

# Лабораторная работа №5

## «ВВОД/ВЫВОД»

---

### Вспоминаем

Коротко перечислим средства группирования команд и перенаправления ввода/вывода:

- `cmd1 arg ...; cmd2 arg ...; ... cmdN arg ...` - последовательное выполнение команд;
- `cmd1 arg ...& cmd2 arg ...& ... cmdN arg ...` - асинхронное выполнение команд;
- `cmd1 arg ... && cmd2 arg ...` - зависимость последующей команды от предыдущей таким образом, что последующая команда выполняется, если предыдущая выдала нулевое значение;
- `cmd1 arg ... || cmd2 arg ...` - зависимость последующей команды от предыдущей таким образом, что последующая команда выполняется, если предыдущая выдала ненулевое значение;
- `cmd > file` - стандартный вывод направлен в файл `file`;
- `cmd >> file` - стандартный вывод направлен в конец файла `file`; `cmd < file` - стандартный ввод выполняется из файла `file`;
- `cmd1 | cmd2` - конвейер команд, в котором стандартный вывод команды `cmd1` направлен на стандартный вход команды `cmd2`.

### Знакомимся. Понятие о потоке ввода-вывода

Среди всех категорий средств коммуникации наиболее употребительными являются каналы связи, обеспечивающие достаточно безопасное и достаточно информативное взаимодействие процессов.

Существует две модели передачи данных по каналам связи 0 поток ввода-вывода и сообщения. Из них более простой является потоковая модель, в которой операции передачи/приема информации вообще не интересуются содержимым того, что передается или принимается. Вся информация в канале связи рассматривается как непрерывный поток байт, не обладающий никакой внутренней структурой.

### Понятие о работе с файлами через системные вызовы и стандартную библиотеку ввода-вывода для языка C

Потоковая передача информации может осуществляться не только между процессами, но и между процессом и устройством ввода-вывода, например между процессом и диском, на котором данные представляются в виде файла. Поскольку понятие файла должно быть знакомо изучающим этот курс, а системные вызовы, используемые для потоковой работы с файлом, во многом соответствуют системным вызовам, применяемым для потокового общения процессов, мы начнем наше рассмотрение именно с механизма потокового обмена между процессом и файлом.

Как мы надеемся, из курса программирования на языке C вам известны функции работы с файлами из стандартной библиотеки ввода-вывода, такие как `fopen()`, `fread()`, `fwrite()`, `fprintf()`, `fscanf()`, `fgets()` и т.д. Эти функции входят как неотъемлемая часть в стандарт ANSI 2 на язык C и позволяют программисту получать информацию из файла или записывать ее в файл при условии, что программист обладает определенными знаниями о содержимом передаваемых данных. Так, например, функция `fgets()` используется для ввода из файла последовательности символов, заканчивающейся символом `'\n'` – перевод каретки. Функция `fscanf()` производит ввод информации, соответствующей заданному формату, и т. д. С точки зрения потоковой модели операции, определяемые функциями стандартной библиотеки ввода-вывода, не являются потоковыми операциями, так как каждая из них требует наличия некоторой структуры передаваемых данных.

В операционной системе UNIX эти функции представляют собой надстройку, сервисный интерфейс, над системными вызовами, осуществляющими прямые потоковые операции обмена информацией между процессом и файлом и не требующими никаких знаний о том, что она содержит. Чуть позже мы кратко познакомимся с системными вызовами `open()`, `read()`, `write()` и `close()`, которые применяются для такого обмена, но сначала нам нужно ввести еще одно понятие – понятие файлового дескриптора.

## Файловый дескриптор

Информация о файлах, используемых процессом, входит в состав его системного контекста и хранится в его блоке управления PCB. В операционной системе UNIX можно упрощенно полагать, что информация о файлах, с которыми процесс осуществляет операции потокового обмена, наряду с информацией о потоковых линиях связи, соединяющих процесс с другими процессами и устройствами ввода-вывода, хранится в некотором массиве, получившем название таблицы открытых файлов или таблицы файловых дескрипторов. Индекс элемента этого массива, соответствующий определенному потоку ввода-вывода, получил название файлового дескриптора для этого потока. Таким образом, файловый дескриптор представляет собой небольшое целое неотрицательное число, которое для текущего процесса в данный момент времени однозначно определяет некоторый действующий канал ввода-вывода.

Некоторые файловые дескрипторы на этапе старта любой программы ассоциируются со стандартными потоками ввода-вывода. Так, например, файловый дескриптор «0» соответствует стандартному потоку ввода, файловый дескриптор «1» стандартному потоку вывода, файловый дескриптор «2» стандартному потоку для вывода ошибок. В нормальном интерактивном режиме работы стандартный поток ввода связывает процесс с клавиатурой, а стандартные потоки вывода и вывода ошибок с текущим терминалом. Более детально строение структур данных, содержащих информацию о потоках ввода-вывода, ассоциированных с процессом, мы будем рассматривать позже, при изучении организации файловых систем в UNIX.

## Открытие файла. Системный вызов `open()`

Файловый дескриптор используется в качестве параметра, описывающего поток ввода-вывода, для системных вызовов, выполняющих операции над этим потоком. Поэтому прежде чем совершать операции чтения данных из файла и записи их в файл, мы должны поместить информацию о файле в таблицу открытых файлов и определить соответствующий файловый дескриптор. Для этого применяется процедура открытия файла, осуществляемая системным вызовом `open()`.

### Прототип системного вызова

```
#include <fcntl.h>

int open(char *path, int flags);
int open(char *path, int flags, int mode);
```

### Описание системного вызова

Системный вызов `open` предназначен для выполнения операции открытия файла и, в случае ее удачного осуществления, возвращает файловый дескриптор открытого файла (небольшое неотрицательное целое число, которое используется в дальнейшем для других операций с этим файлом).

Параметр `path` является указателем на строку, содержащую полное или относительное имя файла.

Параметр `flags` может принимать одно из следующих трех значений:

`O_RDONLY` – если над файлом в дальнейшем будут совершаться только операции чтения;

O\_WRONLY – если над файлом в дальнейшем будут осуществляться только операции записи;

O\_RDWR – если над файлом будут осуществляться и операции чтения, и операции записи.

Каждое из этих значений может быть скомбинировано посредством операции "побитовое или (|)" с одним или несколькими флагами:

O\_CREAT – если файла с указанным именем не существует, он должен быть создан;

O\_EXCL – применяется совместно с флагом O\_CREAT. При совместном их использовании и существовании файла с указанным именем, открытие файла не производится и констатируется ошибочная ситуация;

O\_NDELAY – запрещает перевод процесса в состояние ожидания при выполнении операции открытия и любых последующих операциях над этим файлом;

O\_APPEND – при открытии файла и перед выполнением каждой операции записи (если она, конечно, разрешена) указатель текущей позиции в файле устанавливается на конец файла;

O\_TRUNC – если файл существует, уменьшить его размер до 0, с сохранением существующих атрибутов файла, кроме, быть может, времен последнего доступа к файлу и его последней модификации.

Кроме того, в некоторых версиях операционной системы UNIX могут применяться дополнительные значения флагов:

O\_SYNC – любая операция записи в файл будет блокироваться (т. е. процесс будет переведен в состояние ожидания) до тех пор, пока записанная информация не будет физически помещена на соответствующий нижележащий уровень hardware;

O\_NOCTTY – если имя файла относится к терминальному устройству, оно не становится управляющим терминалом процесса, даже если до этого процесс не имел управляющего терминала.

Параметр mode устанавливает атрибуты прав доступа различных категорий пользователей к новому файлу при его создании. Он обязателен, если среди заданных флагов присутствует флаг O\_CREAT, и может быть опущен в противном случае. Этот параметр задается как сумма следующих восьмеричных значений:

0400 – разрешено чтение для пользователя, создавшего файл;

0200 – разрешена запись для пользователя, создавшего файл;

0100 – разрешено исполнение для пользователя, создавшего файл;

0040 – разрешено чтение для группы пользователя, создавшего файл;

0020 – разрешена запись для группы пользователя, создавшего файл;

0010 – разрешено исполнение для группы пользователя, создавшего файл;

0004 – разрешено чтение для всех остальных пользователей;

0002 – разрешена запись для всех остальных пользователей;

0001 – разрешено исполнение для всех остальных пользователей.

При создании файла реально устанавливаемые права доступа получаются из стандартной комбинации параметра mode и маски создания файлов текущего процесса umask, а именно  $0$  они равны  $\text{mode} \& \sim \text{umask}$ .

При открытии файлов типа FIFO системный вызов имеет некоторые особенности поведения по сравнению с открытием файлов других типов. Если FIFO открывается только для чтения, и не задан флаг O\_NDELAY, то процесс, осуществивший системный вызов, блокируется до тех пор, пока какой-либо другой процесс не откроет FIFO на запись. Если флаг O\_NDELAY задан, то возвращается значение файлового дескриптора, ассоциированного с FIFO. Если FIFO открывается только для записи, и не задан флаг O\_NDELAY, то процесс, осуществивший системный вызов, блокируется до тех пор, пока какой-либо другой процесс не откроет FIFO на чтение. Если флаг O\_NDELAY задан, то констатируется возникновение ошибки и возвращается значение -1.

### **Возвращаемое значение**

Системный вызов возвращает значение файлового дескриптора для открытого файла при нормальном завершении и значение -1 при возникновении ошибки.

Системный вызов `open()` использует набор флагов для того, чтобы специфицировать операции, которые предполагается применять к файлу в дальнейшем или которые должны быть выполнены непосредственно в момент открытия файла. Из всего возможного набора флагов на текущем уровне знаний нас будут интересовать только флаги `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_CREAT` и `O_EXCL`. Первые три флага являются взаимоисключающими: хотя бы один из них должен быть применен и наличие одного из них не допускает наличия двух других. Эти флаги описывают набор операций, которые, при успешном открытии файла, будут разрешены над файлом в дальнейшем: только чтение, только запись, чтение и запись.

Как вам известно, у каждого файла существуют атрибуты прав доступа для различных категорий пользователей. Если файл с заданным именем существует на диске, и права доступа к нему для пользователя, от имени которого работает текущий процесс, не противоречат запрошенному набору операций, то операционная система сканирует таблицу открытых файлов от ее начала к концу в поисках первого свободного элемента, заполняет его и возвращает индекс этого элемента в качестве файлового дескриптора открытого файла. Если файла на диске нет, не хватает прав или отсутствует свободное место в таблице открытых файлов, то констатируется возникновение ошибки.

В случае, когда мы допускаем, что файл на диске может отсутствовать, и хотим, чтобы он был создан, флаг для набора операций должен использоваться в комбинации с флагом `O_CREAT`. Если файл существует, то все происходит по рассмотренному выше сценарию. Если файла нет, сначала выполняется создание файла с набором прав, указанным в параметрах системного вызова. Проверка соответствия набора операций объявленным правам доступа может и не производиться. В случае, когда мы требуем, чтобы файл на диске отсутствовал и был создан в момент открытия, флаг для набора операций должен использоваться в комбинации с флагами `O_CREAT` и `O_EXCL`.

## Системные вызовы `read()`, `write()`, `close()`

Для совершения потоковых операций чтения информации из файла и ее записи в файл применяются системные вызовы `read()` и `write()`.

### Прототипы системных вызовов

```
#include <sys/types.h>
#include <unistd.h>
size_t read(int fd, void *addr, size_t nbytes);
size_t write(int fd, void *addr, size_t nbytes);
```

**Описание системных вызовов** Системные вызовы `read` и `write` предназначены для осуществления потоковых операций ввода (чтения) и вывода (записи) информации над каналами связи, описываемыми файловыми дескрипторами, т.е. для файлов, `pipe`, `FIFO` и `socket`.

Параметр `fd` является файловым дескриптором созданного ранее потокового канала связи, через который будет отсылаться или получаться информация, т. е. значением, которое вернул один из системных вызовов `open()`, `pipe()` или `socket()`.

Параметр `addr` представляет собой адрес области памяти, начиная с которого будет браться информация для передачи или размещаться принятая информация.

Параметр `nbytes` для системного вызова `write` определяет количество байт, которое должно быть передано, начиная с адреса памяти `addr`. Параметр `nbytes` для системного вызова `read` определяет количество байт, которое мы хотим получить из канала связи и разместить в памяти, начиная с адреса `addr`.

### Возвращаемые значения

В случае успешного завершения системный вызов возвращает количество реально посланных или принятых байт. Заметим, что это значение (большее или равное 0) может не

совпадать с заданным значением параметра `nbytes`, а быть меньше, чем оно, в силу отсутствия места на диске или в линии связи при передаче данных или отсутствия информации при ее приеме. При возникновении какой-либо ошибки возвращается отрицательное значение.

## Особенности поведения при работе с файлами

При работе с файлами информация записывается в файл или читается из файла, начиная с места, определяемого указателем текущей позиции в файле. Значение указателя увеличивается на количество реально прочитанных или записанных байт. При чтении информации из файла она не пропадает из него. Если системный вызов `read()` возвращает значение 0, то это означает, что файл прочитан до конца.

Мы сейчас не акцентируем внимание на понятии указателя текущей позиции в файле и взаимном влиянии значения этого указателя и поведения системных вызовов. После завершения потоковых операций процесс должен выполнить операцию закрытия потока ввода-вывода, во время которой произойдет окончательный сброс буферов на линии связи, освободятся выделенные ресурсы операционной системы, и элемент таблицы открытых файлов, соответствующий файловому дескриптору, будет отмечен как свободный. За эти действия отвечает системный вызов `close()`. Надо отметить, что при завершении работы процесса с помощью явного или неявного вызова функции `exit()` происходит автоматическое закрытие всех открытых потоков ввода-вывода.

## Системный вызов `close`

### Прототип системного вызова

```
#include <unistd.h>
int close(int fd);
```

### Описание системного вызова

Системный вызов `close` предназначен для корректного завершения работы с файлами и другими объектами ввода-вывода, которые описываются в операционной системе через файловые дескрипторы: `pipe`, `FIFO`, `socket`. Параметр `fd` является дескриптором соответствующего объекта, т. е. значением, которое вернул один из системных вызовов `open()`, `pipe()` или `socket()`.

### Возвращаемые значения

Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибки.

## Код программы для записи информации в файл

Для иллюстрации сказанного давайте рассмотрим следующую программу:

```
/*Программа, иллюстрирующая использование системных вызовов open(), write() и
close() для записи информации в файл */

#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
int main(){
    int fd;
    size_t size;
    char string[] = "Hello, world!";
    /* Обнуляем маску создания файлов текущего процесса для того, чтобы права
    доступа у создаваемого файла точно соответствовали параметру вызова open() */
    (void)umask(0);
```

```

/* Попробуем открыть файл с именем myfile в текущей директории только для
операций вывода. Если файла не существует, попробуем его создать с правами
доступа 0666, т. е. read-write для всех категорий пользователей */
if((fd = open("myfile", O_WRONLY | O_CREAT, 0666)) < 0){
/* Если файл открыть не удалось, печатаем об этом сообщение и прекращаем
работу */
printf("Can\'t open file\n");
exit(-1);
}
/* Пробуем записать в файл 14 байт из нашего массива, т.е. всю строку "Hello,
world!" вместе с признаком конца строки */
size = write(fd, string, 14);
if(size != 14){
/* Если записалось меньшее количество байт, сообщаем об ошибке */
printf("Can\'t write all string\n");
exit(-1);
}
/* Закрываем файл */
if(close(fd) < 0){
printf("Can\'t close file\n");
}
return 0;
}

```

Обратите внимание на использование системного вызова **umask()** с параметром **0** для того, чтобы права доступа к созданному файлу точно соответствовали указанным в системном вызове **open()**.

Если вам необходимо считывать стандартный поток ввода, то используйте файловый дескриптор равный файловому дескриптору стандартного потока ввода, то есть **fd=0**. Если вам необходимо вывести информацию на экран (стандартный поток вывода), то используйте **fd=1** и системный вызов **write()**.

Пример чтения строки из стандартного потока ввода и запись строки в файл.

```

#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
int main(){
    int fd;
    size_t sizeRead, sizeWrite;
    char string[255];
/*Считываем данные из стандартного потока ввода*/
    sizeRead = read(0, string, 255);
    if(sizeRead <= 0){
        printf("Can\'t read.\n");
        return(-1);
    } ;
    (void)umask(0);
    if((fd = open("myfile", O_WRONLY | O_CREAT, 0666)) < 0){
        printf("Can\'t open file\n");
        return(-1);
    }
/*Запись в файл*/
    sizeWrite = write(fd, string, sizeRead);
/*Вывод на экран
sizeWrite = write(1, string, sizeRead);*/
    if(sizeWrite != sizeRead){
        printf("Can\'t write all string\n");
        return(-1);
    }
    if(close(fd) < 0){
        printf("Can\'t close file\n");
    }
}

```

```
}  
    return 0;  
}
```

С помощью конвейера можно перенаправить поток вывода команды на поток ввода пользовательской программы следующим образом:

```
user@ubuntu:~$ gcc lab5.c -o lab5  
user@ubuntu:~$ ls -l | ./lab5.out
```

## Задание для выполнения

### Часть I

Ознакомиться с руководством по системным вызовам **open**, **read**, **write**, **close**. Вспомнить, что такое *конвейер* и *перенаправление ввода-вывода* (этот пункт задания может оказаться трудновыполнимым, если соответствующие знания не были приобретены в процессе работы над л/р №1-3).

### Часть II

**Вариант 1.** Написать программу, которая получает со стандартного потока ввода список файлов каталога, и выводит их в стандартный поток вывода, добавляя перед каждым именем порядковый номер. Протестировать на различных каталогах с использованием конвейеров в различных комбинациях вашей программы и команд **ls**, **sort**, **head**, **tail**.

**Вариант 2.** Написать программу, которая получает со стандартного потока ввода содержимое каталога /etc и выводит в стандартный поток вывода информацию только о файлах, владельцем которых является система (root). Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **ls**, **sort**, **head**, **tail**.

**Вариант 3.** Написать программу, которая получает со стандартного потока ввода содержимое любого текстового файла и выводит в стандартный поток вывода те строки, в которых больше 3 слов. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat**, **sort**, **head**, **tail**.

**Вариант 4.** Написать программу, которая получает со стандартного потока ввода права доступа к файлам каталога, и выводит в стандартный поток вывода те из них, у которых установлен бит запуска владельцем. Протестировать на различных каталогах с использованием конвейеров в различных комбинациях вашей программы и команд **ls**, **sort**, **head**, **tail**.

**Вариант 5.** Написать программу, которая получает со стандартного потока ввода содержимое любого текстового файла и выводит в стандартный поток вывода те его строки, которые начинаются с цифры, заменив в этих строках все буквы X на Y. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat**, **sort**, **head**, **tail**.

**Вариант 6.** Написать программу, которая получает со стандартного потока ввода список активных процессов, и выводит в стандартный поток вывода процессы только с четными PID, добавив к имени процессов любое число. Протестировать с использованием конвейеров в различных комбинациях вашей программы и команд **ps**, **sort**, **head**, **tail**.

**Вариант 7.** Написать программу, которая получает со стандартного потока ввода содержимое любого текстового файла и выводит его в стандартный поток вывода, поменяв местами буквы в середине слов (первая и последняя буквы слов остаются на своих местах). Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat**, **sort**, **head**, **tail**.

**Вариант 8.** Написать программу, которая получает со стандартного потока ввода содержимое любого текстового файла и выводит в стандартный поток вывода строки, начинающиеся на гласную букву, а в поток ошибки – порядковый номер выведенной строки. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat**, **sort**, **head**, **tail**.



**Вариант 9.** Написать программу, которая получает со стандартного потока ввода строки в формате, аналогичном `/etc/passwd` (`login : password : UID : GID : GECOS : home : shell`), и выводит в стандартный поток вывода строки, с четными значениями UID, а в поток ошибки – с нечетными. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat, sort, head, tail**.

**Вариант 10.** Написать программу, которая получает со стандартного потока ввода текст руководства и выводит в стандартный поток вывода его строки, начинающиеся на гласную букву, а в поток ошибки – порядковый номер выведенной строки. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat, sort, head, tail**.

**Вариант 11.** Написать программу, которая получает со стандартного потока ввода содержимое каталога `/dev`, после чего выводит в стандартный поток вывода символьные устройства, а в стандартный поток ошибок выводит блочные устройства. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **ls, sort, head, tail**.

**Вариант 12.** Написать программу, которая получает со стандартного потока содержимое любого каталога и выводит в стандартный поток вывода имя файла и в стандартный поток ошибок дату создания файла. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **ls, sort, head, tail**.

**Вариант 13.** Написать программу, которая получает со стандартного потока ввода содержимое каталога `/dev` и выводит в стандартный поток вывода информацию только о символьных ссылках (название ссылки и путь к файлу, на который указывает ссылка). Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **ls, sort, head, tail**.

**Вариант 14.** Написать программу, которая получает со стандартного потока ввода содержимое файла `/etc/group` и выводит в стандартный поток вывода название всех групп пользователей. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **cat, sort, head, tail**.

**Вариант 15.** Написать программу, которая получает со стандартного потока ввода права доступа к файлам каталога, и выводит их в стандартный поток вывода, подменяя группы прав следующим образом: `г` на `Ч`, `w` на `З`, `x` на `В`. Протестировать на различных каталогах с использованием конвейеров в различных комбинациях вашей программы и команд **ls, sort, head, tail**.

**Вариант 16.** Написать программу, которая получает со стандартного потока ввода любое руководство, и выводит в стандартный поток вывода все ключи, описанные в данном руководстве. Протестировать на различных командах с использованием конвейеров в различных комбинациях вашей программы и команд **man, sort, head, tail**.

**Вариант 17.** Написать программу, которая получает со стандартного потока ввода список процессов и выводит в поток ошибок пользователей, которые запустили хоть один процесс. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **ps, sort, head, tail**.

**Вариант 18.** Написать программу, которая получает со стандартного потока ввода содержимое каталога и выводит в стандартный поток вывода информацию только о файлах, а в стандартный поток ошибок информацию о каталогах. Протестировать на различных файлах с использованием конвейеров в различных комбинациях вашей программы и команд **ls, sort, head, tail**.

**Вариант 19.** Написать программу, которая получает со стандартного потока ввода права доступа к файлам каталога, и выводит их в стандартный поток вывода, подменяя все группы прав `гwx` словом `ВСЕ`. Протестировать на различных каталогах с использованием конвейеров в различных комбинациях вашей программы и команд **ls, sort, head, tail**.

**Вариант 20.** Написать программу, которая получает со стандартного потока ввода любое руководство, и выводит в стандартный поток вывода заголовки всех секций данного руководства. Протестировать на различных командах с использованием конвейеров в различных комбинациях вашей программы и команд **man, sort, head, tail**.



### Часть III

В зависимости от варианта добавьте к своей программе следующую функциональность:  
В зависимости от варианта добавьте к своей программе следующую функциональность:

**Вариант 1.** В текущем каталоге создайте два файла, в один из которых выводите нечетные, а во второй - четные порядковые номера и соответствующие имена файлов.

**Вариант 2.** Создайте отдельные текстовые файлы для каждого пользователя, и поместите туда имена всех созданных им файлов.

**Вариант 3.** Создайте отдельные текстовые файлы, имеющие числовые значения, и поместите туда все слова, которым соответствует такое количество символов.

**Вариант 4.** Выведите в отдельные файлы списки запускаемых владельцем, читаемых владельцем и записываемых владельцем файлов.

**Вариант 5.** Откройте любой другой текстовый файл и выводите в стандартный поток вывода строки по очереди - согласно заданию Части II и из этого файла.

**Вариант 6.** Откройте любой текстовый файл и добавляйте в стандартном выводе к имени процессов не число, а очередное слово из этого файла.

**Вариант 7.** В текущем каталоге создайте файл, в котором сформируйте словарь слов с переставленными буквами, в виде: слово – совло, словарь – свлораь,

**Вариант 8.** В текущем каталоге создайте два файла, в один из которых выводите нечетные, а во второй - четные строки, начинающиеся на согласную букву.

**Вариант 9.** В текущем каталоге создайте два файла, в один из которых выводите UID и имя пользователя, а в другой – GID и имя пользователя.

**Вариант 10.** Создайте отдельные текстовые файлы для каждой секции руководства, и поместите туда эти секции.

**Вариант 11.** В текущем каталоге создать два файла, в один из которых записываются символьные устройства, а в другой блочные устройства, добавляя порядковый номер перед каждым устройством.

**Вариант 12.** Откройте любой текстовый файл и сформируйте новый файл следующего вида: имя файла из списка {слово из открытого файла} дата создания файла из списка.

**Вариант 13.** В текущем каталоге создать два файла, в один из которых записываются символьная ссылка, а в другой путь к файлу, на который указывает ссылка.

**Вариант 14.** В текущем каталоге создать два файла, в один из которых записываются группы пользователей с четным идентификатором, а в другой идентификатор и список пользователей, которые относятся к данной группе, если такие есть.

**Вариант 15.** В домашнем каталоге создайте три файла, в которые помещайте имена файлов с правами доступа следующим образом: в первый, если есть право r, во второй, если есть право w; в третий, если есть право x.

**Вариант 16.** Создайте отдельные текстовые файлы для каждого ключа руководства, и поместите туда описание применения данного ключа.

**Вариант 17.** Создайте отдельные текстовые файлы для каждого пользователя, и поместите туда все запущенные процессы данным пользователем.

**Вариант 18.** В текущем каталоге создать два файла, в один из которых записываются имена файлов, а в другой имена каталогов, добавляя к названию порядковый номер по списку.

**Вариант 19.** В домашнем каталоге создайте четыре файла, в которые помещайте имена файлов с правами доступа следующим образом: в первый, если права rwx установлены для владельца; во второй, если для группы; в третий, если для остальных; в четвертый, если такой группы прав не установлено ни для кого.

**Вариант 20.** Создайте отдельные текстовые файлы для каждой секции руководства, и поместите туда эти секции.

### Отчет:

Как и в других работах, отчет по проделанной работе представляется преподавателю в стандартной форме: на листах формата А4, с титульным листом (включающим тему, название дисциплины, номер лабораторной работы, фио, номер зачетки и пр.), целью, ходом работы и

выводами по выполненной работе. Каждое задание должно быть отражено в отчете следующим образом: 1) что надо было сделать, 2) как это сделали, 3) что получилось, и, в зависимости от задания – 4) почему получилось именно так, а не иначе. При наличии заданий с вариантами необходимо указать свой вариант, и расчет его номера, а также все вышеуказанное для данного варианта задания