Файловая система в Linux

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение, 2019

Содержание лекции

Аксиоматика файловой системы в Linux

Типичная модель файловой системы Linux:

- **нижний уровень**: драйверы устройств и файловых систем, полностью реализован в ядре Linux.
- **средний уровень**: системные вызовы, работающие с файловыми системами и осуществляющие низкоуровневый ввод-вывод, унифицированный интерфейс файловой системы, семь типов файлов:
  1. каталоги,
  2. символьные устройства,
  3. блочные устройства,
  4. обычные файлы,
  5. каналы FIFO,
  6. символические ссылки,
  7. сокеты.
- верхний уровень: библиотеки и приложения, которые создают полноценную иллюзию единого дерева файловой системы.

Типы файлов. Обычный файл

Создадим пустой файл при помощи программы *touch*:

```
$ touch anyfile
```

Теперь наберем следующую команду:

```
$ ls -l anyfile

-rw-r--r-- 1 nnivanov nnivanov 0 2011-05-06 20:11 anyfile
```

Прочерк в самом начале - поле типа файла с точки зрения среднего уровня реализации файловой системы. *Если здесь стоит минус, то это обычный файл.*

Типы файлов. Каталог

Давайте теперь создадим каталог:

```
$ mkdir anydir
$ ls -l

drwxr-xr-x 2 nnivanov nnivanov 4096 2011-05-06 20:13 anydir/
-rw-r--r-- 1 nnivanov nnivanov 0 2011-05-06 20:11 anyfile
```

В поле типа файла для *anydir* находится символ *d*, показывающий, что это каталог (*directory*).

Типы файлов. Символические ссылки

Символические ссылки - указатели на другие файлы.

Создадим символическую ссылку:

```
$ ln -s anyfile anylink
$ ls -l

drwxr-xr-x 2 nnivanov nnivanov 4096 2011-05-06 20:13 anydir/
-rw-r--r-- 1 nnivanov nnivanov 0 2011-05-06 20:11 anyfile
lrwxrwxrwx 1 nnivanov nnivanov 7 2011-05-06 20:14 anylink -> an
```

Символические ссылки в выводе *ls* обозначаются символом *l* (*link*).

Типы файлов. Устройства

Устройства в Linux могут быть представлены в виде файлов. На среднем уровне реализации файловой системы под устройствами понимают файлы двух типов:

- символьные устройства, обозначаемые в выводе программы *ls* символом *c* (character);
- блочные устройства, обозначаемые символом *b* (*block*).

Введите следующую команду:

```
$ ls -l /dev/null

crw-rw-rw- 1 root root 1, 3 2011-05-06 20:16 /dev/null
```

Вывод программы ls показывает, что /dev/null является символьным устройством.

Типы файлов. Устройства

Унификация обозначения устройств, содержащих файловые системы, жесткие диски, USBнакопители, CD/DVD-приводы, SCSI-устройства:

Список блочных устройств:

```
$ ls -l /dev/sd*

brw-rw---- 1 root disk 8, 0 2011-05-06 18:41 /dev/sda
brw-rw---- 1 root disk 8, 1 2011-05-06 18:41 /dev/sda1
brw-rw---- 1 root disk 8, 2 2011-05-06 18:41 /dev/sda2
brw-rw---- 1 root disk 8, 16 2011-05-06 18:53 /dev/sdb
brw-rw---- 1 root disk 8, 17 2011-05-06 18:53 /dev/sdb1
```

Типы файлов. Каналы. Сокеты

Для взаимодействия процессов существуют особые файлы, которые называются каналами FIFO (First In First Out) или просто FIFO.
Эти файлы создаются командой mkfifo и обозначаются в выводе программы ls символом p (pipe):

Создание канала:

```
$ mkfifo anyfifo
$ ls -l anyfifo

prw-r--r-- 1 nnivanov nnivanov 0 2011-05-06 20:33 anyfifo
```

Сокеты динамически создаются программами при помощи системного вызова *socket()*.
Сокеты обозначаются в выводе команды *ls* символом *s* (socket).

Права доступа

- Когда процесс пытается получить доступ к какому-нибудь файлу, ядро Linux использует UID текущего процесса для проверки прав доступа.
- Если проверка не удалась, в ход идет GID (Group ID, идентификатор группы).
- Если и здесь доступ закрыт, проверяются права доступа для остальных пользователей.

Права доступа файла shadow

```
$ ls -l shadow

-r--r----- 1 root shadow 1012 2011-03-08 10:03 /etc/shadow
```

Данные о пользователях хранятся в файле /etc/passwd, который открыт для всех, а зашифрованные пароли находятся в файле /etc/shadow, доступ к которому разрешен только суперпользователю.

Права доступа

Программу su, которая позволяет временно входить в систему на правах другого пользователя.
Как su "умудряется"сравнивать введенные вами пароли с содержимым закрытого от посторонних глаз файла /etc/shadow?

В первом окне введите команду su:

```
$ su
Password:
```

Перейдем в другое окно и введем:

```
$ ps -ef
...
nnivanov 17952 9299 0 20:37 pts/1 00:00:00 bash
root 17989 9302 0 20:37 pts/0 00:00:00 su
nnivanov 18008 17952 0 20:37 pts/1 00:00:00 ps -efu
```

Права доступа

Введите теперь следующую команду:

```
$ ls -l /bin/su

-rwsr-xr-x 1 root root 34500 2010-04-29 19:17 /bin/su
```

- Триада, показывающая базовые права доступа для пользователя, содержит символ s на месте бита исполнения.
- Этот символ означает, что для исполняемого файла установлен бит SUID (Set User IDentifier), который позволяет запускать данную программу от лица владельца ее исполняемого файла.
- Существует бит SGID (Set Group IDentifier), который позволяет запускать программу от имени группы, которой принадлежит этот файл.

К процессу привязано два идентификатора пользователя:

- реальный UID (Real UID, RUID или просто UID);
- эффективный UID (Effective UID или просто EUID).

В большинстве случаев эти идентификаторы равны. Но если для исполняемого файла программы установлен бит SUID, то ситуация меняется.

Пример

когда пользователь anyuser запускает программу su, то реальный UID процесса равен идентификатору пользователя anyuser, а эффективный UID равен идентификатору пользователя root.

Установить бит SUID для исполняемого файла:

```
$ chmod u+s file
```

Установить бит SGID для исполняемого файла:

```
$ chmod g+s file
```

Расширенные права доступа позволяют задействовать еще один бит, который называется битом SVTX или липким битом (sticky). В каталоге с установленным битом SVTX каждый может создавать, удалять и переименовывать файлы, не мешая другим пользователям делать то же самое.

Каталог /tmp, доступен всем для хранения временных файлов:

```
$ ls -l / | grep tmp

drwxrwxrwt 63 root root 12288 2011-05-06 20:42 tmp/
```

Назначить каталогу directory липкий бит:

```
$ chmod +t directory
```

Служебные файловые системы

Файловая система /dev

- хранилище файлов символьных и блочных устройств.

Файловая система /proc

- была создана для предоставления пользователю информации о работающих процессах;
- затем туда стали помещать туда «полезную» информацию (версия ОС, объем оперативной памяти, сообщения ядра и т. п.);
- в некоторые файлы дерева /proc можно писать, обеспечивая тем самым динамическое конфигурирование ядра.

Файловая система /tmp

- предназначена для хранения временных файлов.
- может располагаться как на диске, так и в памяти.

Устройства

На среднем уровне реализации файловой системы все устройства представляются двумя типами файлов:

Символьные устройства (character devices)

- особенность - механизм последовательного ввода-вывода;
- рассматриваются не в виде линейного массива данных, а как поток информации.

Блочные устройства (block devices)

- читают и записывают данные блоками фиксированного размера с использованием механизма буферизации.
- файлы устройств этого типа чаще всего применяются для дисковых накопителей и для дисковых разделов.
- некоторые сетевые устройства также могут быть представлены файлами этого типа.

Файлы устройств обычно создаются в Linux программой mknod

```
$ mknod [-m MODE] NAME TYPE MAJOR MINOR
```

- -m принимает независимый аргумент MODE, который является правами доступа создаваемого файла в восьмеричном представлении.
- NAME - имя файла устройства.
- TYPE может быть представлен одним из следующих символов: c, b, u или p.
  - c - будет создано символьное устройство.
  - b - инициирует создание блочного устройства.
  - u - блочное устройство, но с выключенным механизмом буферизации (unbuffered).
  - p - создание именованного канала FIFO.
- MAJOR — это старший номер устройства (номер драйвера).
- MINOR — младший номер устройства (форма применения драйвера).

см. Documentation/devices.txt

Монтирование файловых систем

позволяет объединять различные файловые системы в одно дерево.

```
$ mount
/dev/sda2 on / type ext4 (rw)
none on /proc type proc (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/dev/sdb1 on /media/4779-86EA type vfat
(rw,nosuid,nodev,uhelper=udisks,uid=500,gid=500,
shortname=mixed,dmask=0077, utf8=1,flush)
```

DEVICE on MOUNTPOINT type FSTYPE (FLAGS)

- DEVICE — имя устройства.
- MOUNTPOINT — точка монтирования: каталог, являющийся как бы «порталом» для смонтированной файловой системы.
- FSTYPE — тип файловой системы с точки зрения нижнего уровня ее реализации.
- FLAGS — опции монтирования.

Семейство statvfs()

Информация о смонтированных файловых системах обычно хранится в файле/etc/mtab:

```
$ cat /etc/mtab
/dev/sda2 / ext4 rw 0 0
none /proc proc rw 0 0
none /proc/sys/fs/binfmt_misc binfmt_misc rw 0 0
```

Каждая строка /etc/mtab соответствует одной смонтированной файловой системе и содержит шесть полей:

- Первое поле - носитель файловой системы.
- Второе поле - точка монтирования.
- Третье поле - тип файловой системы.
- Четвертое поле - флаги монтирования.
- Пятое поле используется утилитой суперпользователя dump.
- Шестое поле необходимо для утилиты fsck.

Функции для разбора файла /etc/mtab:

```
FILE * setmntent (const char * FILENAME, const char * MODE);
struct mntent * getmntent (FILE * FILEP);
int endmntent (FILE * FILEP);
```

- setmntent() открывает файл, в котором содержится список смонтированных файловых систем;
- getmntent() извлекает следующую запись о файловой системе.
- endmntent() вызывается по окончании чтения записей о файловых системах.

Поля структуры mntent соответствуют полям /etc/mtab:

```
char *mnt_fsname;
char *mnt_dir;
char *mnt_type;
char *mnt_opts;
int mnt_freq;
int mnt_passno;
```

Более подробную информацию о файловых системах позволяют получить следующие функции, объявленные в заголовочном файле sys/statvfs.h:

```
int statvfs (const char * PATH, struct statvfs * FS);
int fstatvfs (int FD, struct statvfs * FS);
```

- *f_bsize* - целое число, содержащее размер блока файловой системы в байтах.
- *f_blocks* - целое число, показывающее количество блоков в файловой системе.
- *f_bfree* - целое число, содержащее количество свободных блоков в файловой системе.
- *f_bavail* содержит число доступных блоков, без учета резервных блоков, которые некоторые файловые системы выделяют для специального использования.
- *f_namemax* - целое число, показывающее максимально возможную длину имени файла в данной файловой системе.

Примеры

Пример 1. Работа функции statvfs()

ex09_01.c

Пример 2. Работа функции fstatvfs()

ex09_02.c

Текущий каталог: getcwd()

Каждый дочерний процесс наследует от своего родителя независимую копию текущего каталога.
Для получения значения текущего каталога используется системный вызов getcwd(), объявленный в заголовочном файле unistd.h следующим образом:

```
char * getcwd (char * BUFFER, size_t SIZE);
```

Чтобы получить только базовое имя каталога, можно воспользоваться функцией basename(), объявленной в заголовочном файле string.h следующим образом:

```
char * basename (const char * PATH);
```

Примеры. Получение текущего каталога

ex09_03.c, ex09_04.c

Смена текущего каталога: chdir()

Любой процесс может сменить свой текущий каталог. Это можно сделать при помощи одного из приведенных далее системных вызовов, объявленных в заголовочном файле unistd.h:

```
int chdir (const char * PATH);
int fchdir (int FD);
```

- Обе функции возвращают 0 при успешном завершении и –1, если произошла ошибка.
- Системный вызов chdir() изменяет текущий каталог, используя его имя (путь), а fchdir() вместо имени читает файловый дескриптор каталога.

Примеры. Смена текущего каталога

ex09_05.c, ex09_06.c

Открытие и закрытие каталога

Пользовательские библиотеки располагаются на верхнем уровне реализации файловой системы, каталог здесь рассматривается как самостоятельная сущность, не являющаяся особым файловым типом.

- В стандартной библиотеке языка C абстракцией каталога является указатель типа DIR.
- Прежде чем читать содержимое каталога, его нужно открыть, а прочитанный каталог полагается закрывать.

```
DIR * opendir (const char * NAME);
int closedir (DIR * DIRP);
DIR * fdopendir (int FD);
```

- Первая функция открывает каталог с именем NAME и возвращает указатель типа DIR. В случае ошибки возвращается NULL.
- Функция closedir() закрывает каталог и возвращает 0 при успешном завершении, в случае ошибки –1

Чтение каталога: readdir()

Для чтения содержимого каталога применяется функция readdir(), объявленная в заголовочном файле dirent.h следующим образом:

```
struct dirent * readdir (DIR * DIRP);
```

Структура dirent содержит несколько полей, но нам понадобится только одно из них:

```
struct dirent
  /* ... */
  char * d_name;
;
```

Функция readdir() заносит в поле d_name структуры dirent имя очередного элемента просматриваемого каталога. Если каталог полностью просмотрен, то readdir() возвращает NULL.

Примеры. Просмотр каталога

ex09_07.c

Получение данных о файлах: семейство stat()

Дополнительную информацию о файлах позволяют получить системные вызовы семейства stat(), объявленные в заголовочном файле sys/stat.h:

```
int stat (const char * FNAME, struct stat * STATISTICS);
int fstat (int FD, struct stat * STATISTICS);
int lstat (const char * FNAME, struct stat * STATISTICS);
```

- Системный вызов stat() читает информацию о файле с именем FNAME и записывает эту информацию в структуру stat по адресу STATISTICS.
- Вызов fstat() также читает информацию о файле, который представлен дескриптором FD.
- lstat() работает аналогично stat(), но при обнаружении символической ссылки lstat() читает информацию о ней, а не о файле, на который эта ссылка указывает.

Структура stat также объявлена в файле sys/stat.h. Для нас наибольший интерес представляют следующие поля этой структуры:

```
struct stat {
  mode_t st_mode;
  uid_t st_uid;
  gid_t st_gid;
  off_t st_size;
  time_t st_atime;
  time_t st_mtime;};
```

- st_mode - это режим файла.
- st_uid - это числовой идентификатор владельца файла;
- st_gid - идентификатор группы;
- st_size - это размер файла в байтах;
- st_atime - содержит дату и время последнего обращения к файлу;
- st_mtime - содержит дату и время последней модификации файла.

Примеры

Пример 8. Системный вызов stat()

ex09_08.c

Пример 9. Системный вызов fstat()

ex09_09.c

Пример 10. Системный вызов lstat()

ex09_10.c

Биты режима файла

Биты режима файла делятся на три группы:

- базовые права доступа;
- расширенные права доступа;
- тип файла.

Проверка базовых прав доступа: сравнить режим файла (st_mode структуры stat) с одной из констант из sys/types.h (S_IRUSR, S_IWUSR и т. п.) с использованием операции побитовой конъюнкции (И).

```
mode & S_IWUSR //владельцу разрешено писать в файл
```

Проверка расширенных прав доступа:

- S_ISUID - установка и проверка бита SUID;
- константа S_ISGID установка и проверка бита SGID;
- константа S_ISVTX установка и проверка sticky-бита.

Биты режима файла

Для определения типа файла применяются следующие макросы::

- S_ISDIR(mode) - каталог;
- S_ISCHR(mode) - символьное устройство;
- S_ISBLK(mode) - блочное устройство;
- S_ISREG(mode) - обычный файл;
- S_ISFIFO(mode) - именованный канал FIFO;
- S_ISLNK(mode) - символическая ссылка;
- S_ISSOCK(mode) - сокет.

Пример. Чтение содержимого каталога

ex09_11.c

Чтение ссылок

Программа ls, вызванная с флагом –l, позволяет узнать, на какой файл указывает символическая ссылка:

```
$ touch myfile
$ ln -s myfile mylink
$ ls -l mylink
lrwxrwxrwx 1 nnivanov nnivanov 6 2011-05-07
    09:59 mylink -> myfile
```

Такая операция называется разыменованием символической ссылки (symlink dereferencing) или разрешением имени файла (filename resolution).

Чтение ссылок

Разыменование символических ссылок осуществляется при помощи системного вызова readlink(), который объявлен в заголовочном файле unistd.h следующим образом:

```
ssize_t readlink (const char * SYMLNK, char * BUF,
    size_t SIZE);
```

- системный вызов помещает в буфер BUF размера SIZE путь к файлу, на который указывает символическая ссылка SYMLNK.
- Возвращаемое значение — число байтов, записанных в буфер BUF. В случае ошибки возвращается –1.
- readlink() не завершает буфер нультерминатором.

Пример. Чтение ссылок

ex09_12.c