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 («everything is file»):
  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

Обычный файл (file)

обозначенная некоторым именем последовательность данных, которые хранятся на диске (устройстве хранения данных).

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

Каталог (директория, directory)

формально тоже считается файлом, содержащим данные о файлах, хранящихся в нем.

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

Символьная или мягкая ссылка (символическая ссылка, symbolic link, simlink)

указатель на другой файл, предназначена для исключения дублирования и более оптимального использования места на диске.

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
                                                  -> anyfile
```

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

Типы файлов. Устройства

Устройства в *Linux* могут быть представлены в виде файлов. Файлы устройств отображаются в каталоге /**dev** дерева каталогов *Linux*.
На среднем уровне реализации файловой системы под устройствами понимают файлы двух типов:

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

Файл устройства не обязательно соответствует какому-либо реальному устройству. Он может соответствовать и виртуальному устройству, реализуемому программно (/dev/random).

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.

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.

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

Для взаимодействия процессов (выполняемых на одной машине) 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
```

Типы файлов. Сокеты

Сокет (socket) похож на канал в том смысле, что тоже является механизмом для обмена данными между двумя разными процессами. Однако сокет имеет два важных отличия от канала:

- процессы чаще всего запущены на разных компьютерах;
- обмен данными осуществляется по сети с использованием стека протоколов *TCP/IP*.

Сокеты обозначаются в выводе команды *ls* символом *s* (socket).

Сокеты динамически создаются программами при помощи системного вызова *socket()*.

Права доступа

В Linux каждый файл имеет своего **владельца** (и только одного) - им обычно является пользователь, создавший этот файл.

Владелец файла определяет **права доступа** к файлу. Для каждого файла могут быть заданы:

- право на чтение;
- право на запись;
- право на выполнение.

Указанные права задаются отдельно для трех категорий пользователей:

- для владельца файла;
- для пользователей, входящих в так называемую группу владельца (ограниченный круг доверенных пользователей владельца, которых он сам определяет, включая в группу пользователей, имя которой совпадает с именем владельца);
- для всех остальных пользователей.

Права доступа

Владельца файла можно посмотреть в расширенном листинге команды **ls**:

```
$ ls -l

-rw-r--r-- 2 user user 132 Map 23 14:39 file1.txt
-rw-r--r-- 2 user user 132 Map 23 14:39 file_hard
lrwxrwxrwx 1 user user 9 Map 23 14:40 file_soft -> file1.txt
```

Права доступа

Основные права доступа часто задают в виде трех восьмеричных цифр, например **764**.

- первая цифра (7) определяет права владельца файла,
- вторая (6) – права группы владельца,
- третья (4) – права остальных пользователей.

Каждая восьмеричная цифра прав складывается из трех двоичных разрядов:

- права на исполнение (в разряде единиц),
- права на запись (в разряде двоек)
- права на чтение (в разряде четверок).

Для каталогов права доступа носят особый смысл:

- право на чтение позволяет просматривать содержимое каталога,
- право на запись позволяет удалять или перемещать файлы в каталоге,
- право на выполнение вместе с правом на чтение – осуществлять поиск в каталоге.

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

Программ **su** 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 могут только владелец файла и суперпользователь.

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:

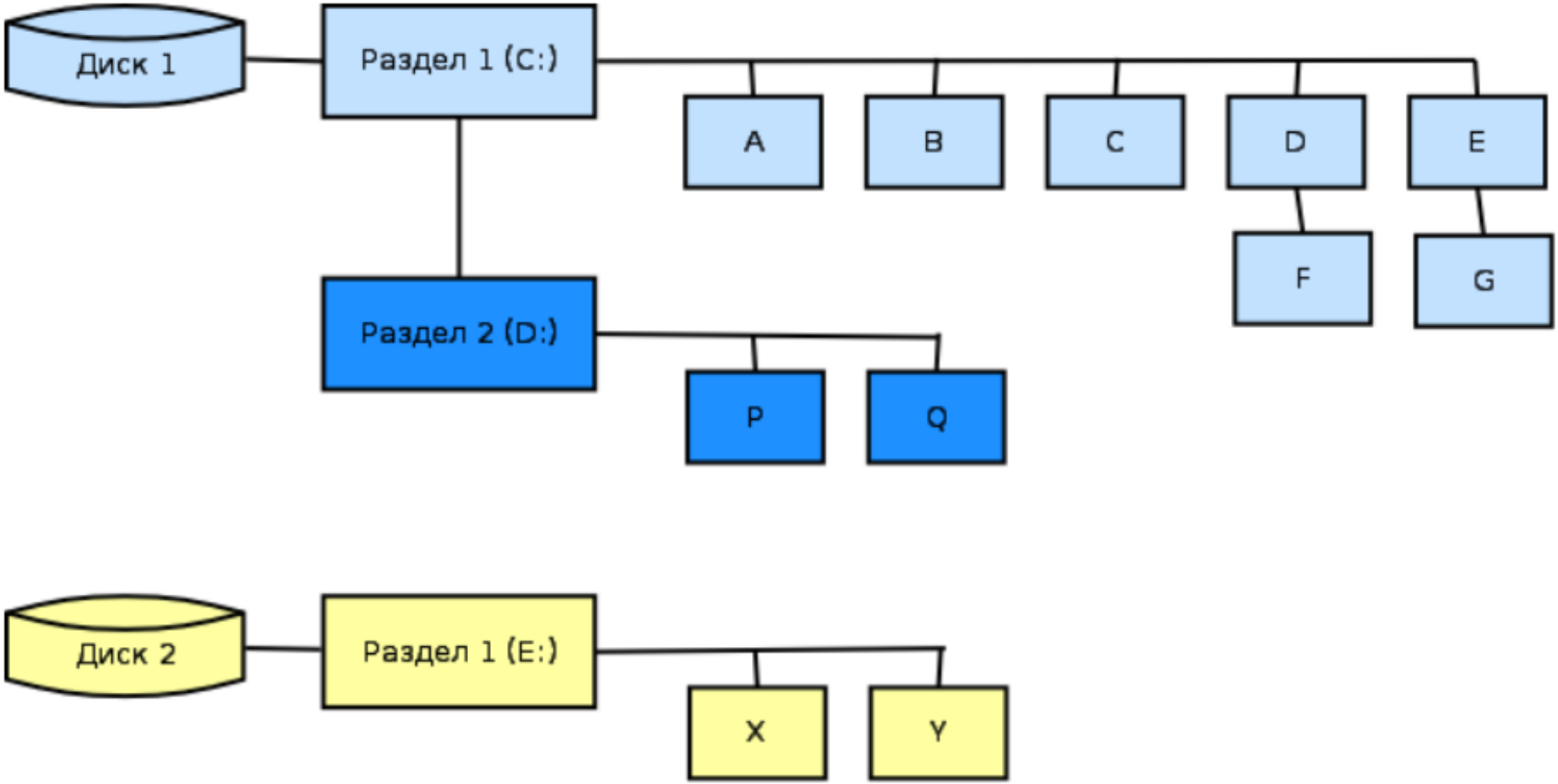
```
$ chmod +t directory
```

Дерево каталогов Linux

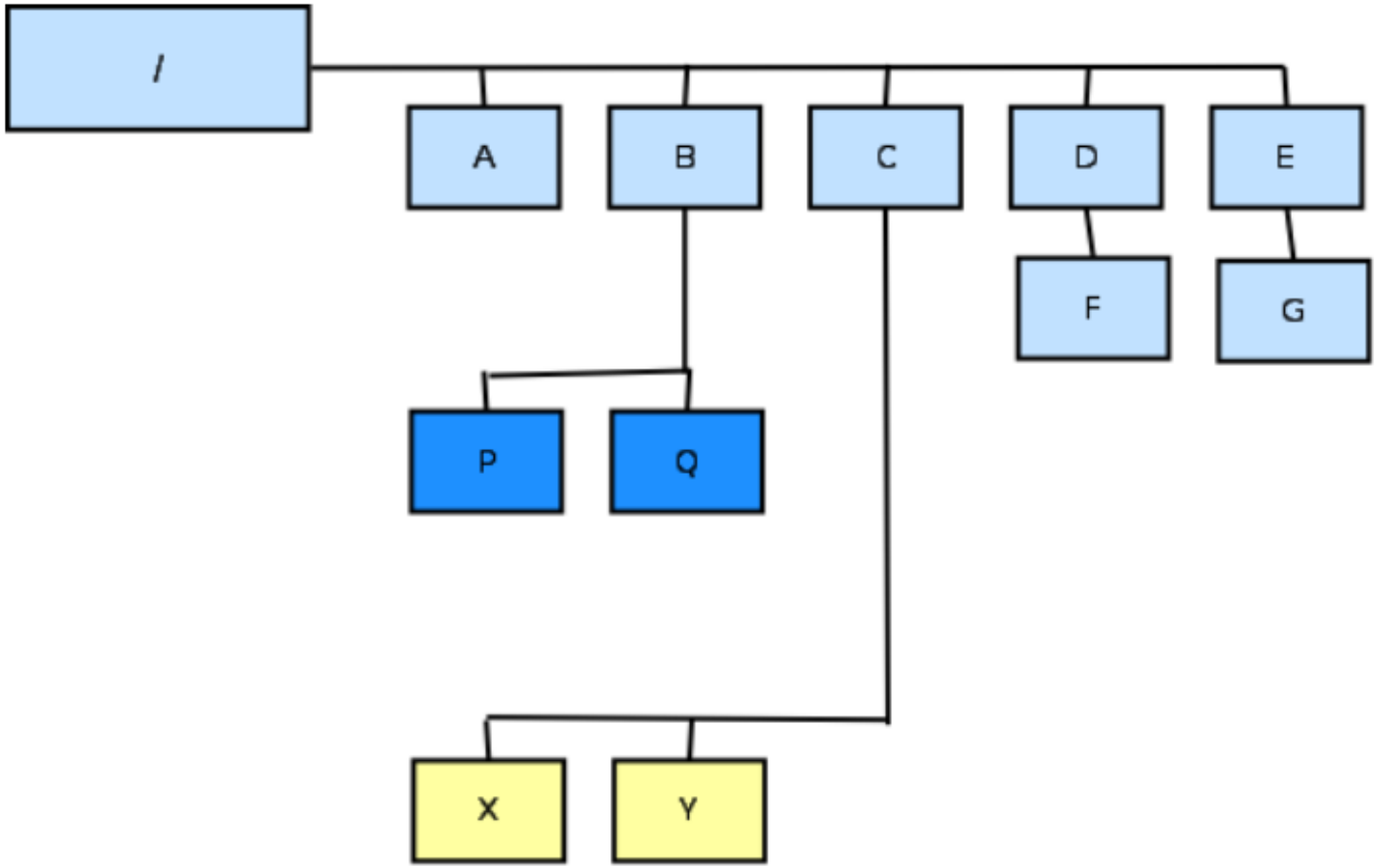
В Linux имеется единое дерево каталогов, вид которого определяется следующим образом:

- основу дерева каталогов составляет корневое устройство;
- все остальные устройства отображаются в одном из каталогов корневой файловой системы;
- операция, заключающаяся во включении файловой системы, находящейся на устройстве, в общее дерево каталогов называется **монтированием**;
- каталог, в котором будет отображаться содержимое монтируемого устройства, называется **точкой монтирования**;
- после завершения работы с некоторыми устройствами, например с флеш-дисками нужно выполнить операцию, противоположную монтированию – **демонтирование**.

Пример структуры каталогов в Windows



Пример структуры каталогов в Linux



Дерево каталогов Linux

Родительским для всех остальных каталогов является так называемый корневой каталог, обозначаемый «/».

- /**bin** – основные системные утилиты, необходимые в однопользовательском режиме.
- /**boot** – файлы загрузчика, например, образ ядра Linux.
- /**dev** – файлы устройств.
- /**etc** – общесистемные конфигурационные файлы и конфигурационные файлы программ.
- /**home** – домашние каталоги пользователей.
- /**lib** – файлы общесистемных разделяемых библиотек и модулей ядра.
- /**media** – точки монтирования для сменных носителей, таких как CD и DVD-диски, flash-накопители и т.п.

Дерево каталогов Linux

- /**mnt** – точки монтирования для временных разделов.
- /**opt** – дополнительные пакеты приложений.
- /**proc** – виртуальная файловую систему proc, которая отображает в виде структуры файлов и каталогов различную информацию о работе системы.
- /**root** – домашний каталог пользователя root.
- /**sbin** – основные системные программы для администрирования и настройки системы.
- /**sys** – точка монтирования еще одной виртуальной файловой системы sys, являющейся расширением proc.
- /**usr** – содержит большинство пользовательских приложений и утилит, использующихся в многопользовательском режиме.
- /**var** – содержит изменяемые файлы, такие как файлы регистрации, временные файлы и т.п.

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.

Монтирование файловых систем может выполняться:

- вручную администратором системы с помощью команды mount
- автоматически при загрузке системы на основе файла настроек /etc/fstab

```
$ 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.
- FSTYPE — тип файловой системы.
- FLAGS — опции монтирования.

Содержание лекции

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

Содержание лекции

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{
  /* режим файла */
  mode_t st_mode;
  /* числовой идентификатор владельца файла */
  uid_t st_uid;
  /* идентификатор группы */
  gid_t st_gid;
  /* размер файла в байтах */
  off_t st_size;
  /* дата и время последнего обращения к файлу */
  time_t st_atime;
  /* дата и время последней модификации файла */
  time_t st_mtime;
};
```

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

Удаление файла: unlink()

Для удаления файла служит системный вызов **unlink**(), объявленный в заголовочном файле *unistd.h* следующим образом:

```
int unlink (const char * FNAME);
```

- Аргумент **FNAME** — это имя удаляемого файла.
- *unlink*() возвращает **0** при успешном завершении.
- В случае ошибки возвращается **−1**.

Удаление файла: unlink()

```c
#include <stdio.h>
#include <unistd.h>

int main (int argc, char ** argv)
{
  if (argc < 2) return 1;

  if (unlink (argv[1]) == -1) {
          fprintf (stderr, "Cannot unlink file (\%s)",
            argv[1]);
          return 1;
  }

  return 0;
}
```

Файл

комплексное понятие, состоящее из следующих компонентов:

- данные (data);
- индексы (inodes);
- ссылки (links).

Индексы

специальные ячейки памяти, зарезервированные файловой системой для разделения данных на файлы.

- Каждый индекс имеет уникальный (в рамках данной файловой системы) номер.
- Индексы содержат информацию о том, в каких блоках файловой системы хранятся данные конкретного файла.
- В индексах содержатся сведения о дате и времени открытия и модификации файла.
- Ссылка - имя индексного узла файловой системы.

Индексы

Программа **ls**, вызванная с флагом **-i**, позволяет увидеть номер индексного узла, на который указывает ссылка:

```
$ mkdir idemo
$ cd idemo
$ touch file1
$ touch file2
$ ls -i

952139 file1
952141 file2
```

- *file1* — это ссылка на индекс с номером 952139
- *file2* указывает на другой индексный узел с номером 952141

Индексы

Продолжим:

```
$ ln -s file1 symlnkf1
$ ls -i

952139 file1
952141 file2
952190 symlnkf1
```

- символическая ссылка является также ссылкой на индекс с номером 952190
- символические ссылки указывают не на индекс, а на имя файла

Индексы

Теперь воспользуемся командой **ln** без флага **-s**:

```
$ ln file1 hardfile1
$ ls -i

952139 file1 952141 file2 952139 hardfile1 952190 symlnkf1
```

- *hardfile1* является жесткой ссылкой на файл *file1*. Вывод команды *ls* показывает, что *file1* и *hardfile1* указывают на один и тот же индекс с номером 952139.
- жесткие ссылки (в отличие от символических) не носят подчиненный характер. *file1* и *hardfile1* — это полноценные ссылки на один и тот же индексный узел.

Индексы

Если две ссылки указывают на один и тот же индекс, то можно сказать, что они имеют доступ к одним и тем же данным:

```
$ echo hello > file1
$ cat hardfile1
hello
```

Индексы являются промежуточным звеном, связывающим ссылку с данными на блочном устройстве.

Команда rm

Программа **rm** работает следующим образом:

- если удаляемый файл является последней ссылкой на соответствующий индексный узел в файловой системе, то данные и индекс освобождаются;
- если в файловой системе еще остались ссылки на соответствующий индексный узел, то удаляется только ссылка.

```
$ rm file1
$ cat hardfile1
hello
```

Теперь обратите внимание на вывод программы **ls** с флагом **-l**:

```
$ ls -l
-rw-r--r-- 1 kt kt 0 2011-05-07 10:00 file2
-rw-r--r-- 1 kt kt 6 2011-05-07 10:00 hardfile1
lrwxrwxrwx 1 kt kt 5 2011-05-07 10:00 symlnkf1 -> file1
```

Команда rm

Символическая ссылка *symlnkf1* по-прежнему указывает на файл *file1*, которого уже не существует:

```
$ cat symlnkf1
cat: symlnkf1: No such file or directory
```

Числа во втором столбце вывода программы **ls** - счетчики ссылок на соответствующие индексные узлы.

```
$ ls -l
-rw-r--r-- 2 kt kt 0 2011-05-07 10:00 file2
-rw-r--r-- 1 kt kt 6 2011-05-07 10:00 hardfile1
-rw-r--r-- 2 kt kt 0 2011-05-07 10:00 hardfile2
lrwxrwxrwx 1 kt kt 5 2011-05-07 10:00 symlnkf1 -> file1
```

Команда df

Если вызвать команду **df** с флагом **-i**, то на экран будет выведена информация по индексным узлам смонтированных файловых систем:

```
$ df -i

Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda6 1311552 264492 1047060 21% /
udev 96028 487 95541 1% /dev
/dev/sda1 66264 58 66206 1% /boot
/dev/sda7 3407872 149600 3258272 5% /home
```

В файловых системах может присутствовать ограниченное число индексов (столбец **Inodes**). Столбец **IUsed** показывает число используемых индексов, а в столбце **IFree** содержится число свободных индексных узлов файловой системы

Команда df

Каждый раз при создании файла в файловой системе выделяется индекс:

```
$ df -i .
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda7 3407872 149586 3258286 5% /home

$ touch file3
$ df -i .

Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda7 3407872 149587 3258285 5% /home
```

Команда df

Создание жесткой ссылки не приводит к появлению в файловой системе нового индекса:

```
$ df -i .
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda7 3407872 149587 3258285 5% /home

$ ln file3 hardfile3
$ df -i .
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/sda7 3407872 149587 3258285 5% /home
```

Пример statvfsinode.c

В рассматриваемом ранее примере программа использовала функцию **statvfs**() для вывода информации о смонтированных файловых системах.
Структура *statvfs* содержит также следующие поля, которые мы не рассматривали:

- f_files — общее число индексов для данной файловой системы;
- f_free — число свободных индексов файловой системы;
- f_favail — число доступных индексов в файловой системе.

Пример statinode.c

Структура **stat** содержит еще одно поле st_ino, в котором находится номер индексного узла, на который ссылается файл.

```
struct stat st;
if (stat(argv[1], &st) == -1) {
    fprintf (stderr, "stat() error\n");
    return 1;
}
printf ("FILE:\t\t%s\n", argv[1]);
printf ("UID:\t\t%d\n", (int) st.st_uid);
printf ("GID:\t\t%d\n", (int) st.st_gid);
printf ("SIZE:\t\t%ld\n", (long int) st.st_size);
printf ("AT:\t\t%s", ctime (&st.st_atime));
printf ("MT:\t\t%s", ctime (&st.st_mtime));
printf ("INODE:\t\t%ld\n", (long int) st.st_ino);
```

Системный вызов unlink() (unlink2.c) удаляет ссылку на индексный узел. Если эта ссылка была последней, то индекс освобождается.

```
$ touch file1
$ ln file1 file2
$ ls -i file1 file2
1048740 file1 1048740 file2
$ ./unlink2 file1
$ ls file1
/bin/ls: file1: No such file or directory
$ cat file2
Hello World
```

Над открытым файлом можно успешно осуществлять операции ввода-вывода, даже если последняя ссылка на этот файл удалена:

```
$ ./unlink2 file2
$ ls file2
/bin/ls: file2: No such file or directory
```

Перемещение файлов

Системный вызов rename() позволяет переименовывать или перемещать файл в пределах одной файловой системы.

```
int rename (const char * OLDF, const char * NEWF);
```

- При успешном завершении rename() возвращает 0.
- В случае ошибки возвращается –1

Пример (rename1.c):

```
if (rename (argv[1], argv[2]) == -1) {
    fprintf (stderr, "rename() error\n");
    return 1;
}
```

Перемещение файлов: rename2.c

Теперь проведем небольшой эксперимент:

```
$ touch file1
$ cat file1
$ ./rename2 file1 file2
$ ls file1
/bin/ls: file1: No such file or directory
$ cat file2
Hello World
```

Перемещение открытого файла никак не отражается на операциях ввода-вывода.

Создание ссылок

Ссылки в файловой системе Linux бывают двух типов:

- символические ссылки (*symbolic links*);
- жесткие (прямые) ссылки (*hard links*).

В распоряжении программиста имеются следующие системные вызовы:

```
int link (const char * FROM, const char * TO);
int symlink (const char * FROM, const char * TO);
```

Оба вызова возвращают **0** при успешном завершении и **−1**, если произошла ошибка. Примеры:

- linkdemo.c
- symlinkdemo.c

Создание каталога

Для создания каталога используется системный вызов **mkdir**():

```
int mkdir (const char * NAME, mode_t MODE)
```

Системный вызов *mkdir*() создает каталог с именем **NAME** и режимом **MODE**. При успешном завершении *mkdir*() возвращает **0**. В случае ошибки возвращается **−1**. Примеры:

- mkdir1.c
- mkdir2.c

Создание каталога

Программа mkdir работает не так, как мы ожидали:

```
$ ./mkdir2 mydir
$ ls -l | grep mydir
drwxr-xr-x 2 kt kt 4096 2011-05-07 10:09 mydir
```

В системный вызов *mkdir*() передавался аргумент **mode**, в котором все биты базовых прав доступа установлены в единицу. Но вывод программы ls показывает, что созданный каталог имеет права доступа **0755**.

К каждому процессу в *Linux* привязана маска прав доступа, которая наследуется потомком от родительского процесса (аналогично текущему каталогу, окружению и т. п.).

Маска прав доступа

число, представляющее собой набор битов прав доступа, которые никогда не будут устанавливаться для создаваемых процессом файлов или каталогов.

Команда **umask** позволяет узнать текущую маску прав доступа командной оболочки:

```
$ umask
0022
```

Маска прав доступа **0022** разрешает при создании файлов или каталогов устанавливать любые права доступа для владельца, но не разрешает права на запись для группы и остальных пользователей.

Маски прав доступа

Текущий процесс вправе изменять свою копию маски прав доступа:

```
$ umask 0044
$ umask
0044
```

Дочерние процессы наследуют копию маски прав доступа родительского процесса:

```
$ umask 000
$ umask
0000
$ bash
$ umask
0000
$ exit
exit
```

Создание каталога

Попробуем запустить программу *mkdir2* с измененной маской прав доступа оболочки:

```
$ umask 0000
$ umask
0000
$ ./mkdir2 mydir
$ ls -l | grep mydir
drwxrwxrwx 2 kt kt 4096 2011-05-07 10:15 mydir
```

Системный вызов umask()

Программа может изменить маску прав доступа текущего процесса при помощи системного вызова **umask**():

```
mode_t umask (mode_t MASK);
```

Этот системный вызов изменяет текущую маску прав доступа и возвращает предыдущее значение маски. Примеры:

- mkdir3.c

```
$ umask 0022
$ umask
0022
$ ./mkdir3 mydir
$ ls -l | grep mydir
drwxrwxrwx 2 kt kt 4096 2011-05-07 10:17 mydir
```

Создание каталога

Пример создания каталога с «липким битом»:

- mkdir4.c

```
$ ./mkdir4 mydir
$ ls -l | grep mydir
drwxrwxrwt 2 kt kt 4096 2011-05-07 10:20 mydir
```

Удаление каталога

Для удаления каталога служит системный вызов **rmdir**():

```
int rmdir (const char * DIR)
```

- Аргумент **DIR** — это имя (путь) к каталогу, который следует удалить.
- При успешном завершении *rmdir*() возвращает **0**. В случае ошибки возвращается **−1**.

Пример:

- rmdirdemo.c

Системный вызов *rmdir*() удаляет только пустые каталоги. Если каталог не пуст, то *rmdir*() завершится неудачей.