4 Диски и файловые системы

В главе 3 мы рассмотрели дисковые устройства верхнего уровня, которые делают ядро доступным. В данной главе мы детально расскажем о работе с дисками в Linux. Вы узнаете о том, как создавать разделы дисков, настраивать и поддерживать файловые системы в этих разделах, а также работать с областью подкачки.

Вспомните о том, что у дисковых устройств есть имена вроде `/dev/sda`, первого диска подсистемы SCSI. Такой тип блочного устройства представляет диск целиком, однако внутри диска присутствуют различные компоненты и слои.

На рис. 4.1 приведена схема типичного диска в Linux (масштаб не соблюден). По мере изучения этой главы вы узнаете, где находится каждый его фрагмент.

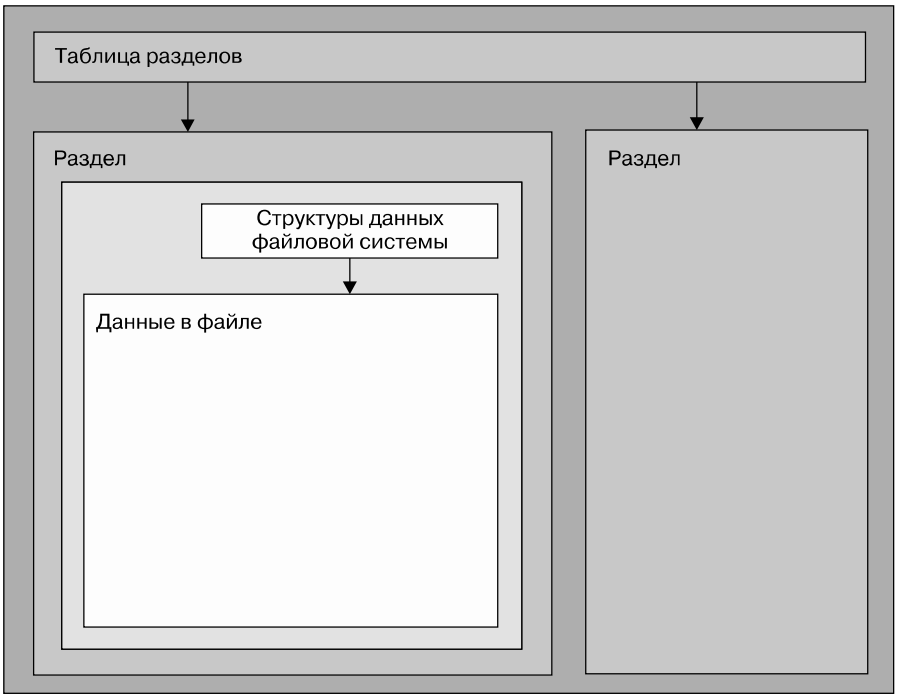

Рис. 4.1. Схема типичного диска Linux

Разделы являются более мелкими частями всего диска. В Linux они обозначаются с помощью цифры после названия блочного устройства и, следовательно, получают такие имена, как, например, `/dev/sda1` и `/dev/sdb3`. Ядро представляет каждый раздел в виде блочного устройства, как если бы это был целый диск. Разделы определяются в небольшой области диска, которая называется *таблицей разделов*.

ПРИМЕЧАНИЕ

Многочисленные разделы были когда-то распространены в системах с большими дисками, поскольку старые ПК могли загружаться только из определенных частей диска. К тому же администраторы использовали разделы, чтобы зарезервировать некоторое пространство для областей операционной системы. Например, они исключали возможность того, чтобы пользователи заполнили все свободное пространство системы и нарушили работу важных служб. Такая практика не является исключительной для Unix; вы по-прежнему сможете найти во многих новых системах Windows несколько разделов на одном диске. Кроме того, большинство систем располагает отдельным разделом подкачки.

Хотя ядро и позволяет вам иметь одновременный доступ ко всему диску и к одному из его разделов, вам не придется это делать, если только вы не копируете весь диск.

Следующий за разделом слой является *файловой системой*. Это база данных о файлах и каталогах, с которыми вы привыкли взаимодействовать в пространстве пользователя. Файловые системы будут рассмотрены в разделе 4.2.

Как можно заметить на рис. 4.1, если вам необходим доступ к данным в файле, вам потребуется выяснить из таблицы разделов расположение соответствующего раздела, а затем отыскать в базе данных файловой системы этого раздела желаемый файл с данными.

Чтобы обращаться к данным на диске, ядро Linux использует систему слоев, показанную на рис. 4.2. Подсистема SCSI и все остальное, описанное в разделе 3.6, представлены в виде одного контейнера. Обратите внимание на то, что с дисками можно работать как с помощью файловой системы, так и непосредственно через дисковые устройства. В этой главе вы попробуете оба способа.

Чтобы уяснить, как все устроено, начнем снизу, с разделов.

4.1. Разделы дисковых устройств

Существуют различные типы таблиц разделов. Традиционная таблица — та, которая расположена внутри *главной загрузочной записи MBR* (Master Boot Record). Новым, набирающим силу стандартом является *глобальная таблица разделов с уникальными идентификаторами* GPT (Globally Unique Identifier Partition Table).

Приведу перечень доступных в Linux инструментов для работы с разделами:

- `parted` — инструмент командной строки, который поддерживает как таблицу MBR, так и таблицу GPT;
- `gparted` — версия инструмента `parted` с графическим интерфейсом;
- `fdisk` — традиционный инструмент командной строки Linux для работы с разделами. Не поддерживает таблицу GPT;

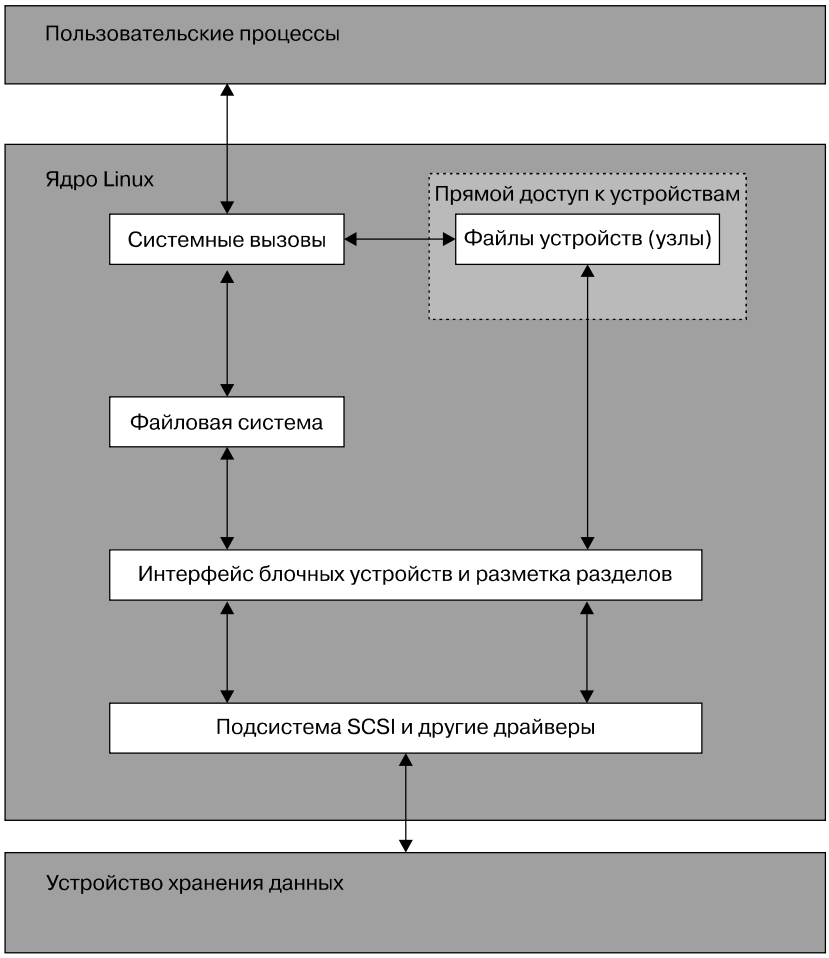

Рис. 4.2. Схема доступа ядра к диску

- `gdisk` — версия инструмента `fdisk`, которая поддерживает таблицу GPT, но не работает с MBR.

Поскольку инструмент `parted` поддерживает обе таблицы (MBR и GPT), в данной книге мы будем пользоваться им. Однако многие пользователи предпочитают интерфейс `fdisk`, и в этом нет ничего плохого.

ПРИМЕЧАНИЕ

Хотя команда parted способна создавать и изменять файловые системы, не следует использовать ее для манипуляций с файловой системой, поскольку вы можете легко запутаться. Имеется существенное отличие работы с разделами от работы с файловой системой. Таблица разделов устанавливает границы диска, в то время как файловая система гораздо сильнее вовлечена в структуру данных. Исходя из этого, мы будем применять команду parted для работы с разделами, а для создания файловых систем используем другие утилиты (см. подраздел 4.2.2). Даже документация к команде parted призывает вас создавать файловые системы отдельно.

4.1.1. Просмотр таблицы разделов

Можно просмотреть таблицу разделов вашей системы с помощью команды `parted -l`. Приведу пример результатов работы для двух дисковых устройств с различными типами таблиц разделов:

```
# parted -l

Model: ATA WDC WD3200AAJS-2 (scsi)
Disk /dev/sda: 320GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number   Start   End    Size   Type      File system    Flags
 1       1049kB  316GB  316GB  primary   ext4           boot
 2       316GB   320GB  4235MB extended
 5       316GB   320GB  4235MB logical   linux-swap(v1)

Model: FLASH Drive UT_USB20 (scsi)
Disk /dev/sdf: 4041MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size     File system  Name        Flags
 1      17.4kB  1000MB  1000MB                myfirst
 2      1000MB  4040MB  3040MB                mysecond
```

Первое устройство, `/dev/sda`, использует традиционную таблицу разделов MBR (которую команда `parted` назвала `msdos`), а второе устройство содержит таблицу GPT.

Обратите внимание на различающиеся параметры в этих таблицах разделов, поскольку сами таблицы различны. В частности, в таблице MBR нет столбца `Name` (Имя), поскольку в этой схеме имена отсутствуют. (Я произвольно указал имена `myfirst` и `mysecond` в таблице GPT.)

Таблица MBR в данном примере содержит основной, расширенный и логический разделы. *Основной раздел* является подразделом диска; раздел 1 — пример тому. В основной таблице MBR предельное количество основных разделов равно четырем. Если вам необходимо больше четырех разделов, вы обозначаете один из них как *расширенный раздел*. Затем вы делите расширенный раздел на *логические разделы*, которые операционная система может использовать подобно любому другому разделу.

В данном примере раздел 2 является расширенным разделом, который содержит логический раздел 5.

ПРИМЕЧАНИЕ

Файловая система, которую выводит команда parted, это не обязательно та система, что определена в поле идентификатора в большинстве записей таблицы MBR. Этот идентификатор является числом; например, 83 — это раздел Linux, а 82 — область подкачки Linux. Таким образом, команда parted пытается самостоятельно определить файловую систему. Если вам необходимо абсолютно точно узнать идентификатор системы для таблицы MBR, используйте команду fdisk -l.

Первичное чтение ядром

При первичном чтении таблицы MBR ядро Linux выдает следующий отладочный результат (вспомните, что увидеть его можно с помощью команды `dmesg`):

```
sda: sda1 sda2 < sda5 >
```

Фрагмент `sda2 < sda5 >` означает, что устройство `/dev/sda2` является расширенным разделом, который содержит один логический раздел, `/dev/sda5`. Как правило, вы будете игнорировать расширенные разделы, поскольку вам будет нужен доступ только к внутренним логическим разделам.

4.1.2. Изменение таблиц разделов

Просмотр таблиц разделов — операция сравнительно простая и безвредная. Изменение таблиц разделов также осуществляется довольно просто, однако при таком типе изменений диска могут возникнуть опасности. Имейте в виду следующее.

- Изменение таблицы разделов сильно усложняет восстановление любых данных в удаляемых разделах, поскольку при этом меняется начальная точка привязки файловой системы. Обязательно создавайте резервную копию диска, на котором вы меняете разделы, если он содержит важную информацию.
- Убедитесь в том, что на целевом диске ни один из разделов в данный момент не используется. Это важно, поскольку в большинстве версий Linux автоматически монтируется любая обнаруженная файловая система (подробности о монтировании и демонтировании см. в подразделе 4.2.3).

Когда вы будете готовы, выберите для себя команду для работы с разделами. Если вы предпочитаете применять команду `parted`, то можете воспользоваться утилитой командной строки или таким графическим интерфейсом, как `gparted`. Для интерфейса в стиле команды `fdisk` воспользуйтесь командой `gdisk`, если вы работаете с разделами GPT. Все эти утилиты обладают интерактивной справкой и просты в освоении. Попробуйте применить их для флеш-накопителя или какого-либо подобного устройства, если у вас нет свободных дисков.

Существуют различия в том, как работают команды `fdisk` и `parted`. С помощью команды `fdisk` вы создаете новую таблицу разделов до выполнения реальных изменений на диске; команда `fdisk` только осуществляет их, когда вы выходите из нее. При использовании команды `parted` разделы создаются, изменяются и удаляются, *когда вы вводите команды*. У вас нет возможности просмотреть таблицу разделов до ее изменения.

Эти различия важны также для понимания того, как данные утилиты взаимодействуют с ядром. Команды `fdisk` и `parted` изменяют разделы полностью в пространстве пользователя; нет необходимости, чтобы ядро обеспечивало поддержку перезаписи таблицы разделов, поскольку пространство пользователя способно считывать и изменять все данные на блочном устройстве.

Однако в конечном счете ядро все же должно считывать таблицу разделов, чтобы представить разделы как блочные устройства. Утилита `fdisk` использует сравнительно простой метод: после изменения таблицы разделов эта команда осуществляет единичный системный вызов к диску, чтобы сообщить ядру о необходимости

повторного считывания таблицы разделов. После этого ядро генерирует отладочный вывод, который можно просмотреть с помощью команды `dmesg`. Например, если вы создаете два раздела на устройстве `/dev/sdf`, вы увидите следующее:

```
sdf: sdf1 sdf2
```

В сравнении с этой командой инструменты `parted` не используют системный вызов для всего диска. Вместо него они сигнализируют ядру об изменении отдельных разделов. После обработки изменения одного раздела ядро не производит приведенного выше отладочного вывода.

Есть несколько способов увидеть изменения разделов.

- Используйте команду `udevadm`, чтобы отследить изменения событий ядра. Например, команда `udevadm monitor --kernel` покажет удаленные устройства-разделы и добавленные новые.
- Посмотрите полную информацию о разделах в файле `/proc/partitions`.
- Поищите в каталоге `/sys/block/device/` измененные системные интерфейсы разделов или в каталоге `/dev` — измененные устройства-разделы.

Если вы хотите быть абсолютно уверенными в том, что таблица разделов изменена, можно выполнить «старомодный» системный вызов, который применяет команда `fdisk`, использовав команду `blockdev`. Например, чтобы ядро принудительно перезагрузило таблицу разделов на устройстве `/dev/sdf`, запустите следующую команду:

```
# blockdev --rereadpt /dev/sdf
```

На данный момент вы знаете все необходимое о работе с разделами дисков. Если вам интересно изучить некоторые дополнительные подробности о дисках, продолжайте чтение. В противном случае переходите к разделу 4.2, чтобы узнать о размещении файловой системы на диске.

4.1.3. Диск и геометрия раздела

Любое устройство с подвижными частями добавляет сложностей в систему программного обеспечения, поскольку физические элементы сопротивляются абстрагированию. Жесткие диски не являются исключением. Хоть и возможно представлять жесткий диск как блочное устройство с произвольным доступом к любому блоку, возникают серьезные последствия для производительности, если вы не позаботились о том, как располагаются данные на диске. Рассмотрим физические свойства простого диска с одной пластиной, изображенного на рис. 4.3.

Диск состоит из вращающейся на шпинделе пластины, а также головки, которая прикреплена к подвижному кронштейну, который может перемещаться вдоль радиуса диска. Когда диск вращается под головкой, последняя считывает данные. Когда кронштейн расположен в определенной позиции, головка может считывать данные только с одной окружности. Эта окружность называется *цилиндром*, поскольку у больших дисков несколько пластин, которые надеты на один шпиндель и вращаются вокруг него. Каждая пластина может иметь одну или две головки, для верхней и/или нижней части пластины, причем все головки крепятся на одном

кронштейне и перемещаются совместно. Поскольку кронштейн двигается, на диске есть много цилиндров, от самых малых около центра диска до самых больших по его краям. Наконец, цилиндр можно разделить на доли, называемые *секторами*. Такой способ представления геометрии диска называется *CHS* (cylinder-head-sector, цилиндр-головка-сектор).

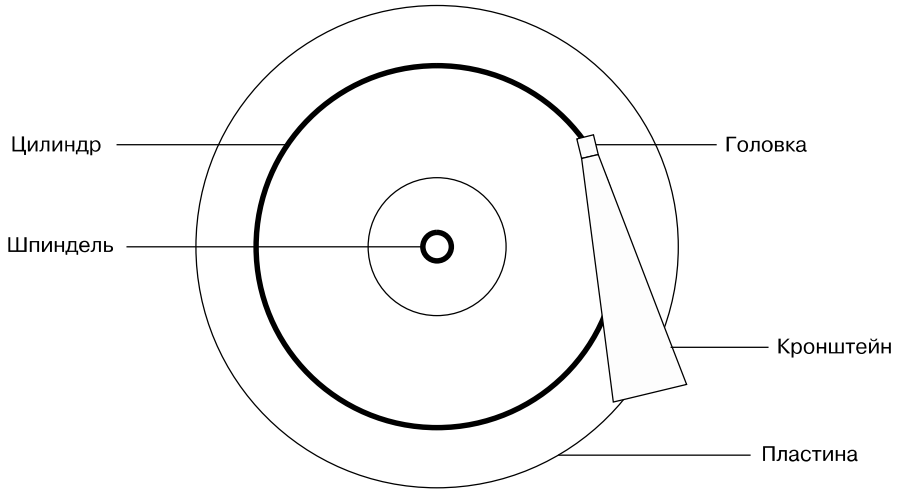

Рис. 4.3. Жесткий диск, вид сверху

ПРИМЕЧАНИЕ

Дорожка является частью цилиндра, к которой имеет доступ одна головка, поэтому на рис. 4.3 цилиндр является также и дорожкой.

Ядро и различные программы для работы с разделами могут сообщить вам о том, что́ из себя представляет диск как совокупность цилиндров (и *секторов*, которые являются частями цилиндров). Однако для современных жестких дисков *сообщаемые значения являются фиктивными*! Традиционная схема адресации, которая использует параметры CHS, не вписывается в современное аппаратное обеспечение жестких дисков. Она также не принимает в расчет тот факт, что в одних цилиндрах можно разместить больше данных, чем в других. Дисковые аппаратные средства поддерживают *блочную адресацию LBA (Logical Block Addressing)*, чтобы просто обращаться к какому-либо месту диска по номеру блока. Однако следы системы CHS еще присутствуют. Например, таблица разделов MBR содержит информацию CHS, а также ее LBA-эквивалент, и некоторые загрузчики системы по-прежнему довольно глупы, чтобы доверять значениям CHS (но не беспокойтесь — в большинстве загрузчиков Linux используются значения LBA).

Тем не менее понятие о цилиндрах оказалось важным для работы с разделами, поскольку цилиндры являются идеальными границами для разделов. Чтение потока данных с цилиндра происходит очень быстро, так как головка может непрерывно считывать данные по мере вращения диска. Раздел, который организован как набор смежных цилиндров, также позволяет получить быстрый доступ

к данным, поскольку головке не приходится перемещаться слишком далеко между цилиндрами.

Некоторые программы для работы с разделами выражают недовольство, если вы не размечаете разделы точно по границам цилиндров. Игнорируйте это. Вы мало чем сможете помочь, поскольку значения CHS для современных дисков попросту недостоверны. Схема LBA гарантирует вам то, что разделы окажутся именно там, где вы предполагали.

4.1.4. Твердотельные накопители (диски SSD)

Устройства хранения без движущихся частей, такие как *твердотельные накопители* (*SSD*), совершенно отличны от вращающихся дисков, если говорить о характеристиках доступа к данным. Для них произвольный доступ не является проблемой, так как отсутствует перемещающаяся вдоль пластины головка. Однако некоторые факторы отражаются на производительности.

Одним из наиболее значимых факторов, влияющих на производительность дисков SSD, является *выравнивание разделов*. Когда вы считываете данные с диска SSD, чтение происходит фрагментарно — как правило, порциями по 4096 байт за один прием, — причем такое чтение должно начинаться с числа, кратного этому размеру. Поэтому, если раздел и данные в нем не располагаются в пределах 4096-байтной зоны, вам может понадобиться выполнить две небольшие операции чтения вместо одной, например чтения содержимого каталога.

Многие утилиты для работы с разделами (например, `parted` и `gparted`) содержат средства для размещения вновь созданных разделов с правильными отступами от начала диска, и вам никогда не придется беспокоиться о неверном выравнивании разделов. Однако, если вам любопытно узнать, где начинаются ваши разделы, чтобы убедиться в том, что они начинаются от границ, можно легко это выяснить, заглянув в каталог `/sys/block`. Вот пример раздела для устройства `/dev/sdf2`:

```
$ cat /sys/block/sdf/sdf2/start
1953126
```

Этот раздел начинается на расстоянии 1 953 126 байт от начала диска. Поскольку это число не делится нацело на 4096, работа с таким разделом не достигала бы оптимальной производительности, если бы он был расположен на диске SSD.

4.2. Файловые системы

Последним звеном между ядром и пространством пользователя для дисков обычно является *файловая система*. С ней вы привыкли взаимодействовать, когда запускали такие команды, как `ls` и `cd`. Как отмечалось ранее, файловая система является разновидностью базы данных; она поддерживает структуру, призванную трансформировать простое блочное устройство в замысловатую иерархию файлов и подкаталогов, которую пользователи способны понять.

В свое время файловые системы, располагавшиеся на дисках и других физических устройствах, использовались исключительно для хранения данных. Однако

древовидная структура каталогов, а также интерфейс ввода-вывода довольно гибки, поэтому теперь файловые системы выполняют множество задач, например роль системных интерфейсов, которые вы можете увидеть в каталогах /sys и /proc. Файловые системы традиционно реализованы внутри ядра, однако инновационный протокол 9P из операционной системы Plan 9 (http://plan9.bell-labs.com/sys/doc/9.html) способствовал разработке файловых систем в пространстве пользователя. Функция *FUSE* (File System in User Space, файловая система в пространстве пользователя) позволяет применять такие файловые системы в Linux.

Слой абстракции *VFS* (виртуальная файловая система) завершает реализацию файловой системы. Во многом подобно тому, как подсистема SCSI стандартизирует связь между различными типами устройств и управляющими командами ядра, слой VFS обеспечивает поддержку стандартного интерфейса всеми реализациями файловых систем, чтобы приложения из пространства пользователя одинаковым образом обращались с файлами и каталогами. Виртуальная файловая система позволяет Linux поддерживать невообразимо большое число файловых систем.

4.2.1. Типы файловых систем

В Linux включена поддержка таких файловых систем, как «родные» разработки, оптимизированные для Linux, «чужеродные» типы, например семейство Windows FAT, универсальные файловые системы вроде ISO 9660 и множество других. В приведенном ниже списке перечислены наиболее распространенные типы файловых систем для хранения данных. Имена типов систем, как их определяет Linux, приведены в скобках после названия файловых систем.

- *Четвертая расширенная файловая система* (ext4) является текущей реализацией в линейке «родных» для Linux файловых систем. *Вторая расширенная файловая система* (ext2) долгое время была системой по умолчанию в системах Linux, которые испытывали влияние традиционных файловых систем Unix, таких как файловая система Unix (UFS, Unix File System) и быстрая файловая система (FFS, Fast File System). В *третьей расширенной файловой системе* (ext3) появился режим журналирования (небольшой кэш за пределами нормальной структуры данных файловой системы) для улучшения целостности данных и ускорения загрузки системы. Файловая система ext4 является дальнейшим улучшением, с поддержкой файлов большего размера по сравнению с допустимым в системах ext2 или ext3, а также большего количества подкаталогов.

  Среди расширенных файловых систем присутствует некоторая доля обратной совместимости. Например, можно смонтировать систему ext2 как ext3 или наоборот, а также смонтировать файловые системы ext2 и ext3 как ext4, однако нельзя смонтировать файловую систему ext4 как ext2 или ext3.

- *Файловая система ISO 9660* (iso9660) — это стандарт для дисков CD-ROM. Большинство дисков CD-ROM использует какой-либо вариант стандарта ISO 9660.

- *Файловые системы FAT* (msdos, vfat, umsdos) относятся к системам Microsoft. Простой тип msdos поддерживает весьма примитивное унылое многообразие систем MS-DOS. Для большинства современных файловых систем Windows следует использовать тип vfat, чтобы получить возможность полного доступа

из ОС Linux. Редко используемый тип umsdos представляет интерес для Linux: в нем есть поддержка таких особенностей Unix, как символические ссылки, которые находятся над файловой системой MS-DOS.

- *Tun HFS+* (hfsplus) является стандартом Apple, который используется в большинстве компьютеров Macintosh.

Хотя расширенные файловые системы были абсолютно пригодны для применения обычными пользователями, в технологии файловых систем были произведены многочисленные улучшения, причем такие, что даже система ext4 не может ими воспользоваться в силу требований обратной совместимости. Эти улучшения относятся главным образом к расширяемости системы, как то: очень большое количество файлов, файлы большого объема и другие подобные вещи. Новые файловые системы Linux, такие как Btrfs, находятся в разработке и могут прийти на смену расширенным файловым системам.

4.2.2. Создание файловой системы

Когда вы завершите работу с разделами, которая описана выше (см. раздел 4.1), можно создавать файловую систему. Как и для разделов, это выполняется в пространстве пользователя, поскольку процесс из пространства пользователя может напрямую обращаться к блочному устройству и работать с ним. Утилита `mkfs` способна создать многие типы файловых систем. Например, можно создать раздел типа ext4 в устройстве `/dev/sdf2` с помощью такой команды:

```
# mkfs -t ext4 /dev/sdf2
```

Команда `mkfs` автоматически определяет количество блоков в устройстве и устанавливает некоторые разумные параметры по умолчанию. Если вы не в полной мере представляете, что делаете, или если не любите читать подробную документацию, не меняйте эти настройки.

При создании файловой системы команда `mkfs` осуществляет диагностический вывод, включая и тот, который относится к *суперблоку*. Суперблок является ключевым компонентом, расположенным на верхнем уровне базы данных файловой системы. Он настолько важен, что утилита `mkfs` создает для него несколько резервных копий на случай утраты оригинала. Постарайтесь записать несколько номеров резервных копий суперблока во время работы команды `mkfs`, они могут вам понадобиться, когда придется восстанавливать суперблок после ошибки диска (см. подраздел 4.2.11).

ВНИМАНИЕ

Создание файловой системы — это задача, которую необходимо выполнять только после добавления нового диска или изменения разделов на существующем. Файловую систему следует создавать лишь один раз для каждого нового раздела, на котором еще нет данных (или который содержит данные, подлежащие удалению). Создание новой файловой системы поверх уже существующей фактически уничтожает старые данные.

Оказывается, утилита `mkfs` является только «лицевой стороной» набора команд для создания файловых систем. Эти команды называются `mkfs.`*`fs`*, где вместо *`fs`* подставлен тип файловой системы. Таким образом, когда вы запускаете команду `mkfs -t ext4`, утилита `mkfs`, в свою очередь, запускает команду `mkfs.ext4`.

Но двуличности здесь еще больше. Исследуйте файлы, которые скрываются за обозначениями `mkfs.*`, и вы увидите следующее:

```
$ ls -l /sbin/mkfs.*
-rwxr-xr-x 1 root root 17896 Mar 29 21:49 /sbin/mkfs.bfs
-rwxr-xr-x 1 root root 30280 Mar 29 21:49 /sbin/mkfs.cramfs
lrwxrwxrwx 1 root root     6 Mar 30 13:25 /sbin/mkfs.ext2 -> mke2fs
lrwxrwxrwx 1 root root     6 Mar 30 13:25 /sbin/mkfs.ext3 -> mke2fs
lrwxrwxrwx 1 root root     6 Mar 30 13:25 /sbin/mkfs.ext4 -> mke2fs
lrwxrwxrwx 1 root root     6 Mar 30 13:25 /sbin/mkfs.ext4dev -> mke2fs
-rwxr-xr-x 1 root root 26200 Mar 29 21:49 /sbin/mkfs.minix
lrwxrwxrwx 1 root root     7 Dec 19  2011 /sbin/mkfs.msdos -> mkdosfs
lrwxrwxrwx 1 root root     6 Mar  5  2012 /sbin/mkfs.ntfs -> mkntfs
lrwxrwxrwx 1 root root     7 Dec 19  2011 /sbin/mkfs.vfat -> mkdosfs
```

Файл `mkfs.ext4` является лишь символической ссылкой на `mke2fs`. Об этом важно помнить, если вы натолкнетесь на какую-либо систему без специальной команды `mkfs` или же когда станете искать документацию по какой-либо файловой системе. Каждой утилите для создания файловой системы посвящена особая страница в руководстве, например, mke2fs(8). В большинстве версий ОС это не создаст проблем, поскольку при попытке доступа к странице mkfs.ext4(8) руководства вы будете перенаправлены на страницу mke2fs(8). Просто имейте это в виду.

4.2.3. Монтирование файловой системы

В Unix процесс присоединения файловой системы называется *монтированием*. Когда система загружается, ядро считывает некоторые конфигурационные данные и на их основе монтирует корневой каталог (/).

Чтобы выполнить монтирование файловой системы, вы должны знать следующее:

- устройство для размещения файловой системы (например, раздел диска; на нем будут располагаться актуальные данные файловой системы);
- тип файловой системы;
- *точку монтирования*, то есть место в иерархии каталогов текущей системы, куда будет присоединена файловая система. Точка монтирования всегда является обычным каталогом. Например, можно использовать каталог `/cdrom` в качестве точки монтирования для приводов CD-ROM. Точка монтирования не обязана находиться именно в корневом каталоге, в системе она может быть где угодно.

Для монтирования файловой системы применяется терминология «смонтировать устройство в точке монтирования». Чтобы узнать статус текущей файловой системы, запустите команду `mount`. Результат будет выглядеть примерно так:

```
$ mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
```

```
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
--snip--
```

Каждая строка соответствует одной файловой системе, смонтированной в настоящее время. Перечислены следующие элементы:

- устройство, например `/dev/sda3`. Обратите внимание на то, что некоторые устройства в действительности не являются таковыми (например, `proc`), а играют роль заместителей для имен реальных устройств, поскольку таким файловым системам специального назначения не нужны устройства;
- слово `on`;
- точка монтирования;
- слово `type`;
- тип файловой системы, как правило, в виде краткого идентификатора;
- параметры монтирования (в скобках) (см. подробности в подразделе 4.2.6).

Чтобы смонтировать файловую систему, используйте приведенную ниже команду `mount`, указав тип файловой системы, устройство и желаемую точку монтирования:

```
# mount -t type device mountpoint
```

Чтобы, например, смонтировать четвертую расширенную файловую систему `/dev/sdf2` в точке `/home/extra`, используйте такую команду:

```
# mount -t ext4 /dev/sdf2 /home/extra
```

Обычно не требуется указывать параметр `-t`, поскольку команда `mount` способна догадаться о нем сама. Однако иногда бывает необходимо сделать различие между сходными типами файловых систем, таких как FAT, например.

В подразделе 4.2.6 можно увидеть еще несколько более длинных параметров монтирования. Чтобы демонтировать (открепить) файловую систему, воспользуйтесь командой `umount`:

```
# umount mountpoint
```

Можно также демонтировать файловую систему вместе с ее устройством, а не с точкой монтирования.

4.2.4. Файловая система UUID

Метод монтирования файловых систем, рассмотренный в предыдущем разделе, зависит от названий устройств. Однако имена устройств могут измениться, поскольку они зависят от порядка их обнаружения ядром. Чтобы справиться с этой проблемой, можно идентифицировать и монтировать файловые системы по их *идентификатору UUID* (Universally Unique Identifier, универсальный уникальный идентификатор), который является стандартом в программном обеспечении. Идентификатор UUID — это своего рода серийный номер, причем каждый такой номер уникален. Команды

для создания файловых систем, такие как mke2fs, присваивают идентификатор UUID при инициализации структуры данных файловой системы.

Чтобы просмотреть список устройств, соответствующих им файловых систем, а также идентификаторы UUID, используйте команду blkid (block ID):

```
# blkid
/dev/sdf2: UUID="a9011c2b-1c03-4288-b3fe-8ba961ab0898" TYPE="ext4"
/dev/sda1: UUID="70ccd6e7-6ae6-44f6-812c-51aab8036d29" TYPE="ext4"
/dev/sda5: UUID="592dcfd1-58da-4769-9ea8-5f412a896980" TYPE="swap"
/dev/sde1: SEC_TYPE="msdos" UUID="3762-6138" TYPE="vfat"
```

В этом примере команда blkid обнаружила четыре раздела с данными: два из них с файловой системой ext4, один с сигнатурой области подкачки (см. раздел 4.3) и один с файловой системой семейства FAT. Все собственные разделы Linux снабжены стандартными идентификаторами UUIDs, однако у раздела FAT он отсутствует. К разделу FAT можно обратиться с помощью серийного номера тома FAT (в данном случае это 3762-6138).

Чтобы смонтировать файловую систему по ее идентификатору UUID, используйте синтаксис UUID=. Например, для монтирования первой файловой системы из приведенного выше списка в точке /home/extra введите такую команду:

```
# mount UUID=a9011c2b-1c03-4288-b3fe-8ba961ab0898 /home/extra
```

Как правило, монтировать файловые системы вручную по их идентификаторам не придется, поскольку вам, вероятно, известно устройство, а смонтировать устройство по его имени гораздо проще, чем использовать безумный номер UUID. Однако все же важно понимать суть идентификаторов UUID. С одной стороны, они являются предпочтительным средством для автоматического монтирования файловых систем в точке /etc/fstab во время загрузки системы (см. раздел 4.2.8). Помимо этого, многие версии ОС используют идентификатор UUID в качестве точки монтирования, когда вы вставляете сменный носитель данных. В приведенном выше примере файловая система FAT находится на флеш-карте. Ubuntu, если какой-либо пользователь зашел в нее, смонтирует данный раздел в точке /media/3762-6138 после вставки носителя. Демон udevd, описанный в главе 3, обрабатывает начальное событие для вставки устройства.

Если необходимо, можно изменить идентификатор UUID для файловой системы (например, если вы скопировали всю файловую систему куда-либо еще, и теперь вам необходимо отличать ее от оригинала). Обратитесь к странице tune2fs(8) руководства, чтобы узнать о том, как это выполнить в файловых системах ext2/ext3/ext4.

4.2.5. Буферизация диска, кэширование и файловые системы

Система Linux, подобно другим версиям Unix, выполняет буферизацию при записи на диск. Это означает, что ядро обычно не сразу же вносит изменения в файловую систему, когда процессы запрашивают их. Вместо этого ядро хранит такие изменения в оперативной памяти до тех пор, пока ядро не сможет с удобством

выполнить реальные изменения на диске. Такая система буферизации очевидна для пользователя и улучшает производительность.

Когда вы демонтируете файловую систему с помощью команды `umount`, ядро автоматически синхронизируется с диском. В любой другой момент времени можно выполнить принудительную запись изменений из буфера ядра на диск, запустив команду `sync`. Если по каким-либо причинам невозможно демонтировать файловую систему до выхода из операционной системы, обязательно запустите сначала команду `sync`.

Кроме того, ядро располагает рядом механизмов, использующих оперативную память, чтобы автоматически кэшировать блоки, считанные с диска. Следовательно, если один или несколько процессов часто обращаются к какому-либо файлу, то ядру не приходится снова и снова получать доступ к диску — оно может просто выполнить чтение из кэша, экономя время и системные ресурсы.

4.2.6. Параметры монтирования файловой системы

Существует множество способов изменить режим работы команды `mount`, поскольку часто бывает необходимо поработать со съемными накопителями или выполнить обслуживание системы. Общее число параметров команды поражает. Исчерпывающее руководство на странице mount(8) является хорошей справкой, но при этом трудно понять, с чего следует начать, а чем можно пренебречь. В данном разделе вы увидите наиболее полезные параметры.

Параметры разделены на две категории:

- общие параметры. Содержат флаг `-t` для указания типа файловой системы;
- параметры, зависящие от файловой системы. Относятся только к определенным типам файловых систем.

Чтобы задействовать параметр для какой-либо файловой системы, используйте перед ним флаг `-o`. Например, параметр `-o norock` отключает расширения Rock Ridge в файловой системе ISO 9660, однако для любой другой файловой системы он не имеет смысла.

Короткие параметры

Наиболее важные общие параметры таковы.

- `-r` — монтирует файловую систему в режиме «только для чтения». Это может пригодиться в разных случаях: начиная с защиты от записи и заканчивая самозагрузкой. Нет необходимости указывать данный параметр при доступе к такому устройству, как CD-ROM, система сделает это за вас (а также уведомит о том, что статус устройства — только для чтения).
- `-n` — гарантирует то, что команда `mount` не будет пытаться обновить исполняемую системную базу данных монтирования `/etc/mtab`. Операция монтирования прерывается, если она не может производить запись в данный файл, а это важно во время загрузки системы, поскольку корневой раздел (и, следовательно, системная база данных монтирования) поначалу доступен только для чтения. Этот параметр может быть полезен, когда вы будете пытаться исправить

системную ошибку в режиме одиночного пользователя, поскольку в этот момент системная база данных монтирования не будет доступна.

- `-t` — задает тип файловой системы.

Длинные параметры

Короткие параметры, вроде `-r`, слишком коротки для постоянно увеличивающегося количества параметров монтирования. К тому же в алфавите не так много букв, чтобы обозначить ими все возможные параметры. Короткие параметры могут также вызвать проблемы, поскольку на основе одной буквы сложно определить значение параметра. Для многих общих параметров, а также для всех параметров, которые зависят от файловой системы, используется более длинный и гибкий формат параметров.

Чтобы применять длинные параметры для утилиты `mount` в командной строке, начните с флага `-o` и добавьте несколько ключевых слов. Вот пример полной команды, снабженной длинными параметрами после флага `-o`:

```
# mount -t vfat /dev/hda1 /dos -o ro,conv=auto
```

Здесь присутствуют два длинных параметра: `ro` и `conv=auto`. Параметр `ro` задает режим «только чтение» и эквивалентен короткому параметру `-r`. Параметр `conv=auto` дает ядру указание об автоматической конвертации определенных текстовых файлов из формата DOS с переводом строки в формат Unix (совсем скоро вы узнаете об этом).

Наиболее полезны следующие длинные параметры:

- `exec`, `noexec` — включает или отключает исполнение команд над файловой системой;
- `suid`, `nosuid` — включает или отключает команды `setuid` (установка идентификатора пользователя);
- `ro` — монтирует файловую систему в режиме «только чтение» (подобно короткому параметру `-r`);
- `rw` — монтирует файловую систему в режиме «чтение-запись»;
- `conv=`*`rule`* (для файловых систем на основе FAT) — конвертирует содержащиеся в файлах символы перевода строки, в зависимости от атрибута *`rule`*, который может принимать значения `binary`, `text` или `auto`. По умолчанию установлено значение `binary`, при котором отключена конвертация символов. Чтобы трактовать все файлы как текстовые, используйте значение `text`. Если указать значение `auto`, конвертация файлов будет происходить на основе их расширения. Например, файл `.jpg` обрабатываться не будет, а файл `.txt` пройдет специальную обработку. Будьте осторожны с этим параметром, поскольку он может повредить файлы. Постарайтесь применять его в режиме «только чтение».

4.2.7. Демонтирование файловой системы

Иногда может возникнуть необходимость заново присоединить недавно смонтированную файловую систему к той же точке монтирования, изменив при этом параметры монтирования. Чаще всего это случается, когда вам необходимо открыть доступ на запись в файловой системе во время восстановления после сбоя.

Следующая команда заново монтирует корневой каталог в режиме «чтение-запись» (параметр `-n` необходим, поскольку команда `mount` не может вести запись в системную базу данных монтирования, если корневой каталог находится в режиме «только чтение»):

```
# mount -n -o remount /
```

Эта команда подразумевает, что корректный перечень устройств для корневого каталога расположен в каталоге `/etc/fstab` (о чем будет сказано в следующем разделе). Если это не так, следует указать устройство.

4.2.8. Таблица файловой системы /etc/fstab

Чтобы смонтировать файловые системы во время загрузки, а также избавить команду `mount` от нудной работы, Linux постоянно хранит список файловых систем и их параметров в таблице `/etc/fstab`. Это файл в обычном текстовом формате, достаточно простом, как можно увидеть из примера 4.1.

Пример 4.1. Список файловых систем и их параметров в файле /etc/fstab

```
proc /proc proc nodev,noexec,nosuid 0 0
UUID=70ccd6e7-6ae6-44f6-812c-51aab8036d29 / ext4 errors=remount-ro 0 1
UUID=592dcfd1-58da-4769-9ea8-5f412a896980 none swap sw 0 0
/dev/sr0 /cdrom iso9660 ro,user,nosuid,noauto 0 0
```

Каждая строка, содержащая шесть полей, соответствует одной файловой системе. Ниже перечислены эти поля (слева направо).

- **Устройство или идентификатор UUID.** Большинство современных систем Linux больше не использует устройство в файле `/etc/fstab`, предпочитая идентификатор UUID. Обратите внимание на то, что запись `/proc` содержит устройство-заместитель с именем `proc`.
- **Точка монтирования.** Указывает, где присоединяется файловая система.
- **Тип файловой системы.** Скорее всего, вам незнаком параметр `swap` в данном перечне; это раздел подкачки (см. раздел 4.3).
- **Параметры.** Использованы длинные параметры, разделенные запятыми.
- **Информация о резервной копии для использования командой сброса.** В этом поле всегда следует указывать значение `0`.
- **Порядок проверки целостности системы.** Чтобы команда `fsck` всегда начинала работу с корневого каталога, устанавливайте в этом поле значение `1` для корневой файловой системы и значение `2` для остальных файловых систем на жестком диске. Используйте значение `0`, чтобы отключить при запуске проверку чего-либо еще, включая приводы CD-ROM, область подкачки и файловую систему `/proc` (о команде `fsck` можно узнать в подразделе 4.2.11).

При использовании команды `mount` можно применять некоторые обходные пути, если файловая система, с которой вы желаете работать, есть в таблице `/etc/fstab`. Если бы, например, вы использовали систему из листинга 4.1 и монтировали CD-ROM, можно было бы просто запустить команду `mount /cdrom`.

Можно также попытаться смонтировать разом все компоненты, перечисленные в таблице /etc/fstab (если они не снабжены параметром noauto), с помощью такой команды:

```
# mount -a
```

Пример 4.1 содержит несколько новых параметров, а именно: errors, noauto и user, поскольку они не применяются вне файла /etc/fstab. Кроме того, вам часто будет встречаться здесь параметр defaults. Перечисленные параметры означают следующее.

- defaults. Используются параметры mount команды по умолчанию: режим «чтение-запись», применение файлов устройств, исполняемых файлов, бита setuid и т. п. Используйте этот параметр, когда вам не нужно специальным образом настраивать файловую систему, однако необходимо заполнить все поля в таблице /etc/fstab.
- errors. Этот параметр, относящийся к файловой системе ext2, определяет поведение ядра, когда операционная система испытывает сложности при монтировании файловой системы. По умолчанию обычно указан вариант errors=continue, который означает, что ядро должно возвратить код ошибки и продолжить работу. Чтобы заставить ядро выполнить монтирование заново в режиме «только чтение», используйте вариант errors=remount-ro. Вариант errors=panic говорит ядру (и вашей системе) о том, что необходимо выполнить останов, когда возникают проблемы с монтированием.
- noauto. Этот параметр сообщает команде mount -a, что данную запись следует игнорировать. Используйте его, чтобы предотвратить во время загрузки системы монтирование сменных накопителей, например дисков CD-ROM или флоппи-дисков.
- user. Данный параметр позволяет пользователям без специальных прав доступа запускать команду mount для какой-либо отдельной записи, что может быть удобно для предоставления доступа к приводам CD-ROM. Поскольку пользователи могут разместить корневой файл setuid на сменном носителе с другой системой, данный параметр устанавливает также атрибуты nosuid, noexec и nodev (чтобы исключить специальные файлы устройств).

4.2.9. Альтернативы таблицы /etc/fstab

Хотя файл /etc/fstab традиционно применяется для представления файловых систем и их точек монтирования, появилось два альтернативных способа. Первый — это каталог /etc/fstab.d, который содержит отдельные файлы конфигурации файловой системы (по одному на каждую файловую систему). Идея очень похожа на многие другие конфигурационные каталоги, которые встретятся вам в этой книге.

Второй способ — конфигурирование модулей демона systemd для файловых систем. Подробности о демоне systemd и его модулях вы узнаете из главы 6. Тем не менее конфигурация модуля systemd часто исходит из таблицы /etc/fstab (или основана на ней), поэтому в вашей системе могут встретиться некоторые частичные совпадения.

4.2.10. Мощность файловой системы

Чтобы увидеть размеры и степень использования смонтированных в данный момент файловых систем, воспользуйтесь командой `df`. Результат ее работы может выглядеть так:

```
$ df
Filesystem    1024-blocks     Used    Available  Capacity Mounted on
/dev/sda1         1011928    71400      889124      7%   /
/dev/sda3        17710044  9485296     7325108     56%   /usr
```

Приведу краткое описание полей в этом выводе:

- `Filesystem` — устройство, на котором расположена файловая система;
- `1024-blocks` — общая мощность файловой системы в блоках по 1024 байта;
- `Used` — количество занятых блоков;
- `Available` — количество свободных блоков;
- `Capacity` — процент использованных блоков;
- `Mounted on` — точка монтирования.

Легко заметить, что эти две файловые системы занимают приблизительно 1 и 17,5 Гбайт. Однако значения мощности могут выглядеть немного странно, поскольку при сложении 71 400 и 889 124 не получается 1 011 928, а 9 485 296 не составляет 56 % от 17 710 044. В обоих случаях 5 % от общей мощности не учтены. На самом деле это пространство присутствует, но оно спрятано в *зарезервированных* блоках. Следовательно, только пользователь superuser может использовать все пространство файловой системы, если остальная часть раздела окажется заполненной. Такая особенность предотвращает немедленный отказ в работе системных серверов, когда заканчивается свободное пространство.

Если ваш диск заполнен и вы желаете знать, где расположены все эти пожирающие пространство медиафайлы, воспользуйтесь командой `du`. При запуске без аргументов эта команда выводит статистику использования диска для каждого каталога в иерархии каталогов, начиная с текущего рабочего каталога. Запустите команду `cd /;`, чтобы понять суть, остановите сочетанием клавиш Ctrl+C. Команда `du -s` работает в режиме общего подсчета и выводит только итоговую сумму. Чтобы проверить какой-либо один каталог, перейдите в него и запустите команду `du -s *`.

ПРИМЕЧАНИЕ

Стандарт POSIX (Portable Operating System Interface for Unix, переносимый интерфейс операционных систем Unix) определяет размер блока равным 512 байтам. Однако такой размер сложнее воспринимается при чтении, поэтому по умолчанию результаты работы команд df и du в большинстве версий Linux выражены в 1024-байтных блоках. Если вы настаиваете на отображении значений в виде 512-байтных блоков, задайте переменную окружения POSIXLY_CORRECT. Чтобы явно указать блоки размером 1024 байта, используйте параметр -k (его поддерживают обе утилиты). У команды df есть также параметр -m, чтобы отображать мощность в блоках размером 1 Мбайт, и параметр -h, который пытается выбрать наиболее удобное представление для чтения.

4.2.11. Проверка и восстановление файловых систем

Предлагаемые файловыми системами Unix оптимизации возможны благодаря замысловатому устройству базы данных. Чтобы файловые системы работали бесперебойно, ядро должно быть уверено в том, что в смонтированной файловой системе нет ошибок. Если они присутствуют, может произойти потеря данных и сбой в работе системы.

Ошибки в файловой системе обычно возникают в результате того, что пользователь грубым образом выходит из системы (например, выдергивая кабель электропитания). В подобных случаях кэш файловой системы в памяти может не соответствовать данным на диске, к тому же система может выполнять изменение файловой системы, когда вы подвергаете компьютер «встряске». Хотя новые поколения файловых систем снабжены журналированием, чтобы сделать их повреждение менее частым, вам всегда следует выходить из системы корректным образом. Вне зависимости от используемой файловой системы для поддержания ее стабильной работы необходимо регулярно выполнять проверки.

Инструмент для проверки файловой системы называется `fsck`. Подобно команде `mkfs`, у него существуют различные версии для каждого типа файловой системы, поддерживаемого Linux. Например, когда вы применяете команду `fsck` для расширенных файловых систем (ext2/ext3/ext4), она распознает тип файловой системы и запускает утилиту `e2fsck`. Следовательно, вам, как правило, не придется вручную вводить `e2fsck`, если только команда `fsck` не сможет выяснить тип файловой системы или вы ищете страницу руководства по команде `e2fsck`.

Информация, представленная в этом разделе, относится к расширенным файловым системам и команде `e2fsck`.

Чтобы запустить команду `fsck` в режиме интерактивного руководства, укажите в качестве аргумента устройство или точку монтирования (как они приведены в таблице `/etc/fstab`). Например, так:

```
# fsck /dev/sdb1
```

ВНИМАНИЕ

Никогда не используйте команду fsck для смонтированной файловой системы, поскольку ядро может изменить данные на диске во время работы проверки. Это вызовет несоответствия во время исполнения, которые могут привести к сбою системы и повреждению файлов. Есть всего одно исключение: если вы монтируете корневой раздел только для чтения в режиме единственного пользователя, то в этом разделе можно запустить команду fsck.

В режиме руководства команда `fsck` выводит подробные сообщения о проходах проверки, которые в случае отсутствия ошибок могут выглядеть так:

```
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information /dev/sdb1: 11/1976 files (0.0% non-
contiguous), 265/7891 blocks
```

Если в режиме руководства команда `fsck` обнаружит ошибку, она остановится и задаст вам вопрос, относящийся к устранению проблемы. Подобные вопросы касаются внутренней структуры файловой системы, например повторного подключения неприкрепленных дескрипторов `inode` или очистки блоков (дескриптор `inode` является строительным блоком файловой системы; о работе этих дескрипторов подробнее в разделе 4.5). Когда команда `fsck` спрашивает вас о повторном подключении дескриптора `inode`, это означает, что обнаружен файл, у которого, по-видимому, нет имени. При подключении такого файла команда `fsck` помещает его в каталог `lost+found` файловой системы, указав число в качестве имени файла. Если такое происходит, вам потребуется определить его имя, основываясь на содержимом файла; исходное имя файла будет, вероятно, утрачено.

Не имеет смысла дожидаться окончания процесса восстановления, если вы всего лишь некорректно вышли из системы, поскольку команда `fsck` может обнаружить большое число ошибок, подлежащих устранению. К счастью, у команды `e2fsck` есть параметр `-p`, который выполняет автоматическое исправление типичных проблем, не задавая вопросов, и прерывает работу только в случае серьезной ошибки. На самом деле версии Linux вовремя запускают какой-либо вариант команды `fsck -p`. Вам может также встретиться команда `fsck -a`, которая выполняет то же самое.

Если вы подозреваете, что в вашей системе есть такие проблемы, как отказ аппаратных средств или неправильная конфигурация устройств, вам необходимо определить порядок действий, поскольку команда `fsck` способна попросту превратить в кашу файловую систему с многочисленными ошибками. Одним из явных признаков того, что у вашей системы есть серьезные проблемы, является обилие вопросов, которые задает команда `fsck` в ручном режиме работы.

Попробуйте запустить команду `fsck -n`, чтобы проверить файловую систему, ничего не изменяя. Если возникла проблема в конфигурации устройства и вы думаете, что ее можно исправить (например, неправильное число блоков в таблице разделов или неплотно подключенные кабели), сделайте это до запуска команды `fsck` в реальном режиме. В противном случае вы можете потерять большое количество данных.

Если вы думаете, что поврежден лишь суперблок (например, когда кто-либо выполнил запись в начале дискового раздела), можно попробовать восстановить файловую систему на основе одной из резервных копий, созданных командой `mkfs`. Используйте команду `fsck -b num`, чтобы заменить поврежденный суперблок альтернативным блоком *num*.

Если вы не знаете, где искать резервную копию суперблока, можно запустить команду `mkfs -n` на данном устройстве, чтобы просмотреть список номеров резервных копий суперблока, не повреждая данные. Опять-таки убедитесь в том, что вы используете флаг `-n`, чтобы не разнести файловую систему на куски.

Проверка файловых систем ext3 и ext4

Обычно вам не понадобится вручную проверять файловые системы ext3 и ext4, поскольку целостность данных обеспечивается журналированием. Однако вам может потребоваться смонтировать «поломанную» файловую систему ext3 или ext4 в режиме ext2, поскольку ядро не станет монтировать такие файловые системы с непустым журналом. Если выход из системы был совершен некорректно, то журнал может

содержать какие-либо данные. Чтобы очистить журнал в файловой системе ext3 или ext4, приведя ее к стандартной базе данных, запустите следующую команду:

```
# e2fsck -fy /dev/disk_device
```

Наихудший случай

Самые серьезные дисковые проблемы оставляют вам не много вариантов для выбора.

- Можно попробовать извлечь из диска образ всей файловой системы с помощью команды `dd` и переместить ее в раздел другого диска с таким же размером.
- Можно попытаться «залатать» файловую систему, насколько это возможно, смонтировать ее в режиме «только чтение» и спасти что удастся.
- Можно попробовать команду `debugfs`.

В первом и во втором случаях вам все же понадобится исправить файловую систему до ее монтирования, если только вы не являетесь любителем ручного разбора сырых данных. Если желаете, можно ответить `y` на все вопросы команды `fsck`, запустив ее в таком виде: `fsck -y`. Однако используйте этот вариант в последнюю очередь, поскольку во время процесса восстановления могут обнаружиться ошибки, которые лучше исправлять вручную.

Инструмент `debugfs` позволяет вам просматривать файлы в файловой системе и копировать их куда-либо. По умолчанию он работает с файловыми системами в режиме «только чтение». Если вы восстанавливаете данные, вероятно, было бы неплохо оставить файлы неприкосновенными, чтобы избежать дальнейшей неразберихи.

Если же вы пришли в полное отчаяние в результате катастрофического сбоя и отсутствия резервных копий, вам остается только надеяться на то, что профессиональный сервис сможет «соскоблить данные с пластин».

4.2.12. Файловые системы специального назначения

Не все файловые системы представляют хранилища на физических носителях. В большинстве версий Unix есть файловые системы, которые играют роль системных интерфейсов. Такие файловые системы не только служат средством хранения данных на устройстве, но способны также представлять системную информацию, например идентификаторы процессов и диагностические сообщения ядра. Эта идея восходит к механизму `/dev`, который является ранней моделью использования файлов для интерфейсов ввода-вывода. Идея применения каталога `/proc` берет начало из восьмого издания экспериментальной версии Unix, которая была реализована Томом Дж. Киллианом (Tom J. Killian) и усовершенствована сотрудниками лаборатории Bell Labs (включая многих первичных разработчиков Unix), создавшими Plan 9 — экспериментальную операционную систему, которая вывела файловую систему на новый уровень абстракции (http://plan9.bell-labs.com/sys/doc/9.html).

Специальные типы файловых систем, которые широко применяются в Linux, включают следующие.

- **proc**. Смонтирована в каталоге `/proc`. Имя `proc` является сокращением слова *process* («процесс»). Каждый нумерованный каталог внутри `/proc` — это идентификатор происходящего в системе процесса; файлы в этих каталогах отражают различные характеристики процессов. Файл `/proc/self` представляет текущий процесс. Файловая система `proc` в Linux содержит внушительное количество дополнительной информации о ядре и аппаратных средствах в файлах вроде `/proc/cpuinfo`. Информация, которая не относится к процессам, перенесена из каталога `/proc` в каталог `/sys`.
- **sysfs**. Смонтирована в каталоге `/sys` (его вы встречали в главе 3).
- **tmpfs**. Смонтирована в каталоге `/run` и других. С помощью файловой системы tmpfs вы можете использовать физическую память и область подкачки в качестве временного хранилища. Например, можно смонтировать tmpfs там, где вам нравится, применяя длинные параметры `size` и `nr_blocks` для контроля максимального размера. Будьте осторожны и не помещайте постоянно данные в систему tmpfs, поскольку в итоге у операционной системы может закончиться память и программы начнут выходить из строя. Так, корпорация Sun Microsystems годами применяла вариант файловой системы tmpfs для каталога `/tmp`, что вызывало проблемы на компьютерах, работающих в течение долгого времени.

4.3. Область подкачки

Не каждый раздел диска содержит файловую систему. Пополнять оперативную память компьютера можно также за счет дискового пространства. Если оперативная память на исходе, система виртуальной памяти в Linux может автоматически перемещать фрагменты памяти на дисковое хранилище и обратно. Этот процесс называется *подкачкой* (свопингом), поскольку участки бездействующих команд перекачиваются на диск в обмен на активные фрагменты памяти, расположенные на диске. Область диска, используемая для хранения страниц памяти, называется *областью подкачки* (или сокращенно *swap*).

Результатом работы команды `free` является следующая информация о применении подкачки (в килобайтах):

```
$ free
              total        used        free
--snip--
Swap:         514072      189804      324268
```

4.3.1. Использование раздела диска в качестве области подкачки

Чтобы полностью применять раздел диска для подкачки, выполните следующее.

1. Убедитесь в том, что раздел пуст.
2. Запустите команду `mkswap` *dev*, в которой в качестве параметра *dev* укажите устройство с необходимым разделом. Эта команда помещает в данный раздел сигнатуру области подкачки.
3. Запустите команду `swapon dev`, чтобы зарегистрировать область с помощью ядра.

После создания раздела подкачки можно внести новую запись о нем в файл `/etc/fstab`, чтобы система смогла применять область подкачки уже при загрузке компьютера. Вот пример такой записи, в которой в качестве раздела подкачки использовано устройство `/dev/sda5`:

```
/dev/sda5 none swap sw 0 0
```

Принимайте во внимание то, что теперь многие системы применяют идентификаторы UUID вместо простых названий устройств.

4.3.2. Использование файла в качестве области подкачки

Можно применять обычный файл в качестве области подкачки, если возникнет ситуация, когда вы будете вынуждены изменить разделы диска, чтобы создать область подкачки. Вы не должны испытать при этом какие-либо сложности.

Воспользуйтесь приведенными ниже командами, чтобы создать пустой файл, инициализировать его для подкачки, а затем добавить в пул подкачки:

```
# dd if=/dev/zero of=swap_file bs=1024k count=num_mb
# mkswap swap_file
# swapon swap_file
```

Здесь параметр *swap_file* задает имя нового файла подкачки, а параметр *num_mb* — желаемый размер в мегабайтах.

Чтобы удалить раздел или файл подкачки из активного пула ядра, используйте команду `swapoff`.

4.3.3. Какой объем области подкачки необходим

В свое время, согласно традиционной мудрости Unix, полагали, что всегда следует резервировать для подкачки по меньшей мере в два раза больший объем памяти по сравнению с оперативной. Теперь этот вопрос не столь однозначен, поскольку стали доступны огромные объемы дискового пространства и оперативной памяти, а также изменились способы использования системы. С одной стороны, дисковое пространство настолько обширно, что возникает искушение выделить больше памяти, чем двойной размер оперативной памяти. С другой — вам, вероятно, никогда не придется использовать область подкачки, когда в наличии имеется столько реальной памяти.

Правило о «двойном размере реальной памяти» возникло в то время, когда к одному компьютеру могло быть подключено одновременно несколько пользователей. Не все они могли быть активны, поэтому было удобно переводить в область

подкачки те участки памяти, которые были выделены для неактивных пользователей, когда активному пользователю требовалось больше памяти.

Это может по-прежнему быть верным для компьютера с одним пользователем. Если вы запускаете много процессов, как правило, будет неплохо переместить в область подкачки части неактивных процессов или даже неактивные фрагменты активных процессов. Тем не менее, если вы постоянно применяете область подкачки, поскольку многие активные процессы желают сразу же использовать память, вы будете испытывать серьезные сложности с производительностью, так как дисковый ввод-вывод происходит слишком медленно и ему не угнаться за остальной частью системы. Выходы таковы: приобрести дополнительную память или завершить некоторые процессы.

Иногда ядро Linux может перевести в область подкачки какой-либо процесс с целью получения дополнительного дискового кэша. Чтобы это предотвратить, администраторы конфигурируют отдельные системы вообще без области подкачки. Например, высокопроизводительным сетевым серверам не следует использовать область подкачки и по возможности избегать обращения к диску.

ПРИМЕЧАНИЕ

Это опасно выполнять для компьютера общего назначения. Если на компьютере полностью будут исчерпаны оперативная память и область подкачки, ядро Linux запускает подавитель OOM (out-of-memory, нехватка памяти), чтобы прервать процесс и освободить некоторое количество памяти. Вы, безусловно, не захотите, чтобы это случилось с вашими приложениями. С другой стороны, высокопроизводительные серверы содержат сложные системы слежения и выравнивания нагрузки, чтобы никогда не оказаться в опасной зоне.

Из главы 8 вы узнаете подробнее о том, как работает система памяти.

4.4. Заглядывая вперед: диски и пространство пользователя

В относящихся к дискам компонентах системы Unix границы между пространством пользователя и пространством ядра довольно сложно определить. Как вы уже видели, ядро оперирует блочным вводом-выводом от устройств, а инструменты из пространства пользователя способны использовать блочный ввод-вывод с помощью файлов устройств. Тем не менее пространство пользователя, как правило, применяет блочный ввод-вывод только при инициализации таких операций, как создание разделов, файловых систем и области подкачки. В нормальном режиме пространство пользователя задействует только поддержку файловой системы, которая обеспечивается ядром в верхнем слое блочного ввода-вывода. Подобным образом ядро управляет и множеством мелких деталей, когда оно имеет дело с областью подкачки в системе виртуальной памяти.

В следующей части этой главы вкратце рассказано про внутренние части файловой системы Linux. Это более сложный материал, и вам определенно нет необходимости знать его, чтобы продолжить чтение книги. Переходите к следующей главе, чтобы начать изучение процесса загрузки системы Linux.

4.5. Внутри традиционной файловой системы

Традиционная файловая система Unix содержит два основных компонента: пул блоков данных, где можно хранить данные, и базу данных, которая управляет пулом данных. В основу базы данных положена структура данных inode. *Дескриптор inode* — это набор данных, который описывает конкретный файл, включая его тип, права доступа и (что, возможно, наиболее важно) расположение его данных в пуле данных. Дескрипторы inode распознаются по номерам, перечисленным в соответствующей таблице.

Имена файлов и каталогов также реализованы в виде дескрипторов inode. Дескриптор каталога содержит перечень имен файлов и соответствующих ссылок на другие дескрипторы.

Чтобы привести реальный пример, я создал новую файловую систему, смонтировал ее и сменил каталог на точку монтирования. После этого добавил несколько файлов и каталогов с помощью таких команд (попробуйте выполнить это самостоятельно на флеш-накопителе):

```
$ mkdir dir_1
$ mkdir dir_2
$ echo a > dir_1/file_1
$ echo b > dir_1/file_2
$ echo c > dir_1/file_3
$ echo d > dir_2/file_4
$ ln dir_1/file_3 dir_2/file_5
```

Обратите внимание на то, что я создал каталог `dir_2/file_5` в виде жесткой ссылки на каталог `dir_1/file_3`. Это означает, что данные два имени файлов на самом деле представляют один и тот же файл.

Если вы рассмотрите каталоги в этой файловой системе, то ее содержимое выглядело бы так, как показано на рис. 4.4. Реальная разметка файловой системы, как показано на рис. 4.5, не выглядит настолько ясной, как представление на уровне пользователя.

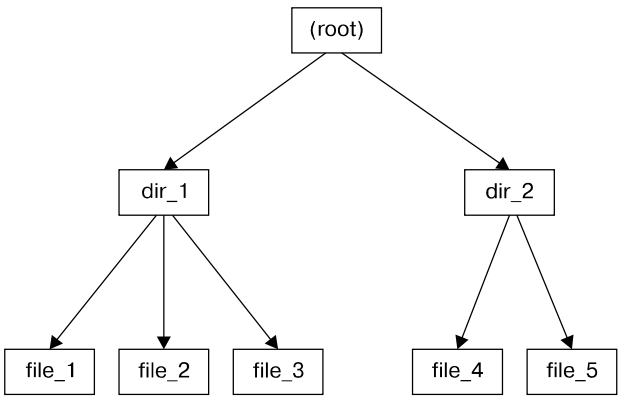

Рис. 4.4. Представление файловой системы на уровне пользователя

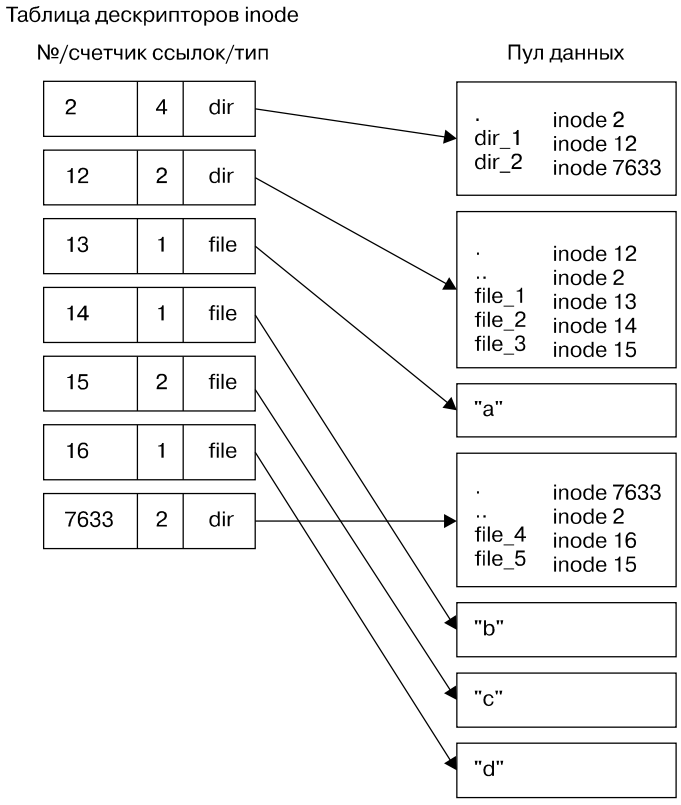

Рис. 4.5. Структура дескрипторов файловой системы, показанной на рис. 4.4

В любой расширенной файловой системе (ext2/3/4) нумерация дескрипторов начинается с 2 — *корневого дескриптора inode*. Из таблицы дескрипторов на рис. 4.5 можно заметить, что это дескриптор каталога (dir), поэтому можно перейти по стрелке к пулу данных, где вы увидите содержимое корневого каталога: два элемента с именами `dir_1` и `dir_2`, которые соответствуют дескрипторам inode 12 и 7633. Чтобы разобраться с этими элементами, вернитесь в таблицу дескрипторов и посмотрите на любой из них.

Для проверки ссылки `dir_1/file_2` в этой файловой системе ядро выполняет следующие действия.

1. Определяет компоненты пути: за каталогом `dir_1` следует компонент с именем `file_2`.
2. Переходит к корневому дескриптору и его данным о каталогах.
3. Отыскивает в данных о каталогах у дескриптора inode 2 имя `dir_1`, которое указывает на дескриптор с номером 12.
4. Ищет дескриптор inode 12 в таблице дескрипторов и проверяет, является ли он дескриптором каталога.

5. Следует по ссылке дескриптора inode 12 к информации о каталоге (это второй сверху контейнер в пуле данных).
6. Обнаруживает второй компонент пути (`file_2`) в данных о каталогах дескриптора inode 12. Эта запись указывает на дескриптор inode с номером 14.
7. Отыскивает дескриптор inode 14 в таблице каталогов. Это дескриптор файла.

К этому моменту ядро знает свойства файла и может открыть его, проследовав по ссылке на данные дескриптора inode 14.

Такая система дескрипторов, указывающих на структуры данных каталогов, и структур данных, указывающих на дескрипторы, позволяет создать иерархию файловой системы, к которой вы привыкли. Кроме того, обратите внимание на то, что дескрипторы каталогов содержат записи для текущего каталога (`.`) и родительского (`..`), исключая лишь корневой каталог. С их помощью можно легко получить точку отсчета при переходе на уровень выше в структуре каталогов.

4.5.1. Просмотр деталей дескрипторов inode

Чтобы увидеть значения дескриптора для любого каталога, используйте команду `ls -i`. Вот что получится, если применить ее к корневому каталогу нашего примера. Для получения более детальной информации воспользуйтесь командой `stat`.

```
$ ls -i
  12 dir_1 7633 dir_2
```

Теперь вас, вероятно, заинтересует счетчик ссылок. Он уже встречался в результатах работы обычной команды `ls -l`. Как счетчик ссылок относится к файлам, показанным на рис. 4.5, и, в частности, к «жестко связанному» файлу `file_5`? Поле со счетчиком ссылок содержит общее количество записей в каталоге (по всем каталогам), которые указывают на дескриптор inode. У большинства файлов счетчик ссылок равен 1, поскольку они появляются лишь один раз среди записей каталога. Этого и следовало ожидать: в большинстве случаев при создании файла вы создаете новую запись в каталоге и соответствующий ей новый дескриптор inode. Однако дескриптор 15 появляется дважды: сначала он создан как дескриптор для файла `dir_1/file_3`, а затем связан с файлом `dir_2/file_5`. Жесткая ссылка является лишь созданной вручную записью в каталоге, связанной с уже существующим дескриптором inode. Команда `ln` (без параметра `-s`) позволяет вручную создавать новые ссылки.

По этой причине удаление файла иногда называется *разъединением*. Если запустить команду `rm dir_1/file_2`, ядро начнет поиск записи с именем `file_2` в каталоге дескриптора inode 12. Обнаружив, что `file_2` соответствует дескриптору inode 14, ядро удаляет эту запись из каталога, а затем вычитает 1 из счетчика ссылок для дескриптора inode 14. В результате этот счетчик ссылок становится равным 0 и ядро будет знать о том, что с данным дескриптором inode не связаны никакие имена. Следовательно, этот дескриптор и все относящиеся к нему данные можно удалить.

Однако, если запустить команду `rm dir_1/file_3`, в результате получится, что счетчик ссылок для дескриптора inode 15 изменится с 2 на 1 (поскольку ссылка `dir_2/file_5` все еще указывает на него) и ядро узнает о том, что данный дескриптор не следует удалять.

Счетчик ссылок для каталогов устроен во многом так же. Заметьте, что счетчик ссылок дескриптора inode 12 равен 2, поскольку присутствуют две ссылки на этот дескриптор: одна для каталога *dir_1* в записях каталога для дескриптора inode 2, а вторая ссылка в собственных записях каталога (`.`) ведет на саму себя. Если создать новый каталог *dir_1/dir_3*, счетчик ссылок для дескриптора inode 12 станет равным 3, поскольку новый катлог будет включать запись о родительском каталоге (`..`), который связан с дескриптором inode 12, подобно тому, как родительская ссылка дескриптора inode 12 указывает на дескриптор inode 2.

Есть одно небольшое исключение. У корневого дескриптора inode 2 счетчик ссылок равен 4. Однако на рис. 4.5 показаны только три ссылки на записи каталогов. «Четвертая» ссылка находится в суперблоке файловой системы, поскольку именно он указывает, где отыскать корневой дескриптор inode.

Не бойтесь экспериментировать со своей системой. Создание структуры каталогов с последующим применением команды `ls -i` или `stat` для исследования различных частей является безопасным. Вам не придется выполнять перезагрузку системы (если только вы не смонтировали новую файловую систему).

Тем не менее один фрагмент еще отсутствует: каким образом файловая система при назначении блоков из пула данных для нового файла узнает о том, какие блоки использованы, а какие свободны? Один из самых простых способов состоит в применении дополнительной структуры управления данными, которая называется *битовой картой блоков*. В этой схеме файловая система резервирует последовательность байтов, в которой каждый бит соответствует одному блоку в пуле данных. Если бит равен 0, это означает, что данный блок свободен, а если 1, то блок используется. Таким образом, в основу распределения блоков положено переключение состояния битов.

Проблемы в файловой системе возникают тогда, когда данные из таблицы дескрипторов inode не соответствуют данным о размещении блоков или когда счетчики ссылок неверные; это может произойти, если работа системы была завершена некорректно. Тогда при проверке файловой системы (как было рассказано в подразделе 4.2.11) команда `fsck` будет просматривать таблицу дескрипторов inode и структуру каталогов, чтобы определить новые счетчики ссылок и создать новую карту размещения блоков (такую как битовая карта блоков), после чего она сравнит только что созданные данные с файловой системой на диске. Если обнаружатся несоответствия, команда `fsck` должна исправить счетчики ссылок и определить, как поступить с теми дескрипторами inode и/или данными, которые не были обнаружены при просмотре структуры каталогов. Большинство команд `fsck` помещает такие «осиротевшие» новые файлы в каталог `lost+found`.

4.5.2. Работа с файловыми системами в пространстве пользователя

При работе с файлами и каталогами в пространстве пользователя вы можете не беспокоиться о том, как это реализовано на нижнем уровне. От вас ожидают, что доступ к содержимому файлов и каталогов смонтированной файловой системы будет осуществляться с помощью системных вызовов ядра. Любопытно, что при

этом у вас все же есть доступ к определенной системной информации, которая не очень вписывается в пространство пользователя, в частности, системный вызов `stat()` возвращает номера дескрипторов inode и счетчики ссылок.

Если вы не заняты обслуживанием файловой системы, следует ли волноваться насчет дескрипторов inode и счетчиков ссылок? Как правило, нет. Эти данные доступны для команд из пространства пользователя в основном в целях обратной совместимости. К тому же не все файловые системы, доступные в Linux, располагают необходимой для них «начинкой». Слой интерфейса VFS обеспечивает то, что системные вызовы всегда возвращают значения дескрипторов inode и счетчиков ссылок, однако такие числа не обязаны что-либо значить.

В нетрадиционных файловых системах вы можете быть лишены возможности выполнять обычные для Unix операции с файловой системой. Например, нельзя использовать команду `ln`, чтобы создать жесткую ссылку в смонтированной файловой системе VFAT, поскольку в ней совершенно другая структура записей о каталогах.

Системные вызовы, которые доступны в пространстве пользователя Unix/Linux, обеспечивают достаточный уровень абстракции для безболезненного доступа к файлам: вам не обязательно знать что-либо о том, как он реализован, чтобы работать с файлами. Более того, гибкий формат имен файлов и поддержка использования разного регистра символов облегчают возможность применения других файловых систем с иерархической структурой.

Помните о том, что ядро не обязано поддерживать специфические файловые системы. В файловых системах с пространством пользователя ядро лишь выступает в роли проводника для системных вызовов.

4.5.3. Эволюция файловых систем

Даже в самой простой файловой системе имеется множество различных компонентов, требующих обслуживания. В то же время предъявляемые к файловым системам требования неуклонно возрастают по мере появления новых задач, технологий и возможностей хранения данных. Нынешний уровень производительности, целостности данных и требований безопасности намного превосходит ранние реализации файловых систем, поскольку технология файловых систем постоянно меняется. Мы уже упоминали в качестве примера о Btrfs — файловой системе нового поколения (см. подраздел 4.2.1).

Одним из примеров того, как изменяются файловые системы, является использование новыми файловыми системами раздельных структур данных для представления каталогов и имен файлов, а не дескрипторов inode, описанных здесь. Они по-другому ссылаются на блоки данных. Кроме того, в процессе развития находятся файловые системы, оптимизированные для дисков SSD. Постоянные изменения в развитии файловых систем являются нормой, однако имейте в виду, что эволюция файловых систем не изменяет их предназначения.