Лабораторная работа по курсу "Операционные системы"

Операции с файлами и каталогами в ОС Linux

Цель работы: Знакомство с организацией файловой системы ОС Linux.

1. Файловая система и специальные файлы

Файловая система ОС Linux имеет иерархическую структуру, образующую единое дерево каталогов и файлов. Каждый раздел жесткого диска или отдельный съемный носитель информации (дискета, CD-ROM и т.п.) содержит свою собственную файловую систему. Эти файловые системы организованы независимо и имеют свой собственный корневой каталог. Отдельные файловые системы можно монтировать (присоединять к единому дереву каталогов) или размонтировать (отсоединять от него).

В каждой файловой системе каждому файлу соответствует свой ***индексный дескриптор файла*** (*inode*), который содержит атрибуты файла и информацию о расположении файла на носителе. Файлы не имеют имени. Все индексные дескрипторы в каждой файловой системе пронумерованы последовательно и различаются своими номерами.

Собственно имена файлов хранятся не в индексных дескрипторах файлов, а в каталогах. ***Каталог*** - это файл, который содержит таблицу, каждый элемент которой содержит имя файла и ссылку на индексный дескриптор файла (номер *inode*). Таким образом, файл существует отдельно от каталога и нет взаимно-однозначного соответствия между именем файла и самим файлом. На один файл могут ссылаться несколько элементов как одного, так и разных каталогов. Т.е., к одному и тому же файлу можно обращаться под разными именами.

В *inode* содержится следующая информация о файле (атрибуты файла): тип файла; идентификатор владельца; идентификатор группы; разрешения на доступ к файлу; счетчик числа ссылок на файл; даты создания, модификации и др.

Счетчик числа ссылок на файл - это количество имен (элементов каталога), которые ссылаются на данный файл. При удалении файла удаляется только элемент каталога с именем **name**, при этом в *inode* счетчик ссылок уменьшается. ОС физически удаляет файл с диска при удалении последней ссылки на данный файл.

Создание ссылки на файл выполняется командой **ln**. Пример использования команды **ln**:

ln old new - создать новое имя **new** для файла **old**;

Такая ссылка называется "***жесткая ссылка***" (*hard link*). После ее создания невозможно определить, какой из элементов каталога являлся первоначальным именем файла, а какой - ссылкой, созданной позже. При создании ссылки на файл необходимо, чтобы и ссылка и сам файл располагались в одной файловой системе (на одном физическом носителе информации). В программе жесткие ссылки создаются системным вызовом **link** и удаляются системным вызовом **unlink**.

Существует и другой тип ссылки - ***символьная ссылка*** (*symbolic link*). Это просто ссылка на другой элемент каталога (похожа на ярлык в Windows). *inode* символьной ссылки вместо информации о расположении файла на диске содержит путь к тому элементу каталога, на который указывает эта ссылка. При этом большинство операций (чтение, запись), выполняемых над символьной ссылкой, выполняются над элементом каталога, на который указывает ссылка. Но некоторые операции (например, удаление) выполняются непосредственно над самой символьной ссылкой. Символьная ссылка и элемент каталога, на который она ссылается, могут располагаться в разных файловых системах.

Создать символьную ссылку можно командой **ln** с флагом **s**.

Задание 1.1. Создайте текстовый файл **a** и введите в него какие-нибудь данные. Создайте жесткую ссылку **h** и символьную ссылку **s** на файл **a**. Измените разрешения доступа к файлу **h** и его содержимое. Изменилось ли содержимое и разрешения доступа для файла **a**? *Получите и расшифруйте в отчете информацию о файлах **a**, **h** и **s** (включая и номера inode) при помощи команды **ls** (выясните флаг команды ls для вывода номеров inode).* Просмотрите содержимое файла **s**. Затем удалите файл **a**. Просмотрите содержимое файлов **h** и **s**. Получите снова информацию о файлах **h** и

s при помощи команды **ls**. *В отчет занесите введенные Вами команды, полученную информацию и Ваши выводы.*

Специальные файлы. Ядро ОС UNIX идентифицирует устройства по типу (блочное или символьное), а также по паре номеров - старший (идентифицирует драйвер устройства) и младший номера устройства (идентифицирует определенный экземпляр устройства в группе одинаковых устройств, управляемых одним драйвером). По старшему номеру устройства ОС определяет необходимый драйвер. По младшему номеру устройства драйвер определяет порт контроллера.

Для пользователя предоставляется более простой способ именования устройств. Для этого с каждым устройством ОС Linux связывает ***специальный файл,*** и доступ к устройству производится путем обращения к этому файлу. Над специальным файлом можно выполнять те же операции, что и над любым другим файлом: открыть, читать, писать, и т.д. Однако результат применения этих операций зависит от того, какому конкретному физическому устройству соответствует данный специальный файл. Например, специальный файл **dev/lp0** соответствует первому параллельному порту (принтеру). Открытие и вывод в этот файл некоторой информации приведет к выводу ее на принтер:

```
int fd = open ("/dev/lp0", O_WRONLY);
write (fd, &buffer, buffer_length);
close (fd);
```

Попытки прочитать информацию из этого файла лишены смысла.

Обычно доступ к специальным файлам разрешен лишь системным программам и привилегированному пользователю. Для хранения специальных файлов по соглашению используется каталог **/dev**. При просмотре содержимого каталога командой **ls** блочные специальные файлы помечаются символом '**b**', символьные специальные файлы - символом '**c**'. Вместо размера файла выводятся старший и младший номера устройств.

Монтирование файловой системы. Поскольку в полном имени файла отсутствует имя устройства, и все имена определяются относительно одного корневого каталога, то для того, чтобы обратиться, например, к файлу на гибком диске, необходимо подключить файловую систему, находящуюся на гибком диске, к корневой файловой системе. Для этого служит команда и системный вызов **mount**, который монтирует файловую систему, находящуюся на устройстве, соответствующем специальному файлу **special_file**, в какой-либо существующий каталог **directory** корневой файловой системы. При этом каталог **directory** будет называться ***точкой монтирования*** файловой системы. Простейший вариант команды **mount**:

mount special_file directory

Например, специальный файл **/dev/fd0** соответствует флоппи-дисководу. После выполнения команды **mount /dev/fd0 / mnt/floppy**
к файлу **student/ivanov**, находящемуся на дискете, теперь можно обратиться по имени **/mnt/floppy/student/ivanov**. Размонтирование файловой системы, смонтированной командой **mount**, выполняется при помощи команды **umount**:

umount special_file или **umount directory**

Специальные устройства. Существует несколько символьных устройств, не соответствующих каким-либо аппаратным устройствам. Специальные файлы этих устройств используют в качестве старшего номера устройства число **1**. Этот номер соответствует не драйверу устройства, а драйверу памяти ядра ОС.

- **/dev/null** - ***null*** устройство. а) Linux игнорирует данные, направляемые в этот файл. Поэтому для того, чтобы подавить вывод какой-либо команды, следует перенаправить ее **stdout** в этот файл: **$ some_command > /dev/null**

б) Операция чтения из этого файла всегда возвращает 0 байтов (конец файла). Пример создания файла **file1** нулевой длины: **$ cp /dev/null > file1**

- **/dev/zero** - ведет себя как бесконечно длинный файл, содержащий двоичные нули.
- **/dev/full** - ведет себя как файл в файловой системе, которая исчерпала всю память, т.е. попытка записать данные в этот файл вызовет ошибку **ENOSPC**.

- **/dev/random** и **/dev/urandom** - обеспечивают доступ к генератору случайных чисел Linux. В отличие от датчика псевдослучайных чисел из стандартной библиотеки C, эти случайные числа невоспроизводимы при повторном запуске программы.

Задание 1.2. При помощи команды **ls** *выпишите и расшифруйте информацию* (тип устройства, старший и младший номера устройств, владельца и разрешения на доступ к файлу) для следующих специальных файлов:

/dev/fd0	Первый флоппи-дисковод
/dev/fd1	Второй флоппи-дисковод
/dev/ttyS0	Первый последовательный порт
/dev/console	Системная консоль
/dev/tty1	Первый виртуальный терминал
/dev/tty	Текущее устройство-терминал для процесса
/dev/audio	Звуковая карта
/dev/null	Устройство ***null***

Задание 1.3.
Введенная без параметров, команда **mount** показывает список смонтированных файловых систем. Выпишите его в отчет с Вашими пояснениями.

Задание 1.4.
Выполните программу **cat**, задавая в качестве имен входных/выходных файлов имена специальных устройств, перечисленные выше. *Выпишите в отчет Ваш результат.*

Псевдотерминалы. Специальные файлы **/dev/tty1**, **/dev/tty2** и т.п. служат для доступа к ***виртуальным терминалам*** (**TTY**). Переключиться на первый виртуальный терминал можно клавишами **Ctrl+Alt+F1**, на второй - **Ctrl+Alt+F2** и т.д. Виртуальный терминал номер 7 обычно используется для графического интерфейса пользователя (X Window System).

Специальный файл **/dev/tty** соответствует ***терминальному устройству*** процесса. Если ввод-вывод информации осуществлять при помощи этого файла, то пользователь по-прежнему будет видеть данные на своем экране, но при этом он уже не сможет перенаправлять ввод-вывод при помощи символов ">" и "<". Поэтому ввод из файла **/dev/tty** часто используется при запросе пароля у пользователя.

При работе в графической среде ОС Linux создает для каждого терминального окна свой ***псевдотерминал*** (**PTS**). Файловая система псевдотерминалов, **devpts**, смонтирована в каталог **/dev/pts**, и каждому терминалу в ней соответствует свой специальный файл с именем, соответствующим номеру псевдотерминала. В отличие от обычных файловых систем, файловой системе **devpts** не соответствует какое-либо физическое устройство (см. листинг команды **mount**). В отличие от остальных элементов каталога **/dev**, которые расположены на диске, **devpts** создается динамически ядром ОС в оперативной памяти, содержимое этой файловой системы меняется в процессе запуска и завершения процессов. Увидеть соответствие псевдотерминалов и процессов можно в листинге команды **ps**.

Задание 1.5. а) При помощи команды **ps** определите, сколько псевдотерминалов открыто. Затем сравните Ваши результаты с содержимым каталога псевдотерминалов **/dev/pts**. Совпадают ли они? б) Перейдите на первый виртуальный терминал (**Ctrl+Alt+F1**) и войдите в систему. Затем вернитесь в графическую среду, на седьмой виртуальный терминал (**Alt+F7**). Запустите две оболочки *bash*. Определите имя специального файла для псевдотерминала первой оболочки (**ls -l /dev/pts**). Из окна второй оболочки направьте какое-либо сообщение в файл псевдотерминала первой оболочки (Воспользуйтесь командой **echo** и перенаправлением стандартного вывода). Затем направьте сообщение в файлы первого и второго виртуального терминала. *Поясните в отчете Ваш результат, сравнив разрешения обоих файлов. Выпишите в отчет введенные Вами команды и полученные результаты.*

Файловая система /proc является виртуальной и в действительности она не существует на диске. Ядро создает ее в памяти компьютера. Система **/proc** предоставляет информацию о системе. Если вы выведете список содержимого каталога **/proc**, вы увидите каталоги, именами которых

являются **pid** процессов. Эти каталоги содержат информацию о всех процессах в системе, запущенных в данный момент. Каждый из каталогов содержит одинаковые пункты. Подробную информацию о структуре и содержании файловой системы **/proc** можно найти в **man**-руководстве к **proc** (а также в файле **proc.pdf** из каталога данной лабораторной работы).

Задание 1.6. Зайдите в каталог, имя которого совпадает с **pid** одного из процессов Вашего сеанса. *Выпишите в отчет информацию, полученную Вами о процессе из данного каталога.*

2. Печать содержимого каталогов

Задание 2.1. Разработать программу **dirpr.c**, которая получает в командной строке один параметр - имя файла или каталога. Программа печатает информацию об этом файле/каталоге. Если это - каталог, то программа распечатывает его содержимое.

Для разработки данной программы необходимо изучить содержимое структур **stat** (содержит информацию о файле) и **dirent** (содержит информацию об элементе каталога).

Определение структуры **stat** (файлы **sys/stat.h** и **sys/types.h**):

```
struct stat {
      dev_t       st_dev;           /* устройство                          */
      ino_t       st_ino;           /* inode                               */
      umode_t     st_mode;          /* защита                              */
      nlink_t     st_nlink;         /* число ссылок на файл                */
      uid_t       st_uid;           /* ID владельца                        */
      gid_t       st_gid;           /* ID группы владельца                 */
      dev_t       st_rdev;          /* тип устройства (для устройств)      */
      off_t       st_size;          /* общий размер, байтов                */
      unsigned long st_blksize;     /* размер блока для файловой системы   */
      unsigned long st_blocks;      /* число выделенных блоков             */
      time_t      st_atime;         /* время последнего доступа            */
      time_t      st_mtime;         /* время последн.модиф.содержимого файла  */
      time_t      st_ctime;         /* время последн.изменения статусной инф. */
    };
```

Одно из полей структуры **stat** - поле **st_mode**. Оно содержит тип файла и режимы доступа к файлу. Эта информация может быть получена посредством побитовой операции **И** над содержимым **st_mode** и одной из следующих констант (Полный список констант см. в **man**.):

```
   S_IFDIR        Каталог
   S_IFREG        Обычный файл
   S_IREAD        Разрешение на чтение (для владельца)
   S_IWRITE       Разрешение на запись (для владельца)
   S_IEXEC        Разрешение на выполнение (для владельца)
```

Определение структуры **dirent** (файл **dirent.h**):

```
struct dirent {
          long           d_ino;
          off_t          d_off;
          unsigned short d_reclen;
          char    d_name[NAME_MAX + 1]; /* NAME_MAX определено в limits.h */
          };
```

Ниже приведена схема программы **dirpr.c**.

```
   #include  <stdio.h>
   #include  <sys/types.h>
   #include  <sys/stat.h>
   #include  <sys/dir.h>
   main (argc, argv)
   int    ... ;
   char  ...;
   {
   DIR  *dp;                          /* The type DIR is defined in dirent.h    */
   struct   dirent   *dir;            /* The type dirent is defined in dirent.h */
   struct   stat    info;             /* The type stat is defined in sys/stat.h */
   char  pathname[BUFSIZ+1];          /* For a pathname of a file or directory  */
```

Проверьте число параметров командной строки. Если число неверное, распечатайте сообщение

об ошибке и завершите процесс (**exit** с ненулевым кодом возврата)

Сделайте системный вызов **stat()**, первый параметр - имя, полученное из командной строки, второй параметр - адрес структуры **info** типа **stat**. Если код возврата этого системного вызова - ошибка, то распечатайте сообщение об ошибке и завершите процесс. В любом случае распечатайте здесь соответствующее сообщение.

Определите, является ли полученное имя каталогом или обычным файлом (поле **st_mode** структуры **info**). Для этого следует проверить значение **info.st_mode&S_IFREG** (для обычного файла) и **info.st_mode&S_IFDIR** (для каталога). Если ни одно из этих значений не является истиной, завершите процесс.

Если полученное имя - имя обычного файла, то распечатайте имя файла и его размер. Для этого используйте поле **st_size** структуры **info**. Затем завершите процесс.

Если полученное имя - имя каталога, то распечатайте это имя и размер каталога (используйте поле **st_mode** структуры **info**). Затем попытайтесь открыть этот каталог при помощи системного вызова **dp = opendir ()**, указывая полученное имя в качестве аргумента.

Если результат **opendir()** ошибочный (NULL) то распечатайте сообщение об ошибке (при помощи **perror**()) и завершите процесс.

Считывайте очередной элемент каталога в цикле при помощи библиотечного вызова **dir = readdir(dp)** до тех пор, пока **dir** не NULL. После каждого считывания игнорируйте результат, если **dir->d_ino** равно нулю (Это соответствует ранее удаленному файлу). В противном случае прочитайте имя файла **dir->d_name** и сконструируйте полное имя для очередного элемента каталога посредством конкатенации имени из командной строки (оно будет являться именем пути) и имени очередного элемента данного каталога (используйте, например, оператор **sprintf(pathname, "%s/%s", argv[1], dir->d_name**);). Переменную **pathname** следует затем использовать в качестве первого аргумента системного вызова **stat()** (второй аргумент - адрес **info),** для получения текущего состояния файла или каталога. В каждой итерации цикла печатайте соответствующее сообщение для случая, когда вызов **stat()** вернет ошибку. Если **stat()** завершится успешно, то выясните, является ли очередной элемент каталога обычным файлом (проверьте значение **info.st_mode&S_IFREG**). Если это обычный файл, то распечатайте его имя (**dir->d_name**), размер (**info.st_size**), даты последнего доступа, модификации, изменения (**info.st_atime**, **info.st_mtime**, **info.st_ctime**), ID владельца и группы (**info.st_uid**, **info.st_gid**). Если это каталог, то распечатайте его имя и размер.

Закройте каталог при помощи системного вызова **closedir(dp)** и завершите процесс.

Задание 2.2. Модифицируйте программу **dirpr.c**: вместо ID пользователя и ID группы выводите имя пользователя и имя группы. Воспользуйтесь функциями **getpwuid** и **getgrid**.

Порядок выполнения лабораторной работы

Выполните задания и занесите в отчет описание заданий со всеми требуемыми пояснениями.

Требования

1. При подготовке к работе (дома) запишите в конспект программу **dirpr.c** с комментариями.
2. Студент должен знать ответы на следующие вопросы:

Вопросы

1. В чем отличие команды **ln** от команды **cp**? В чем отличие жесткой ссылки от символьной?
2. Можно ли создать жесткую и символьную ссылки на файл, расположенный на другом ПК?
3. Что такое специальный файл? Что такое символьные и блочные специальные файлы?
4. Каково назначение системного вызова **stat()**? Какие у него аргументы?
5. Можно ли использовать **stat()** для файла, который не открыт в текущем процессе?
6. Как определить, является ли элемент каталога обычным файлом или каталогом?
7. Как можно открыть каталог? Как можно прочитать очередной элемент каталога?
8. Как можно узнать размер файла? Как можно узнать имя владельца файла и имя группы?
