Лабораторная работа 1. Файлы и каталоги.

Цель работы: обучение взаимодействию с файловой системой через команды оболочки и системные вызовы ОС.

1. Теоретическая часть.

В UNIX-подобных системах понятие файла является центральным, поскольку в виде объектов файловой системы представляются не только обычные данные, но и многие объекты ОС: процессы и потоки, оперативная память, устройства, примитивы синхронизации и т.д («**В UNIX все есть файл**»).

1.1 Определения.

Файл — именованный набор данных.

Файловая система (ФС) — система, обеспечивающая взаимодействие с файлами. В частности, ФС определяет порядок сопоставления имени и набора данных.

Индексный узел (i-node) — структура, хранящая метаданные файла (тип, атрибуты, расположение блоков данных на диске и т.д.). Индексный узел не хранит имя файла.

Каталог — файл специального вида, хранящий информацию о связанных с ним (вложенных) файлах. Любой каталог как минимум содержит подкаталоги «.»(ссылка на самого себя) и «..» (ссылка на родительский каталог).

Корневой каталог – каталог, являющийся корнем дерева файловой системы UNIX. Имя корневого каталога – «/».

Абсолютный путь – путь, начинающийся с корневого каталога (первый символ равен /).

Рабочий каталог процесса – каталог, от которого отсчитываются относительные пути.

Относительный путь – путь, не начинающийся с корневого каталога (первый символ не равен /). Эквивалентный абсолютный путь равен <рабочий каталог>/<относительный путь>.

Жесткая ссылка — запись в каталоге. Жесткая ссылка сопоставляет имя файла в каталоге и его индексный узел. Несколько записей из разных каталогов могут указывать на один и тот же узел. Индексный узел не удаляется, пока есть хотя бы одна ассоциированная с ним жесткая ссылка.

Символьная ссылка — специальный файл, хранящий имя (путь) файла-назначения и позволяющий взаимодействовать с ним. При удалении файла-назначения жесткая ссылка становится висячей (dangling).

Файлы устройств — файлы, представляющие (псевдо)устройства в файловой системе. Обычно располагаются в каталоге /dev.

Символьное устройство – устройство, ориентированное на последовательное чтение/запись без возможности произвольного доступа. Примером файла символьного устройства является терминал (файл /dev/tty).

Блочное устройство — устройство, ориентированное на произвольный доступ к данным. Примером блочного устройства является жесткий диск (файл /dev/sda).

Дескриптор файла – целое неотрицательное число, являющееся индексом в таблице открытых файлов процесса.

1.2 Полезные системные вызовы

1.2.1 Манипуляции с рабочим каталогом

```
char* getcwd(char* buf, size_t size); // получить рабочий каталог
```

Вызов `getcwd` принимает на вход буфер и размер буфера, в который записывается строка, содержащая абсолютный путь к рабочему каталогу (включая оконечный нулевой символ). В случае успеха возвращается `buf`. Если размер буфера недостаточен, возвращается `NULL`, и `errno` выставляется в `ERANGE`.

В POSIX явно не указывается, что произойдет, если `buf == NULL`. Разные ОС определяют собственное поведение (см. *`man getcwd`*).

```
int chdir(const char* path); // изменить рабочий каталог
```

Вызов `chdir` возвращает 0 в случае успеха и -1 в случае ошибки. В случае ошибки причина ошибки указана в `errno` (см. *`man chdir`*).

1.2.2 Создание, открытие и закрытие файлов

```
//открыть файл
int fd = open(const char *pathname, int flags, mode_t mode);
```

Данный системный вызов открывает файл с именем `pathname` и возвращает дескриптор файла. Дополнительные опции передаются через `flags` (отдельные опции объединяются посредством побитового ИЛИ). Среди полезных опций:

`O_CREAT` – создать файл, если не существует.

`O_TRUNC` – если файл существует и открывается для записи, стереть старые данные.

`O_APPEND` – всегда записывать данные в конец файла.

`O_EXCL` – используется совместно с `O_CREAT`. В случае, если файл существует – завершить вызов с ошибкой. Иными словами, комбинация `O_CREAT|O_EXCL` позволяет убедиться, что был создан новый файл, а не открыт уже существующий.

O_DIRECTORY – вернуть ошибку, если открываемый файл не является каталогом. Обычно используется, если полученный дескриптор планируется передать последующему вызову opendir или fopenat/fchdir/fpathconf (см. *man* для указанный функций).

Параметр mode содержит флаги доступа, которые устанавливаются при создании нового файла (параметр игнорируется при открытии уже существующего файла). Для выполнения лабораторной достаточно указать в качестве значения параметра константу S_IRWXU (права на полный доступ к файлу для создателя файла).

```
int close(int fd); //закрыть открытый файл
```

Хотя желательно закрывать файлы сразу после того, как они станут не нужны, строго говоря, это не обязательно. Все открытые файлы автоматически закроются при завершении процесса.

1.2.3 Чтение и запись

```
/* прочитать данные из файла */
ssize_t read(int fd, void* buf, size_t count);

/* записать данные в файл */
ssize_t write(int fd, const void* buf, size_t count);
```

Первым параметром fd обоих вызовов является дескриптор открытого файла. Параметр buf должен содержать указатель на буфер данных. В параметре count передается число байт, которые надо прочитать из буфера/записать в буфер. Очевидно, что размер буфера должен быть больше count.

Возвращаемым значением обоих вызовов является число фактически считанных/записанных байт. Это значение может быть меньше count. Например, в случае чтения данное значение будет меньше, если достигнут конец файла (0 – если конец файла уже был достигнут и новых данных больше нет). В случае записи данное значение будет меньше, если диск заполнен и удалось записать только часть данных. В случае, если ошибка произошла до того, как какие-либо данные были прочитаны/записаны, возвращенное значение будет равно -1 с кодом ошибки в errno.

```
/* изменить позицию чтения записи */
off_t lseek(int fd, off_t offset, int whence);
```

Вызов изменяет текущую позицию в файле. Следующая операция чтения/записи будет начинаться по новой позиции.

Тип off_t соответствует одному из знаковых целочисленных типов. Аргумент fd соответствует дескриптору файла. Аргумент offset содержит значение, на которое изменяется позиция в файле. Во whence передается одна из именованных констант SEEK_SET, SEEK_CUR, SEEK_END. Если pos – позиция в файле, file_size – размер файла, то, в зависимости от значения whence, новая позиция вычисляется следующим образом:

```
if      (whence == SEEK_SET) pos  = offset; /*новое значение равно заданному*/
     else if(whence == SEEK_CUR) pos += offset; /*позиция смещается на offset байт от текущей*/
     else if(whence == SEEK_END) pos  = file_size + offset; /*позиция смещается на offset байт от конца файла*/
```

В случае успеха возвращается новое значение позиции. В случае ошибки возвращается -1 с кодом ошибки в `errno`. Ошибка возможна, например, при работе с файлом символьного устройства, которые не поддерживают чтение/запись по произвольной позиции.

1.2.4. Изменение размера файла

```
int ftruncate(int fd, off_t length); //изменить размер файла
```

В аргументе `fd` должен передаваться дескриптор файла, открытого на запись. В случае, если значение `length` меньше текущего размера данных, данные в конце файла теряются. В случае, если значение `length` больше, чем текущий размер файла, его размер увеличивается до length (с дополнением нулями).

В случае успеха возвращается 0. В случае ошибки возвращается -1 с кодом ошибки в `errno`.

1.2.5. Создание ссылок

```
/*создать жесткую ссылку*/
int link(const char *oldpath, const char *newpath);
```

Путь `oldpath`, должен указывать на существующий файл. Все каталоги в пути `newpath` должны существовать, но сам целевой файл существовать не должен.

Вызов возвращает 0 в случае успеха и -1 в случае ошибки.

Жесткие ссылки на каталоги может создавать только суперпользователь, однако создание таких ссылок может приводить к зацикливанию и повреждению ФС.

```
/*создать символьную ссылку*/
int symlink(const char *target, const char *linkpath);
```

Вызов создает символьную ссылку на `target` по пути `linkpath`, при этом `target` может указывать на несуществующий файл. Вызов возвращает 0 в случае успеха и -1 в случае ошибки.

Поскольку символьная ссылка является отдельным файлом, она может также быть открыта вызовом `open()`, для этого в `flags` нужно передать `O_NOFOLLOW`. По умолчанию `open()` открывает именно файл, на который указывает символьная ссылка.

Символьные ссылки на каталоги создавать можно.

1.2.6. Удаление файлов

```
int unlink(const char* path); // уничтожить жесткую ссылку
```

Параметр `path` должен содержать путь к файлу (но не к каталогу).

Вызов возвращает 0 в случае успеха и -1 в случае ошибки.

Вызов `unlink` уничтожает жесткую ссылку на файл, т.е. удаляет соответствующую запись из каталога. Если `path` указывает на символьную ссылку, будет уничтожен файл символьной ссылки. Окончательно файл удаляется с диска только если на него не осталось жестких ссылок.

Следует отметить, что файл не будет окончательно удален с диска до тех пор, пока он открыт хотя бы одним процессом. Если на файл больше нет жестких ссылок, но он открыт в программе, он будет удален после завершения программы.

1.2.7 Создание и удаление каталогов

```
int mkdir(const char* path, mode_t mode); //создать каталог
```

Параметр `path` содержит путь, по которому создается директория. Параметр `mode` должен содержать права доступа к создаваемой директории, однако в пределах лабораторной его можно считать равным `S_IRWXU`. Вызов возвращает 0 в случае успеха и -1 в случае ошибки.

```
int rmdir(const char* path); // удалить каталог
```

Параметр `path` содержит путь к удаляемой пустой директории. Вызов возвращает 0 в случае успеха и -1 в случае ошибки.

1.2.8 Просмотр каталогов

```
/* получить список всех элементов каталога */
int scandir(const char *dirp,
            dirent ***namelist,
            int (*filter)(const dirent *),
            int (*compar)(const dirent **, const dirent **)
);


/* функция, сравнивающая элементы каталога по имени */
int alphasort(const dirent **a, const dirent **b);
```

Формально `scandir` не является системным вызовом. Данная функция открывает файл каталога на диске, читает его и возвращает все его элементы. Путь к каталогу передается в `dirp`.

В случае успеха возвращается число элементов каталога. В случае ошибки возвращается -1. Массив элементов каталога возвращается через `namelist`. Память для массива и для структур `dirent` выделяется с помощью `malloc`. Освободить память должен сам разработчик.

Структура dirent определена POSIX, как

```
struct dirent {
      ino_t d_ino;
      char  d_name[NAME_MAX+1]; //имя файла
      /*...*/
};
```

В структуре может быть больше полей — в зависимости от ОС.

В параметре `filter` может передаваться функция-фильтр, котора отбирает только элементы каталога, подходящие под определенные условия. Если в `filer` передан `NULL`, то возвращаются все элементы каталога.

Параметр `compar` должен содержать функцию-компаратор, которая используется при сортировке элементов каталога. Для сортировки элементов каталога по имени можно передать в `compar` функцию `alphasort`.

Пример использование `scandir` можно найти в документации (см. *`man scandir`*).

```
int main(void)
{
      struct dirent **namelist;
      int n;

      n = scandir(".", &namelist, NULL, alphasort);
      if (n == -1) {
          exit(-1);
      }

      while (n--) {
          printf("%s\n", namelist[n]->d_name);
          free(namelist[n]);
      }

      free(namelist);

      exit(0);
 }
```

1.3 Полезные команды оболочки

1.3.1 Манипуляции с рабочим каталогом

```
pwd       # распечатать рабочий каталог
cd <путь> # изменить рабочий каталог
```

1.3.2 Создание файлов

В оболочке создание файла может быть произведено 2-мя способами.

Первый способ — это перенаправление потока вывода какой-либо программы. Например, текстовый файл foo с текстом «bar» может быть создан командой

```
$ echo bar > foo
```

Второй способ — команда *touch*. Данная команда обновляет имя доступа к файлу, если он существует, иначе — создает пустой файл. К примеру, данная команда создаст пустой файл foo.

```
$ touch foo
```

1.3.3 Изменение размера файла

```
truncate -s <число> <файл> # изменить размер файла на указанный
```

Если файл не существует, он будет создан и при необходимости заполнен 0.

1.3.4 Чтение и запись файлов

Прочитать весь файл можно утилитой cat. Если файл не указан, cat будет дублировать в поток вывода все данные из потока ввода.

```
cat <файл> # вывести в поток вывода все содержимое файла
```

Если необходимо вывести в поток вывода только часть файла, можно воспользоваться комбинацией команд head и tail.

```
head [-c/-n <число>] <файл> # прочесть данные в начале файла
tail [-c/-n <число>] <файл> # прочесть данные в конце файла
```

Ключ *-c* интерпретирует число как число байт, которые нужно прочесть. Ключ *-n* интерпретирует число как число линий. При отсутствии каких-либо опций подразумевается *-n 10*.

В комбинации с конвейерами данные команды позволяют читать файлы по частям. Например, данная команда выведет только линии с 100 по 120.

```
$ cat foo.txt | head -n 120 | tail -n 20
```

Запись в начало/середину файла из консоли возможна, но нетривиальна. Дополнить файл легко с использованием перенаправления >> вместо >.

```
$ echo bar > foo  # перезаписать foo
$ echo bar >> foo # дополнить foo
```

1.3.5 Создание ссылок

В оболочке оба типа ссылок могут быть созданы командой `ln`.

```
ln [-s] <имя файла> <имя ссылки> # создать ссылку на файл
```

По умолчанию данная команда создает жесткую ссылку. Для создания символьной ссылки нужно указать флаг `-s`.

```
$ ln file.txt hrdlnk.txt    # создать жесткую ссылку hrdlnk.txt
$ ln -s file.txt symlnk.txt # создать символьную ссылку symlnk.txt
```

Допустимо создавать символьные ссылки на каталоги. Жесткие ссылки на каталоги может создавать только суперпользователь, однако создание таких ссылок может приводить к зацикливанию и повреждению ФС.

1.3.6 Создание каталогов

Каталоги могут быть созданы командой mkdir:

```
mkdir [-p] <путь> # создать каталог
```

Ключ `-p` создает все каталоги в указанном пути, если они не существуют.

1.3.7 Просмотр каталогов

Просмотр элементов каталога осуществляется командой `ls`.

```
ls [-R -a -l -h] <путь> # вывести элементы каталога
```

Ключ `-R` позволяет вывести содержимое каталога и всех подкаталогов.
Ключ `-a` позволяет вывести все скрытые файлы.
Ключ `-l` позволяет вывести свойства файлов.
Ключ `-h` выводит размер файлов в человекочитаемом формате (1M вместо 1048576).

1.3.8 Удаление файлов и каталогов

Файлы и каталоги могут быть удалены командой rm

```
rm [-R -d] <путь> # удалить файл или каталог
```

По умолчанию *rm* может удалить только файл. Ключ *-d* позволяет удалить пустой каталог. Ключ *-R* позволяет удалить непустой каталог.

1.4 Полезные файлы

/dev/zero – бесконечный файл, чтение из которого возвращает 0, а запись приводит к потере записанных данных.

/dev/urandom – бесконечный файл, чтение из которого возвращает псевдослучайные данные.

1.5 Скрипты

Помимо выполнения команд, оболочка может выполнять программы, написанные на скриптовом языке оболочки (скрипты). Каждая оболочка обладает собственным языком для написания скриптов. В простейшем случае скрипт — это последовательность команд оболочки.

По умолчанию файл скрипта не является исполняемым. Чтобы выполнить скрипт, необходимо явно разрешить выполнение командой:

```
$ chmod u+x script.sh
```

После получения права на исполнение скрипт может быть запущен на выполнение так же, как и любая программа:

```
$ ./script.sh
```

2 Задание на лабораторную работу

Преподавателем задается структура некоторого каталога с его содержимым. Для файлов задаются имя, размер и контент (пустой файл, заданная строка, случайные данные). Кроме того, файл может быть символьной или жесткой ссылкой на другой файл в пределах загадываемой структуры.

Структура каталога задается во время занятия, каждая попытка сдачи лабораторной — новый каталог со своей структурой.

Необходимо:

1. Написать программу на языке C/C++, создающую заданный каталог со всем содержимым. Для манипуляции с файлами/каталогами запрещено использовать что-то, кроме приведенных в разделе 1.2 функций и системных вызовов. Результат подтвердить выводом команды ls (или tree).
2. Написать sh-скрипт, создающий тот же самый каталог в оболочке (допустимо создать каталог командами оболочки в реальном времени под наблюдением преподавателя). Результат подтвердить выводом команды ls (или tree).

Рекомендуется заранее написать заготовку для программы на C/C++.

Варианты лабораторной:

***Легкий*:** выполнить основное задание.
***Средний*:** легкий + написать программу на C/C++, удаляющую ваш каталог
Сложный*:** средний + примонтировать выданную преподавателем флешку в заданный преподавателем каталог с помощью консольной утилиты ***mount (см. *man mount*).

Пример задания:

a/b/cat.txt (“catts”)
/c/dog.txt (пустой)
/d (ссылка на /c)
/e/f.txt (жесткая ссылка на b/cat.txt)
/g.bin (500 байт, случайный)
/h.bin (35 байт, заполнен 0)

Пример соответствующего sh-скрипта:

```
mkdir a
cd a
mkdir b c e
ln -s c d
cd b
echo catts > cat.txt
cd ../c
touch dog.txt
cd ../e
ln ../b/cat.txt f.txt
```

```
head -c 500 /dev/urandom > g.bin
truncate -s 35 h.bin
cd ../..
```