7. ФАЙЛОВАЯ СИСТЕМА

7.1. Типы файлов

Файловая система в Linux многих пугает своей мнимой сложностью. Устройства, ссылки, "иноды", права доступа: всё это кажется очень непонятным. На самом же деле, если все "разложить по полочкам", то становится очевидным, что проще и быть не может. В этой главе мы совершим экскурс в файловую систему Linux и посмотрим на нее с позиции программиста.

Начнем с типов файлов. С точки зрения содержимого или назначения, файлов огромное множество (изображения, музыка, тексты, исполняемые файлы программ и т. п.). Но нас интересует, какие типы файлов присутствуют в Linux с точки зрения ядра. А их всего семь!

Первый тип - обычные файлы (regular files). Эти файлы предназначены для хранения информации на носителях памяти. К обычным файлам относятся упомянутые выше муыкальные и текстовые файлы; сюда же относят исполняемые файлы программ, конфигурационные файлы, файлы журналов и прочие "хранители различной информации".

Обычные файлы, как мы уже знаем, создаются при помощи системных вызовов open() и creat(). С точки зрения пользователя, существует великое множество способов создания обычных файлов: текстовые редакторы, программы работы с графикой и прочие. Чтобы создать пустой файл, можно, например, воспользоваться утилитой touch, как показано ниже.

```
$ touch regularfile
$ ls -l regularfile
-rw-r--r-- 1 sergio sergio 0 2008-10-12 23:38 regularfile
```

Программа ls, вызванная с флагом -l, вывела на экран дополнитеьлную информацию о файле regularfile. Обратите внимание на поле прав доступа: в самом начале стоит минус, который как раз и сообщает нам, что regularfile - это обычный файл.

Чтение обычных файлов и запись информации осуществляются при помощи системных вызовов, описанных в главе 5. Для удаления обычного файла используется системный вызов unlink(). Почему системный вызов называется именно unlink(), а не что-нибудь более благозвучное, наподобие remove() или delete(), мы узнаем в последующих разделах этой главы. Пользователь может удалить обычный файл командой rm.

Второй тип файлов - это каталоги (directories). Каталоги особым образом хранят информацию о других файлах. Если каталог A содержит информацию о файлах B и C, то говрят, что файлы B и C содержатся в каталоге A. Если же, например, C тоже окажется каталогом, то мы скажем, что C - это подкаталог каталога A.

Каталоги создаются системным вызовом mkdir() или одноимённой утилитой.

```
$ mkdir directory
$ ls -ld directory
drwxr-xr-x 2 sergio sergio 4096 2008-10-13 00:28 directory
```

В показанном выше примере мы создаём каталог с именем directory. Ключ -d программы ls позволяет просмотреть информацию о каталоге, а не о содержимом этого каталога. Первым символом в поле прав доступа теперь является d, сообщающий нам, что это каталог.

Для чтения содержимого каталога используются системные вызовы readdir() и getdents().

Однако эти системные вызовы не предназначены для применения в пользовательских приложениях, поэтому программисты пользуются библиотечными оболочками этих вызовов - одноимённой функцией readdir(), а также функцией rewinddir(). Для открытия и закрытия каталогов применяют библиотечные функции opendir() и closedir(). Пользователь может просмотреть содержимое каталога при помощи утилиты ls.

Для удаления каталогов используется системный вызов rmdir(). Пользователь может удалить пустой каталог одноимённой утилитой rmdir.

Третий тип файлов - символические ссылки (symbolic links). Символическая ссылка хранит информацию о местоположении какого-либо файла в файловой системе и, в большинстве случаев, является "полномочным представителем" этого файла.

Символические ссылки создаются при помощи системного вызова symlink(). Пользователь может создать символическую ссылку программой ln с флагом -s.

```
$ echo "Hello" > anyfile
$ ln -s anyfile anysymlink
$ cat anysymlink
Hello
$ ls -l anysymlink
lrwxrwxrwx 1 sergio sergio 7 2008-10-13 01:57 anysymlink -> anyfile
```

Как видно из приведенного выше примера, символическая ссылка может передаваться программам вместо файла, на который она ссылается. Но важно помнить, что программист при желании может добавить в программу код, распознающий символические ссылки и, например, не допускающий их использования. Но подобные задачи ставятся редко.

В выводе программы ls с флагом -l символическая ссылка обозначается в графе прав доступа символом l. Символические ссылки удаляются подобно обычным файлам: посредством системного вызова unlink() или утилитой rm.

Четвертый и пятый типы файлов - это, соответственно, символьные (character devices) и блочные (block devices) устройства. Как правило, символьное устройство оперирует потоками данных, а блочные устройства имеют дело с блоками информации фиксированного размера. Символьные устройства создаются для непосредственного обмена данными с драйвером, находящимся в ядре. Блочные устройства в основном предназначены для того, чтобы создавать на них файловые системы. Подробнее об устройствах речь пойдет в других главах книги.

Сейчас лишь важно понять, что символьные и блочные устройства связаны с драйверами, которые находятся в ядре. Это могут быть драйверы, связанные с реальными аппаратными устройствами (например, принтер, жесткий диск, раздел жесткого диска), так и виртуаьлные драйверы (генератор случайных чисел, генератор нулей /dev/zero, блочная петля (loop device)).

Каждому драйверу в ядре присвоена пара номеров, причем у драйверов символьных устройств своя нумерация, у блочных - своя. В реальности несколькими парами номеров может управлять один драйвер, но сейчас речь не об этом. Первый номер в паре называют старшим (major), а второй - младшим (minor). Некоторые номера жестко прописаны за конкретными устройствами (например, /dev/null всегда имеет старший номер 1 и младший 3), другие же являются "плавающими" и выделяются ядром динамически из свободного списка по мере необходимости.

Зная старший и младший номера интересующего нас устройства, мы можем создать файл этого устройства при помощи утилиты mknod. Не лишним будет заметить, что для одной и той же пары номеров можно создать в файловой системе сколь угодно много устройств.

Символьные и блочные устройства создаются при помощи системного вызова mknod() или при помощи одноименной утилиты. В зависимости от конфигурации системы, у обычного пользователя может не быть прав на использование mknod().

Типичным представителем символьных устройств является /dev/null, виртуальное устройство, которое бесследно "съедает" любую передаваемую ему информацию.

```
$ ls -l /dev/null
crw-rw-rw- 1 root root 1, 3 2008-10-12 18:36 /dev/null
```

Как видим, в выводе ls с флагом -l символьные устройства показаны символом c. В поле, где у обычных файлов пишется размер, у устройств фигурируют их номера, в данном случае 1 и 3. Блочные устройства обозначаются в выводе ls -l символом b.

```
$ ls -l /dev/?da
brw-r----- 1 root disk 8, 0 2008-10-12 18:36 /dev/sda
```

Удаляются файлы устройств при помощи системного вызова unlink() или утилитой rm.

Шестым типом файлов являются именованные каналы FIFO (First In, First Out). Эти файлы используются для локального взаимодействия процессов, когда один процесс записывает в файл информацию, а другой процесс читает. Подробно именованные каналы FIFO будут рассматриваться в последующих главах книги.

FIFO создаются при помощи системного вызова mkfifo() или утилитой mkfifo.

```
$ ls -l myfifo
prw-r--r-- 1 sergio sergio 0 2008-10-13 03:50 myfifo
```

В выводе ls -l FIFO обозначаются символом p. Для удаления именованного канала используется системный вызов unlink() или утилита rm.

И, наконец, седьмой тип файлов - локальные сокеты или, как их еще называют, Unix-сокеты (Unix-sockets). Сокеты - это очень мощный и универсальный способ взаимодействия процессов. Этой теме будет посвящена отдельная глава книги.

В выводе ls -l сокеты обозначаются символом s. Создаются локальные сокеты при помощи системных вызовов socket() и bind(), а удаляются при помощи системного вызова unlink() или утилитой rm.

7.2. Индексные дескрипторы и жесткие ссылки

С понятием файла в Linux связаны три сущности: данные, индексные дескрипторы и жесткие ссылки. На счет данных все понятно: в зависимости от типа файла, ядро предоставляет все необходимые механизмы для чтения, записи и хранения информации, связанной с конкретным файлом.

Индексные дескрипторы (иноды, i-nodes, индексы) - это структура данных, содержащая все метаданные файла, т.е. права доступа, даты создания, последней модификации, размер файла и т.д., а также номер, который уникален в рамках одного диска, раздела или иного носителя информации. Индексные дескрипторы не содержат в себе имя файла. В ОС Unix файлы связываются с упомянутым номером, а не с его именем. Имена файлов содержат каталоги, таким образом содержимое каталогов можно условно представить в виде списка имен файлов

и номеров индексных дескрипторов. Пара состоящая из имени файла и номера индексного дескриптора называется жесткой ссылкой, далее будем говорить просто ссылка. Иными словами, у каждого файла есть свой уникальный номер, через который осуществляется взаимодействие с данными. Чтобы посмотреть этот номер, достаточно вызвать ls с опцией -i (--inode - длинный вариант).

```
$ touch myfile1
$ touch myfile2
$ ls -i
6373383 myfile1  6373384 myfile2
```

Нетрудно убедиться, что у любого файла в файловой системе, независимо от типа, есть индексный дескриптор.

```
$ ls -li /dev/null
252 crw-rw-rw- 1 root root 1, 3 2008-10-21 03:43 /dev/null
$ mkfifo myfifo
$ ls -li myfifo
6373385 prw-r--r-- 1 sergio sergio 0 2008-10-21 04:06 myfifo
$ mkdir mydir
$ ls -lid mydir
5898479 drwxr-xr-x 2 sergio sergio 4096 2008-10-21 04:06 mydir
```

Как видим, у каждого файла есть свой номер (индексный дескриптор). Каталоги - не исключение. Но обратите внимание на число, которое идет после поля прав доступа. В нашем примере у /dev/null и у myfifo оно равно единице, а у каталога mydir - 2. Это число показывает количество ссылок на данный индексный дескриптор в файловой системе. И тут мы подходим к последней сущности, связанной с файлами.

То, что мы привыкли называть файлами - это всего лишь ссылки. Естественно, нам приятнее работать с именем /dev/null, а не с дескриптором номер 252. Но самое интересное в том, что на один и тот же индексный дескриптор в файловой системе может быть сколь угодно много ссылок (в пределах разумного, естественно). Третье поле вывода ls -l как раз и показывает это количество.

При создании обычного файла, устройства, FIFO, символической ссылки или локального сокета в файловой системе появляется индексный дескриптор и одна ссылка на него. Но при создании каталога создаются сразу две ссылки. Рассмотрим пример, объясняющий такое поведение.

```
$ ls -lid mydir
5898479 drwxr-xr-x 2 sergio sergio 4096 2008-10-21 04:06 mydir
$ ls -lia mydir
5898479 drwxr-xr-x 2 sergio sergio 4096 2008-10-21 04:06 .
5898448 drwxr-xr-x 3 sergio sergio 4096 2008-10-21 04:06 ..
```

Как видим, в каталоге mydir находится ссылка на тот же номер (точка), а также ссылка на родительский каталог (две точки). Этим и объясняется наличие двух ссылок на создаваемый каталог.

Итак, мы выяснили, что файловая система в обычном понимании - это набор ссылок, причем одному дескриптору может соответствовать несколько ссылок. Теперь важно понять две вещи.

- Все ссылки на один индексный дескриптор равноправны: нет ни главных, ни второстепенных.
- Файл существует, пока существует хотя бы одна ссылка на индексный дескриптор. Файл считается удалённым, когда удаляется последняя ссылка на него. Ссылки на несуществующие индексные дескрипторы (осиротевшие ссылки), а также индексные дескрипторы, не имеющие хотя бы одной ссылки (осиротевшие иноды) не могут существовать в нормально работающей файловой системе. Указанные случаи "сиротства" свидетельствуют об ошибках файловой системы.

Таким образом, чтобы удалить файл, нужно удалить все ссылки. Этим и объясняется название системного вызова unlink(), о чем говорилось в предыдущем разделе. После удаления последней ссылки, файловая система автоматически вычёркивает файл из своих закромов.

Ссылки можно создавать самостоятельно программой ln. Важно понимать, что символические ссылки, которые мы рассматривали в предыдущем разделе не имеют ничего общего с жесткими ссылками. Символическая ссылка - это рядовой файл со своим индексным дескриптором. Рассмотрим пример.

```
$ cd mydir
$ ls
$ touch file1
$ ln -s file1 mylink1
$ ln file1 mylink2
$ echo "i am here" > file1
$ ls -li
491528 -rw-r--r-- 2 sergio sergio 10 2008-10-21 04:57 file1
491617 lrwxrwxrwx 1 sergio sergio  5 2008-10-21 04:56 mylink1 -> file1
491528 -rw-r--r-- 2 sergio sergio 10 2008-10-21 04:57 mylink2
```

mylink1 - это символическая ссылка; то есть особый тип файла, хранящий путь к другому файлу. Обратите внимание, что mylink1 соответствует отдельный индексный дескриптор с номером 491617. А вот file1 и mylink2 - это абсолютно равноправные ссылки на один и тот же индексный дескриптор с номером 491528. Таким образом, символическая ссылка указывает на имя, а жесткая ссылка - на индексный дескриптор. Если удалить ссылку file1, то символическая ссылка mylink1 "осиротеет". Однако это не является ошибкой файловой системы.

```
$ rm file1
$ cat mylink1
cat: mylink1: No such file or directory
```

Программа rm вызвала системный вызов unlink() и удалила одну из двух ссылок на индексный дескриптор с номером 491528. Однако другая ссылка (mylink2) продолжает жить и здравствовать.

```
$ ls -li
491617 lrwxrwxrwx 1 sergio sergio  5 2008-10-21 04:56 mylink1 -> file1
491528 -rw-r--r-- 1 sergio sergio 10 2008-10-21 04:57 mylink2
$ cat mylink2
i am here
```

В выводе ls мы видим, что на индексный дескриптор 491528 осталась одна ссылка. Удалив

её, мы уничтожим файл окончательно и освободим номер индексного дескриптора.

С точки зрения ядра обычный файл и каталог ничем не отличаются; для их различия вводится спецальный флаг в структуру индексного дескриптора. Но в отличие от обычного файла, жесткую ссылку на каталог создать нельзя. Это запрещено ядром и какими бы вы правами не обладали, обойти этот запрет нельзя. Сделанно это для того, чтобы невозможно было создавать жесткие ссылки на родительские каталоги. В противном случае программы, обходящие дерево файловой системы, зависали бы в бесконечном цикле, поскольку две жесткие ссылки абсолютно равноправны и различить их невозможно. Несмотря на этот запрет, в виртуальной файловой системе все же существуют каталоги, которые содержат ссылку на самих себя и на родительский каталог. Это упомянутые выше директории с именами . (точка) и .. (две точки). Поскольку имя у них одно и то же, то в программе легко можно предусмотреть игнорирование этих каталогов.

И, наконец, возникает два вопроса.

- Можно ли создать символическую ссылку в одной файловой системе (например, в домашнем каталоге) на файл, содержащийся в другой файловой системе (на flash-накопителе, к примеру)?
- Можно ли создать жесткую ссылку в одной файловой системе (пример - домашний каталог) на файл, содержащйся в другой файловой системе (например, на flash-носителе)?

Если вы поняли, о чем говорилось в этом разделе, то без труда ответите на эти вопросы. Пусть это будет вашим "домашним заданием".

7.3. Режим файла

Под режимом файла (file mode) понимается связанный непосредственно с индексным дескриптором (а не со ссылкой) 16-битный набор, регламентирующий порядок доступа и работы с файлом. Иногда режим файла путают с правами доступа, однако это не совсем верно.

Биты режима файла можно разделить на 3 группы:

- биты 0-8: основные права доступа;
- биты 9-11: дополнительные права доступа;
- биты 12-15: тип файла.

Основные права доступа определяют возможность чтения, записи и исполнения для владельца, группы и остальных пользователей. Здесь лишь следует отметить, что бит исполнения для каталога означает возможность "заходить" в него, то есть делать каталог текущим. Право на чтение для каталога означает возможность получения его содержимого, а право на запись позволяет создавать, удалять или переименовывать файлы внутри каталога.

В Unix-системах основные права доступа чаще всего представляются в двух форматах: символический (rwx-формат) и восьмеричный. При вызове ls с опцией -l, права доступа выводятся в символическом формате (например, rw-r--r--). Если же каждый из символов цепочки rwxrwxrwx представить в виде девяти битов, где значение каждого бита определяет наличие (1) или отсутствие (0) соответствующих прав доступа, а затем представить эту цепочку восьмеричным числом, то получим цифровой восьмеричный формат прав доступа.

Если, к примеру, некоторый файл обладает правами на чтение и запись для владельца, только чтения для группы и отсутствием каких-либо прав для остальных пользователей, то в символическом виде это будет представлено цепочкой rw-r-----. В двоичном виде эта цепочка будет представлена последовательностью 110100000, а в восьмеричном виде - 640.

Для перевода двоичного числа в восьмеричное используют следующий простой алгоритм.

- Двоичное число при необходимости спереди дополняют нулями так, чтобы количество цифр без остатка делилось на три. Например, двоичное число 1010 превращают в 001010.
- Полученную цепочку разбивают на триады. Таким образом, число 001010 разбивают на триады 001 и 010.
- Каждую триаду переводят в восьмеричную цифру следующим образом:
  000 - 0
  001 - 1
  010 - 2
  011 - 3
  100 - 4
  101 - 5
  110 - 6
  111 - 7
- Полученные цифры будут образовывать искомое восьмеричное число. Таким образом, двоичному числу 1010 соответствует восьмеричное 12.

Иногда перед восьмеричным числом (например, в языке программирования C) ставят ноль. Это, кроме прочего, позволяет не спутать число с десятеричным.

В случае с правами доступа всё еще проще, поскольку 9 битов уже образуют три триады, и никакого добавления ведущих нулей не требуется. Немного попрактиковавшись, вы поймете, что перевод прав доступа из одного представления в другое - это очень простая задача, которая обычно решается в уме за считанные секунды. Для самопроверки можете использовать утилиту stat следующим образом.

```
$ touch file1
$ ls -l file1
-rw-r--r-- 1 sergio sergio 0 2008-11-09 13:55 file1
$ stat -c %a file1
644
```

Дополнительные права доступа представлены тремя битам - SUID, SGID и sticky-бит. Последний также известен под названием "липкий бит". Бит SUID позволяет запускать исполняемый файл не от имени текущего пользователя, а от имени владельца этого файла. Бит SGID даёт возможность запускать исполняемый файл с правами группы-владельца файла. Здесь есть определенные тонкости, о которых будет рассказано в других главах книги.

Липкий бит в ранних Unix-системах использовался для исполняемых файлов, чтобы заставить ядро не выгружать из памяти код программы после её завершения. Тогда он назывался битом SVTX (SaVe TeXt). В современных системах sticky-бит устанавливается для каталогов. Если каталог имеет право на запись для всех, то каждый может создавать, удалять и переименовывать там файлы. Если же для каталога установлен sticky-бит, то любой пользователь также может удалять и переименовывать файлы, но только свои, а не чужие. Практически в любой Linux-системе для каталога /tmp установлен липкий бит, позволяющий разным пользователям мирно уживаться в едином пространстве для временных файлов.

Что касается типа файла, то об этом уже писалось в разделе 7.1. Вернемся лучше к правам доступа. Возможно вы слышали что-нибудь про маску прав доступа (umask). Это 9 битов, которые вычитаются из битов основных прав доступа при создании файла. Естественно, речь идет о побитовом вычитании.

Если вы используете оболочку bash, то значение umask можно посмотреть, набрав одноимённую команду.

```
$ umask
0022
```

umask - это не отдельная утилита, а встроенная команда оболочки. Обратите внимание, что umask выводит восьмеричную маску именно с ведущим нулём. Значение 022 показывает, что при создании файла биты записи для группы и остальных пользователей будут сбрасываться.

Значение umask, подобно окружению, привязывается к процессу и передаётся по наследству его потомкам. Для переустановки маски создания файлов в оболочке bash используется опция -S команды umask.

```
$ umask -S 0026
u=rwx,g=rx,o=x
$ umask
0026
```

Полученное значение umask будет распространяться только на текущий процесс оболочки, а также на все процессы, порожденные этой оболочкой. Более подробно umask будет рассматриваться в последующих главах книги.

Программа ls, вызванная с опцией -l, умещает в десяти полях все 16 бит режима файла.

- Первое поле показывает тип файла символами из набора -dscbpl. Об этом подробно рассказывалось в разделе 7.1.
- Второе поле символами из набора -r показывает наличие или отсуствие бита чтения для владельца.
- Третье поле символами -w показывает наличие или отсутствие бита записи для владельца.
- Четвертое поле может содержать один из четырех символов набора -xsS. Символ - (минус) означает отсуствие бита выполнения для владельца, x - его наличие. Символ s показывает, что бит выполнения для владельца отсутствует, но стоит бит SUID. И, наконец, символ S означает одновременное наличие бита выполнения и бита SUID.
- Пятое поле содержит один из символов набора -r, показывающий наличие или отсутствие прав на чтение для группы.
- В шестом поле используются символы из набора -w, означающие отсутствие или наличие прав на запись для группы.
- Седьмое поле может быть заполнено одним из символов из набора -xsS. Символ - (минус) говорит об отсутствии прав на выполнение для группы, x - об их наличии. Символ s показывает наличие бита SGID при отсутствии прав на выполнение. Если в седьмом поле стоит S, то это означает одновременное наличие бита SGID и бита прав на выполнение для группы.
- Восьмое поле означает наличие или отсутствие прав на чтение для остальных пользователей. Здесь используются символы из набора -r.
- В девятом поле используются символы из набора -w, обозначающие отсутствие (-) или наличие (w) прав на запись для остальных.
- И в десятом поле могут быть символы из набора -xtT. Символ - (минус) означает отсутсвтие прав на выполнение остальными, а x - наличие таковых. Символ t говорит о наличие sticky-бита при отсутствии прав на выполнение для остальных. Если в десятом поле стоит символ T, то этим сообщается об одновременном наличии липкого бита и бита выполнения для остальных.