# ЛАБОРАТОРНАЯ РАБОТА № 2.

## Практическое задание №4

## Программирование работы с файлами. Компиляция программы.

**Цель**. Освоить системное программирование работы с файлами в Linux.

*Краткие теоретические сведения.*

Прежде всего укажем, как писать программы на языке C и выполнять их в Linux. Используем редактор gedit. Набираем код на языке C в редакторе таким образом. Вызываем редактор

**$ gedit hello.c**

Набираем текст

```
#include <stdio.h>
main() {
        printf("Hello World\n");
    }
```

Сохраняем файл в редакторе, например **hello.c**. Будет сохранен в текущем каталоге.

Компилируем файл

**$ gcc -o hello hello.c**

Запускаем на выполнение

**$ ./hello**

//СОЗДАНИЕ ФАЙЛА

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <string.h>
  int fd;
  char *filename="myf.txt";
      int main()
  { fd= creat(filename, 0644);
    if(fd==-1)
       { printf("Error while creating file");
         exit(-1);}
      const char *buf1="we are learning linux";
      ssize_t nr;
      nr = write(fd, buf1, strlen(buf1));
   if(nr==-1)
       { printf("Error while writing file");
         exit(-1);}
     printf ("file %s created", filename);
     return 0;
  }
```

Наберем эту программу и выполним ее. После сохранения файла filepr1.c в окне редактора надо ввести сначала строку для компиляции файла

```
$gcc –o  filepr1 filepr1.c
```

а затем ввести строку для выполнения файла

```
$./filepr1.
```

Файл создаем с помощью функции операционной системы creat. В качестве первого параметра указываем имя файла.

Второй параметр представлен числовой константой 644, которая означает, что из файла можем читать, писать и выполнять.

Если файл успешно создан, то в переменной fd указывается целое число, представляющее так называемый дескриптор файла. При возникновении ошибки указывается число -1. Далее мы заносим в буфер строку, которую хотим записать в файл и представленную переменной buf1.

Записываем строку в файл с помощью команды write. Первый аргумент представляет дескриптор файла, второй буфер с подлежащими записи данными, третий – размер буфера.

Теперь несколько изменим нашу программу. Мы добавляем команду **system**(“cat myf.txt”); для выполнения системной функции вывода содержимого файла на консоль.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <string.h>
    int fd;
    char *filename="myf.txt";
    const char *buf1="Learn Linux, future programmer\n";
       int main()
{ fd = creat(filename,0644);
  if(fd==-1)
    { printf ("Error in create file\n"); }
   else
   { ssize_t nr;
     nr = write (fd, buf1, strlen(buf1));
     printf("File successfully written\n");
     system("cat myf.txt");
     close(fd);
   }
     return 0;
}
```

**Задание 1**. Изменить программу так, чтобы она сначала предложила ввести текст с консоли, а затем записала введенный текст в файл.

Теперь рассмотрим, как читать из файла в массив байтов и выполнять обработку.

### Чтение файла: системный вызов read()

Системный вызов read(), объявленный в файле unistd.h, позволяет читать данные из файла в отличие от библиотечных функций файлового ввода-вывода, которые предоставляют возможность интерпретации считываемых данных. Можно, например, записать в файл следующее содержимое:

**fscanf (filep, "%s", buffer);**
**fscanf (filep, "%d", number);**

Системный вызов read() читает данные в "сыром" виде, то есть как последовательность байтов, без какой-либо интерпретации. Ниже представлен адаптированный прототип read().

**ssize_t read (int fd, void * buffer, size_t count);**

Первый аргумент — это файловый дескриптор. Здесь больше сказать нечего.

Второй аргумент — это указатель на область памяти, куда будут помещаться данные.

Третий аргумент - это количество байтов, которые функция read() будет **пытаться** прочитать из файла. Возвращаемое значение - количество прочитанных байтов, если чтение состоялось и -1, если произошла ошибка. Заметим, что если read() возвращает значение меньше count, то это не символизирует об ошибке.

Тип **size_t** в Linux используется для хранения размеров блоков памяти. Какой тип реально скрывается за **size_t**, зависит от архитектуры: как правило, это unsigned long int или unsigned int.

Тип **ssize_t** (Signed SIZE Type) — это тот же size_t, только знаковый. Используется, например, в тех случаях, когда нужно сообщить об ошибке, вернув отрицательный размер блока памяти. Системный вызов read() именно так и поступает.

**Задание 2.** Прочитать содержимое файла.
**Задание 3.** Написать программу, которая считывает содержимое файла и выводит в отсортированном виде.

**Задание 4**. Написать программу, которая выводит меню в виде:

1. Add record in the form Name:Group
2. Find Group by Name
3. Exit

и обрабатывает пункты (пользователь вводит номер пункта):

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Задание 5.** В предыдущей программе добавить и запрограммировать пункт «Показать все записи».