с общим проводом схемы. Расширитель ввода-вывода может получать питание либо с вывода «3V» на плате RPi, либо от отдельного источника с напряжением 3,3 В. В данной схеме напряжение 3,3 В обеспечивается линейным регулятором LM1117-3.3, на который подается входное напряжение +5 В.

Программная часть проекта написана на языке Python. В программе используется целый ряд библиотечных функций, описание которых приводится в табл. 8.1.

Таблица 8.1. Функции SMBus

Функция	Описание	Параметры	Возвращаемое значение
SMBus Access			
write_quick(addr)	Quick transaction	int addr	long
read_byte(addr)	Read Byte transaction	int addr	long
write_byte(addr,val)	Write Byte transaction	int addr,char val	long
read_byte_data(addr,cmd)	Read Byte Data transaction.	int addr,char cmd	long
write_byte_data(addr,cmd,val)	Write Byte Data transaction	int addr,char cmd,char val	long
read_word_data(addr,cmd)	Read Word Data transaction	int addr,char cmd	long
write_word_data(addr,cmd,val)	Write Word Data transaction	int addr,char cmd,int val	long
process_call(addr,cmd,val)	Process Call transaction	int addr,char cmd,int val	long
read_block_data(addr,cmd)	Read Block Data transaction	int addr,char cmd	long[]
write_block_data(addr,cmd,vals)	Write Block Data transaction	int addr, char cmd, long[]	None
block_process_call(addr,cmd,vals)	Block Process Call transaction	int addr,char cmd, long[]	long[]
I^2C Access			
read_i2c_block_data(addr,cmd)	Block Read transaction	int addr, char cmd	long[]
write_i2c_block_ data(addr,cmd,vals)	Block Write transaction	int addr,char cmd, long[]	None

Исходный текст программы для данного проекта приведен в листинге 8.4.

Листинг 8.4

```
import smbus
import time

bus = smbus.SMBus(1)

bWrite = 0xFF
mask = 0xF7
bus.write_byte(0x38, 0xF7)
while (True):
    res = bus.read_byte(0x38)
    res &= 0xC0
    print('P6-P7 bit state: 0x%X' %res)
    if ((res == 0x0) or (res == 0xC0)):
        time.sleep(2)
        continue
    if (res == 0x80):
        bWrite = 0x8 | mask
        bus.write_byte(0x38, bWrite)
    if (res == 0x40):
        bWrite &= mask
        bus.write_byte(0x38, bWrite)
    time.sleep(2)
```

Здесь адрес устройства PCF8574 равен 0x38 – он легко определяется путем выполнения команды

```
$ i2cdetect -y 1
```

Оператор

```
bus.write_byte(0x38, 0xF7)
```

переводит биты P0–P2 и P4–P7 в лог. «1» – это необходимо перед установкой режима ввода.

Программный код периодически считывает состояние выводов P6–P7 и в зависимости от полученного двоичного кода устанавливает на выводе P3 либо лог. «0», либо лог. «1». Если оба переключателя, SW1 и SW2, оказываются замкнутыми или открытыми, состояние выхода P3 не меняется.

Следующая последовательность операторов выполняет указанные действия:

```
if ((res == 0x0) or (res == 0xC0)):
      time.sleep(2)
      continue
```

Когда на P6 лог. «0» а на P7 – лог. «1» (переключатель SW1 нажат), то на выводе P3 устанавливается высокий уровень, который открывает транзистор Q1, тем самым подключая питание к нагрузке. Это выполняется следующими операторами:

```
if (res == 0x80):
    bWrite = 0x8 | mask
    bus.write_byte(0x38, bWrite)
```

Если на P6 присутствует уровень лог. «1», а на P7 – уровень лог. «0» (SW2 нажат), то транзистор Q1 закрывается, отключая питание от нагрузки LOAD:

```
if (res == 0x40):
    bWrite &= mask
    bus.write_byte(0x38, bWrite)
```

Контроль состояния входов P6 и P7 осуществляется каждые 2 секунды функцией `sleep(2)`.

Использование расширителя ввода-вывода MCP23008

Другим популярным расширителем ввода-вывода является микросхема MCP23008. В следующем проекте будут показаны подключение к плате Raspberry Pi и программирование этого устройства. Несколько слов о самой микросхеме расширителя. MCP23008 обеспечивает расширение интерфейса I^2C в 8-битовый параллельный цифровой порт ввода-вывода. Микросхема MCP23008 включает несколько 8-битовых регистров конфигурации, с помощью которых задается направление операции, а также полярность выходных сигналов. Перечень регистров устройства приводится в табл. 8.2.

В данном проекте нужно сконфигурировать только регистр IODIR. Каждый бит этого регистра определяет направление передачи данных в соответствующем бите регистра данных GP. Если бит в IODIR установлен, то соответствующий ему вывод GP работает на ввод данных. Если бит в IODIR очищен, то соответствующий ему вывод GP работает на вывод данных.

Таблица 8.2. Регистры устройства

Адрес	Регистр
00h	IODIR
01h	IPOL
02h	GPINTEN
03h	DEFVAL
04h	INTCON
05h	IOCON
06h	GPPU
07h	INTF
08h	INTCAP (Read-only)
09h	GPIO
0Ah	OLAT

Схема аппаратной части проекта показана на рис. 8.11.

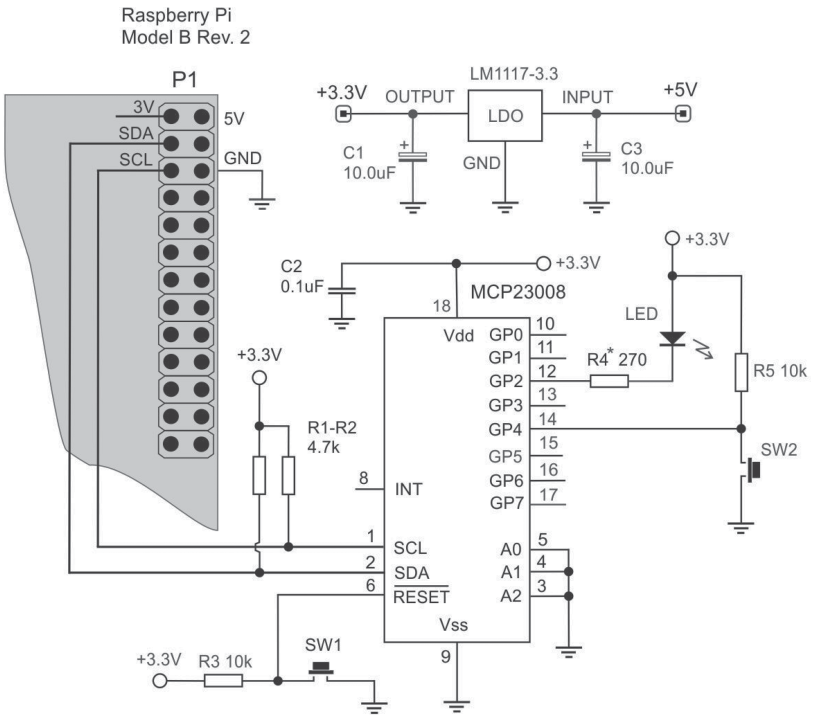

Рис. 8.11

Соединения между платой Raspberry Pi и MCP23008 показаны в табл. 8.3.

Таблица 8.3

Вывод разъема P1 RPi	Вывод MCP23008
SCL	1 (SCL)
SDA	2 (SDA)

Общий провод платы Raspberry Pi (вывод «`GND`») соединяется с общим проводом схемы. Расширитель ввода-вывода может получать питание либо с вывода «`3V`» на плате RPi, либо от отдельного источника с напряжением 3,3 В. В данной схеме напряжение 3,3 В обеспечивается линейным регулятором LM1117-3.3, на который подается входное напряжение +5 В.

Система работает следующим образом. Программа периодически опрашивает состояние вывода GP4, к которому подсоединен переключатель SW2. Если SW2 нажат, то выход GP2 переключается в противоположное состояние, что приводит к переключению светодиода LED. При разомкнутом переключателе SW2, GP2 не меняет состояния до тех пор, пока вход GP4 остается в состоянии лог. «1».

Адрес устройства MCP23008 на шине I^2C определяется с помощью утилиты `i2cdetect` и оказывается равным 0x20 (рис. 8.12).

Исходный текст программы, управляющей расширителем, показан в листинге 8.5.

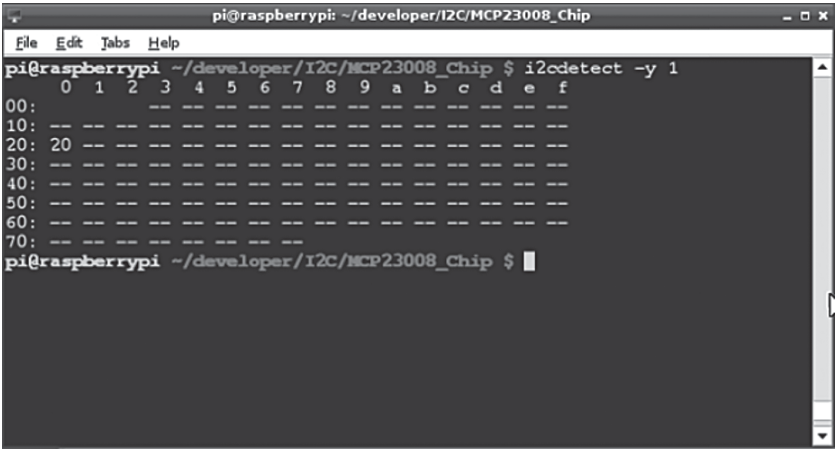

Рис. 8.12

Листинг 8.5

```
import smbus
import time

bus = smbus.SMBus(1)
bus.write_byte_data(0x20, 0x00, 0x10)
while (True):
    bRead = bus.read_byte_data(0x20, 0x9)
    bRead &= 0x10
    if (bRead == 0x0):
          print('GP4 is low.')
          bus.write_byte_data(0x20, 0x9, 4)
          time.sleep(1)
          bus.write_byte_data(0x20, 0x9, 0)
          time.sleep(1)
```

Здесь мы конфигурируем все порты GP (за исключением GP4) для ввода данных путем записи соответствующего значения в регистр IODIR по адресу 0x0:

```
bus.write_byte_data(0x20, 0x00, 0x10)
```

Затем программа переходит к выполнению цикла **while**, в котором производится периодическая проверка вывода GP4 с помощью следующей последовательности операторов:

```
bRead = bus.read_byte_data(0x20, 0x9)
bRead &= 0x10
if (bRead == 0x0):
       . . .
```

Если GP4 становится в 0 (SW2 нажат), то переменной **bRead** присваивается значение 0x0 и светодиод LED на выводе GP2 начинает мигать.

Система измерения температуры на базе интерфейса I²C

Представленная ниже система измерения температуры окружающей среды базируется на основе популярного датчика температуры DS1621, управление которым осуществляется по интерфейсу I²C с модуля Raspberry Pi. Полученные от датчика данные представлены в цифровом формате и пригодны для немедленной обработки. Краткое описание функционирования датчика приводится далее.

Датчик температуры DS1621 может выполнять функции цифрового термометра и термостата, выдавая результат в 9-битовом формате. Кроме того, устройство может устанавливать активный уровень сигнала на специальном выходе **Tout** при превышении заданного порога **TH**, который может задаваться пользователем. Выход **Tout** остается в активном состоянии до тех пор, пока температура не упадет ниже указанного порога.

Пользовательские установки сохраняются в энергонезависимой памяти устройства, поэтому датчик DS1621 может быть запрограммирован один раз и затем установлен в систему. Операции конфигурирования и чтения данных с устройства осуществляются посредством интерфейса I^2C.

Описание выводов микросхемы датчика приводится в табл. 8.4.

Таблица 8.4. Назначение выводов датчика DS1621

Вывод	Обозначение	Описание
1	SDA	Data input/output pin for 2-wire serial communication port
2	SCL	Clock input/output pin for 2-wire serial communication port
3	T_{OUT}	Thermostat output. Active when temperature exceeds TH; will reset when temperature falls below TL
4	GND	Ground pin
5	A2	Address input pin
6	A1	Address input pin
7	A0	Address input pin
8	V_{DD}	Supply voltage input power pin (2.7V to 5.5V)

Считанный с датчика двоичный код температуры представлен в форме дополнительного 9-битового кода, который выдается устройством при подаче команды **READ TEMPERATURE**. В табл. 8.5 показаны соотношения между измеренной температурой и двоичным кодом на выходе датчика.

Данные датчика передаются в микропроцессор, начиная со старшего байта (MSB), и содержат 2 байта. Если нужна точность в 1 °C, то достаточно обработать старший байт. Для точности в 0,5 °C необходимо обработать оба байта данных.

Формат данных, прочитанных с датчика температуры, представляется следующим образом (рис. 8.13).

Для получения более высокого разрешения нужно отбросить младший байт (LSB) полученных данных и затем выполнить чтение дополнительных данных с помощью команд **Read Counter** и

Таблица 8.5. Соответствие температуры двоичному коду для некоторых значений

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125 °C	01111101 00000000	7D00h
+25 °C	00011001 00000000	1900h
+1/2 °C	00000000 10000000	0080h
+0 °C	00000000 00000000	0000h
−1/2 °C	11111111 10000000	FF80h
−25 °C	11100111 00000000	E700h
−55 °C	11001001 00000000	C900h

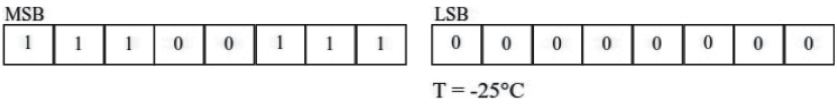

Рис. 8.13

Read Slope. Результат может быть подсчитан с помощью формулы, показанной на рис. 8.14.

$$TEMPERATURE=TEMP\_READ-0.25+\frac{(COUNT\_PER\_C-COUNT\_REMAIN)}{COUNT\_PER\_C}$$

Рис. 8.14

Для расшифровки аббревиатур следует обратиться к даташиту на устройство DS1621.

Аппаратная часть системы показана на рис. 8.15.

Исходный текст программы на языке Python представлен в листинге 8.6.

Листинг 8.6

```
import smbus
import time

addrDS1621 = 0x90 >> 1

bus = smbus.SMBus(1)
bus.write_byte_data(addrDS1621, 0xAC, 0x3)

while (True):
```

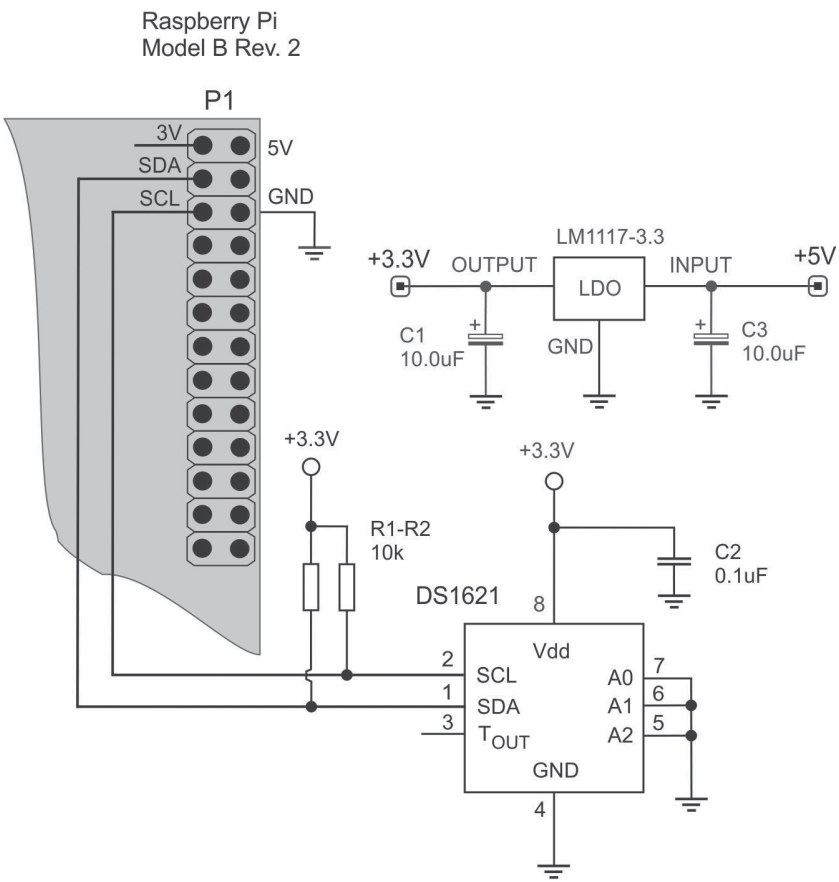

Рис. 8.15

```
bus.write_byte(addrDS1621, 0xEE)
time.sleep(3)
tmp = bus.read_i2c_block_data(addrDS1621, 0xAA)

bus.write_byte(addrDS1621, 0xA8)
counter = float(bus.read_byte(addrDS1621))
bus.write_byte(addrDS1621, 0xA9)
slope = float(bus.read_byte(addrDS1621))

temp = float(tmp[0]) -0.25 + (slope -counter) / slope
print('TEMPERATURE in the ROOM, deg.Celsius: %4.2f'%temp)
```

Результаты выполнения программы показаны на рис. 8.16.

К настоящему времени разработано большое количество проектов различных измерительных систем на основе Raspberry Pi, информацию по которым можно легко обнаружить в Интернете.

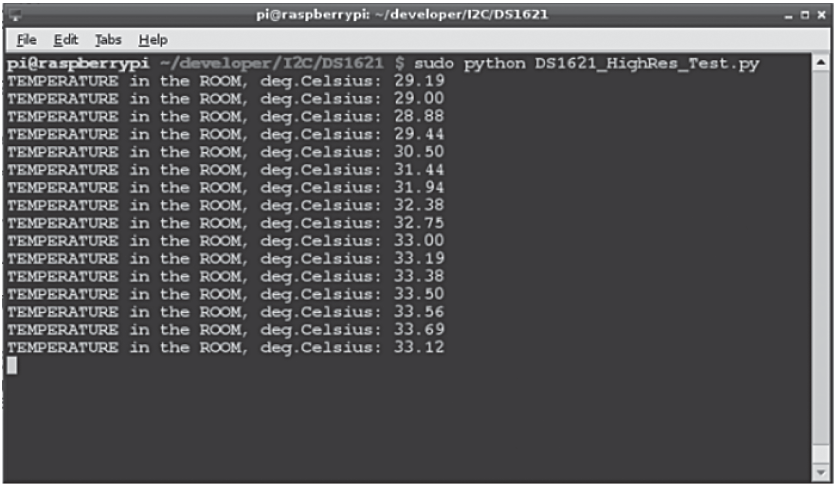

Рис. 8.16

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС БУКС» наложенным платежом, выслав открытку или письмо по почтовому адресу: 123242, Москва, а/я 20 или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. (499) 725-54-09, 725-50-27; электронный адрес **books@alians-kniga.ru**.

Магда Юрий Степанович

Raspberry Pi.
Руководство по настройке и применению

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Подписано в печать 20.11.2013. Формат 60×90 1/16.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 6,75. Тираж 200 экз.

Веб-сайт издательства: www.дмк.рф