

Отчёт по лабораторной работе №5

Хорошева Алёна Евгеньевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	9
4	Выполнение лабораторной работы	13
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	“Создание папки в ms”	13
4.2	“Редактор файла nano в ms”	14
4.3	“Запуск 1й программы”	14
4.4	“Копирование файла в ms”	15
4.5	“Запуск 2й программы”	15
4.6	“Запуск 2й программы с изменением функции на sprint”	15
4.7	“Запуск 3й программы”	16
4.8	“Запуск 4й программы”	16

Список таблиц

1 Цель работы

Приобрести навыки работы с файловым менеджером - mc (Midnight Commander).

Освоить программы языка ассемблера с вводом строки с клавиатуры и выводом этой строки

2 Задание

1. Откройте Midnight Commander

user@dk4n31:~\$ mc

2. Пользуясь клавишами \uparrow , \downarrow и Enter перейдите в каталог `~/work/arch-pc` созданный при выполнении лабораторной работы №4.
3. С помощью функциональной клавиши F7 создайте папку lab05 и перейдите в созданный каталог.
4. Пользуясь строкой ввода и командой touch создайте файл lab5-1.asm.
5. С помощью функциональной клавиши F4 откройте файл lab5-1.asm для редактирования во встроенном редакторе. Как правило в качестве встроенного редактора Midnight Commander используется редакторы nano или mcedit.
6. Введите текст программы из листинга 5.1 (можно без комментариев), сохраните изменения и закройте файл.
7. С помощью функциональной клавиши F3 откройте файл lab5-1.asm для просмотра. Убедитесь, что файл содержит текст программы.
8. Оттранслируйте текст программы lab5-1.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл. Программа выводит строку 'Введите строку:' и ожидает ввода с клавиатуры. На запрос введите Ваши ФИО.

```
user@dk4n31:~$ nasm -f elf lab5-1.asm
user@dk4n31:~$ ld -m elf_i386 -o lab5-1 lab5-1.o
user@dk4n31:~$ ./lab5-1
Введите строку:
Имя пользователя
user@dk4n31:~$
```

9. Скачайте файл `in_out.asm` со страницы курса в ТУИС.
10. Подключаемый файл `in_out.asm` должен лежать в том же каталоге, что и файл с программой, в которой он используется.
11. С помощью функциональной клавиши F6 создайте копию файла `lab5-1.asm` с именем `lab5-2.asm`. Выделите файл `lab5-1.asm`, нажмите клавишу F6 , введите имя файла `lab5-2.asm` и нажмите клавишу Enter.
12. Исправьте текст программы в файле `lab5-2.asm` с использованием подпрограмм из внешнего файла `in_out.asm` (используйте подпрограммы `sprintLF`, `sread` и `quit`) в соответствии с листингом 5.2. Создайте исполняемый файл и проверьте его работу.
13. В файле `lab5-2.asm` замените подпрограмму `sprintLF` на `sprint`. Создайте исполняемый файл и проверьте его работу. В чем разница?

Задания для самостоятельной работы:

1. Создайте копию файла `lab5-1.asm`. Внесите изменения в программу (без использования внешнего файла `in_out.asm`), так чтобы она работала по следующему алгоритму: • вывести приглашение типа “Введите строку:”; • ввести строку с клавиатуры; • вывести введенную строку на экран.
2. Получите исполняемый файл и проверьте его работу. На приглашение ввести строку введите свою фамилию.

3. Создайте копию файла lab5-2.asm. Исправьте текст программы с использованием подпрограмм из внешнего файла in_out.asm, так чтобы она работала по следующему алгоритму:
 - вывести приглашение типа “Введите строку:”;
 - ввести строку с клавиатуры;
 - вывести введенную строку на экран.
4. Создайте исполняемый файл и проверьте его работу.

3 Теоретическое введение

1. **Основы работы с mc** Midnight Commander (или просто mc) — это программа, которая позволяет просматривать структуру каталогов и выполнять основные операции по управлению файловой системой, т.е. mc является файловым менеджером. Midnight Commander позволяет сделать работу с файлами более удобной и наглядной. Для активации оболочки Midnight Commander достаточно ввести в командной строке mc и нажать клавишу Enter. В Midnight Commander используются функциональные клавиши F1 — F10, к которым привязаны часто выполняемые операции. Следующие комбинации клавиш облегчают работу с Midnight Commander: • Tab используется для переключения между панелями; • ↑ и ↓ используется для навигации, Enter для входа в каталог или открытия файла (если в файле расширения mc.ext заданы правила связи определённых расширений файлов с инструментами их запуска или обработки); • Ctrl + u (или через меню Команда > Переставить панели) меняет местами содержимое правой и левой панелей; • Ctrl + o (или через меню Команда > Отключить панели) скрывает или возвращает панели Midnight Commander, за которыми доступен для работы командный интерпретатор оболочки и выводимая туда информация. • Ctrl + x + d (или через меню Команда > Сравнить каталоги) позволяет сравнить содержимое каталогов, отображаемых на левой и правой панелях. Дополнительную информацию о Midnight Commander можно получить по команде man mc и на странице проекта.
2. **Структура программы на языке ассемблера NASM** Программа на языке

ассемблера NASM, как правило, состоит из трёх секций: секция кода программы (SECTION .text), секция инициированных (известных во время компиляции) данных (SECTION .data) и секция неинициализированных данных (тех, под которые во время компиляции только отводится память, а значение присваивается в ходе выполнения программы) (SECTION .bss). Таким образом, общая структура программы имеет следующий вид: SECTION .data ; Секция содержит переменные, для которых задано начальное значение SECTION .bss ; Секция содержит переменные, для которых не задано начальное значение SECTION .text ; Секция содержит код программы GLOBAL _start _start ; Точка входа в программу ... ; Текст программы mov eax,1 ; Системный вызов для выхода (sys_exit) mov ebx,0 ; Выход с кодом возврата 0 (без ошибок) int 80h ; Вызов ядра Для объявления инициированных данных в секции .data используются директивы DB, DW, DD, DQ и DT, которые резервируют память и указывают, какие значения должны храниться в этой памяти: • DB (define byte) — определяет переменную размером в 1 байт; • DW (define word) — определяет переменную размером в 2 байта (слово); • DD (define double word) — определяет переменную размером в 4 байта (двойное слово); • DQ (define quad word) — определяет переменную размером в 8 байт (четверное слово); • DT (define ten bytes) — определяет переменную размером в 10 байт. Директивы используются для объявления простых переменных и для объявления массивов. Для определения строк принято использовать директиву DB в связи с особенностями хранения данных в оперативной памяти. Синтаксис директив определения данных следующий: DB [,] [,] Для объявления неинициированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти.

3. Элементы программирования. Описание инструкции mov

Инструкция языка ассемблера mov предназначена для дублирования данных источника в приёмнике. В общем виде эта инструкция записывается в

виде

```
mov dst,src
```

Здесь операнд `dst` — приёмник, а `src` — источник. В качестве операнда могут выступать регистры (`register`), ячейки памяти (`memory`) и непосредственные значения (`const`). ВАЖНО! Переслать значение из одной ячейки памяти в другую нельзя, для этого необходимо использовать две инструкции `mov`:

```
mov eax, x  
mov y, eax
```

Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать. Использование слудующих примеров приведет к ошибке: • `mov al,1000h` — ошибка, попытка записать 2-байтное число в 1-байтный регистр; • `mov eax,cx` — ошибка, размеры операндов не совпадают

4. Элементы программирования. Описание инструкции `int`

Инструкция языка ассемблера `int` предназначена для вызова прерывания с указанным номером. В общем виде она записывается в виде

```
int n
```

Здесь `n` — номер прерывания, принадлежащий диапазону 0–255.

При программировании в Linux с использованием вызовов ядра `sys_calls` `n=80h` (принято задавать в шестнадцатеричной системе счисления).

После вызова инструкции `int 80h` выполняется системный вызов какой-либо функции ядра Linux. При этом происходит передача управления ядру операционной системы. Чтобы узнать, какую именно системную функцию нужно выполнить, ядро извлекает номер системного вызова из регистра

eax. Поэтому перед вызовом прерывания необходимо поместить в этот регистр нужный номер. Кроме того, многим системным функциям требуется передавать какие-либо параметры. По принятым в ОС Linux правилам эти параметры помещаются в порядке следования в остальные регистры процессора: ebx, ecx, edx. Если системная функция должна вернуть значение, то она помещает его в регистр eax.

5. Элементы программирования. Системные вызовы для обеспечения диалога с пользователем

Простейший диалог с пользователем требует наличия двух функций — вывода текста на экран и ввода текста с клавиатуры. Простейший способ вывести строку на экран — использовать системный вызов `write`. Этот системный вызов имеет номер 4, поэтому перед вызовом инструкции `int` необходимо поместить значение 4 в регистр `eax`. Первым аргументом `write`, помещаемым в регистр `ebx`, задаётся дескриптор файла. Для вывода на экран в качестве дескриптора файла нужно указать 1 (это означает «стандартный вывод», т. е. вывод на экран). Вторым аргументом задаётся адрес выводимой строки (помещаем его в регистр `ecx`, например, инструкцией `mov ecx, msg`). Строка может иметь любую длину. Последним аргументом (т.е. в регистре `edx`) должна задаваться максимальная длина выводимой строки.

Для ввода строки с клавиатуры можно использовать аналогичный системный вызов `read`. Его аргументы – такие же, как у вызова `write`, только для «чтения» с клавиатуры используется файловый дескриптор 0 (стандартный ввод).

Системный вызов `exit` является обязательным в конце любой программы на языке ассемблер. Для обозначения конца программы перед вызовом инструкции `int 80h` необходимо поместить в регистр `eax` значение 1, а в регистр `ebx` код завершения 0.

4 Выполнение лабораторной работы

1. Открываем редактор mc через терминал и создаём каталог для выполнения лабораторной - lab05. Это мы можем сделать клавишей F7.

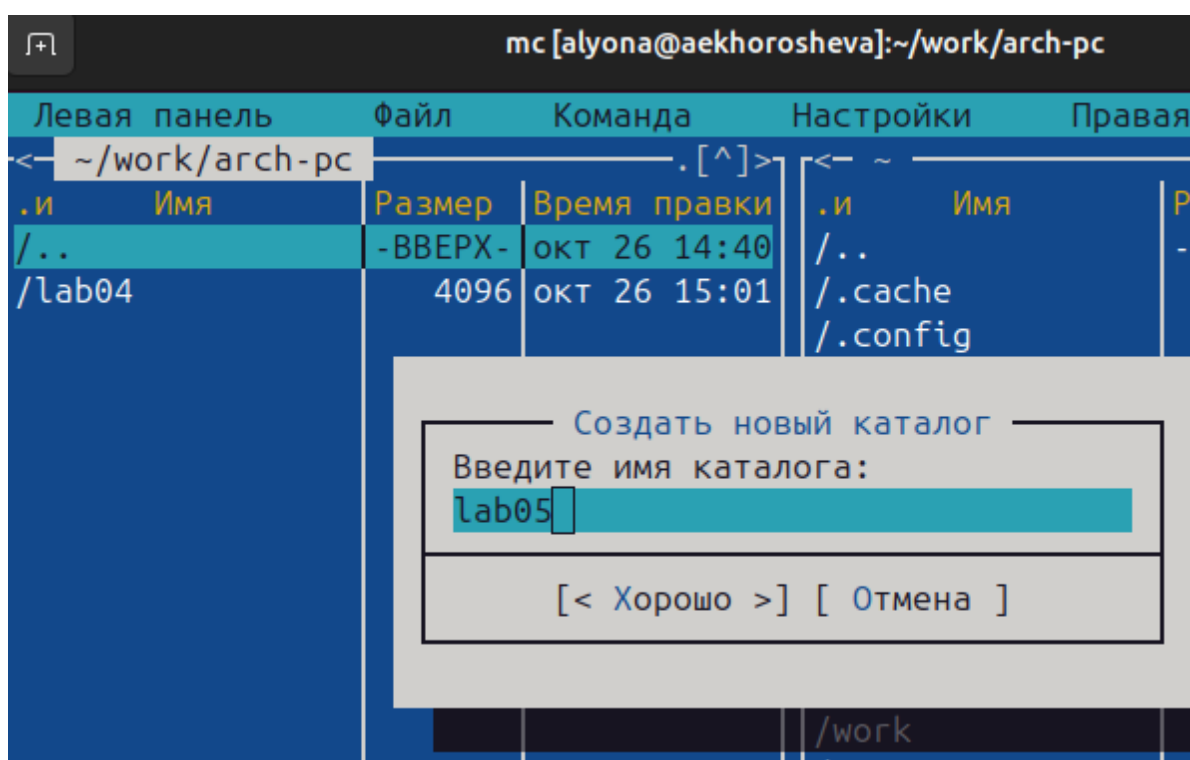
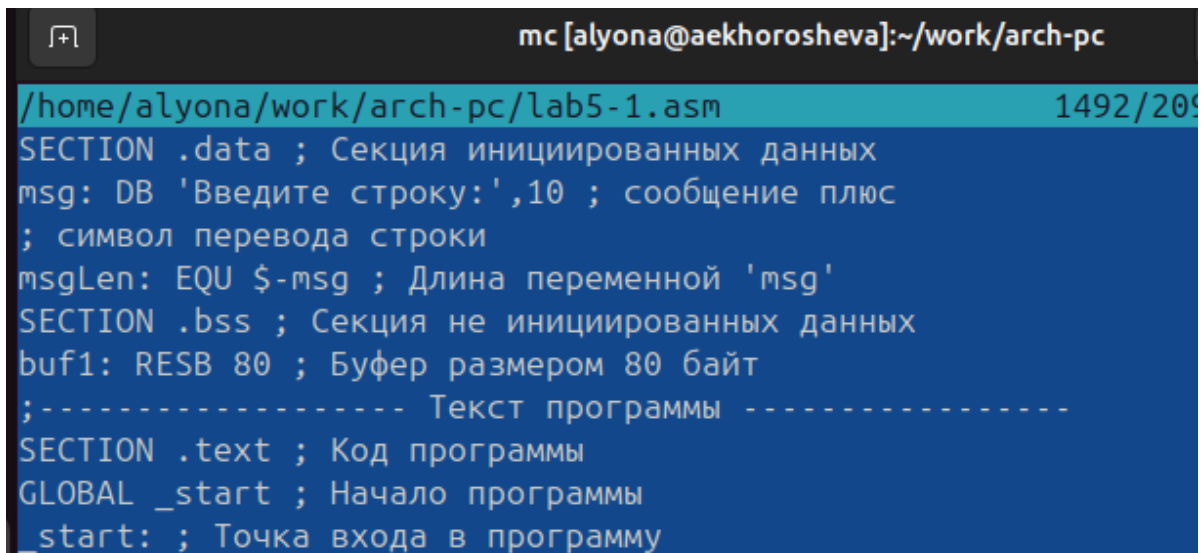


Рис. 4.1: “Создание папки в mc”

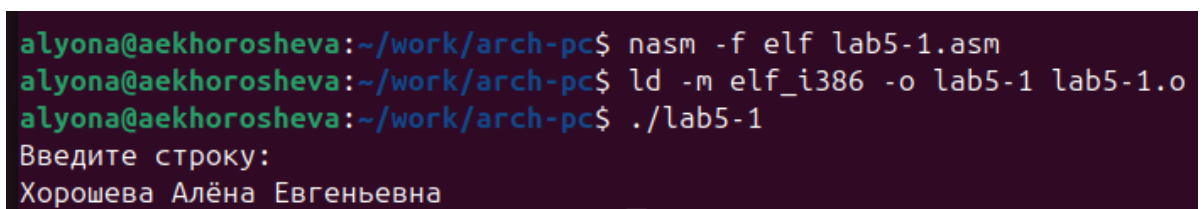
2. Командой touch в терминале создаём файл lab5-1.asm и переходим в него в окне mc с помощью клавиши F4. У нас откроется режим редактирования файла и можно выбрать один из редакторов списка. Я выбрала nano. Далее ввожу текст программы из листинга 5.1. С помощью F3 убедимся, что текст программы сохранился в файле.



```
mc [alyona@aekhorosheva]:~/work/arch-pc
/home/alyona/work/arch-pc/lab5-1.asm 1492/209
SECTION .data ; Секция иницированных данных
msg: DB 'Введите строку:',10 ; сообщение плюс
; символ перевода строки
msgLen: EQU $-msg ; Длина переменной 'msg'
SECTION .bss ; Секция не иницированных данных
buf1: RESB 80 ; Буфер размером 80 байт
;----- Текст программы -----
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
```

Рис. 4.2: “Редактор файла nano в mc”

3. Теперь нужно проверить работу программы. Для этого сначала транслируем текст в объектный файл и выполняем компоновку через команды `nasm`, `ld`. Далее запускаем исполняемый файл и вводим строку с клавиатуры.



```
alyona@aekhorosheva:~/work/arch-pc$ nasm -f elf lab5-1.asm
alyona@aekhorosheva:~/work/arch-pc$ ld -m elf_i386 -o lab5-1 lab5-1.o
alyona@aekhorosheva:~/work/arch-pc$ ./lab5-1
Введите строку:
Хорошева Алёна Евгеньевна
```

Рис. 4.3: “Запуск 1й программы”

4. Скачиваем файл `in_out.asm` с текстом нескольких функций для дальнейшего использования в программах. Необходимо поместить файл в тот же каталог `lab05`, где находится `lab5-1.asm`, чтобы мы могли импортировать подпрограммы из внешнего файла. В `mc` с помощью горячей клавиши `F6` создаём копию файла и переименовываем в `lab5-2.asm`.

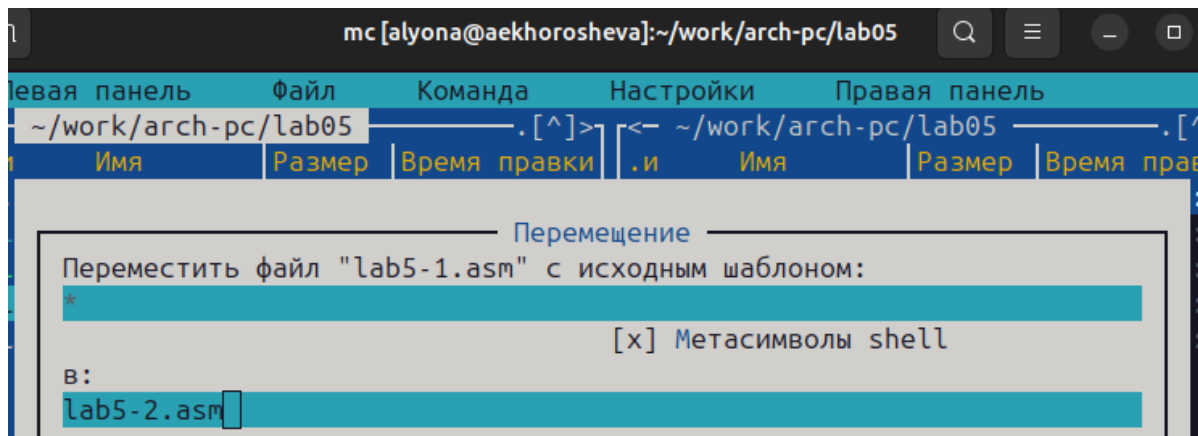


Рис. 4.4: “Копирование файла в mc”

5. Вводим текст программы из листинга 5.2 в новый файл вместо прошлой программы. Как и с файлом lab5-1.asm, мы создаём исполняемый файл и проверяем, что всё работает.

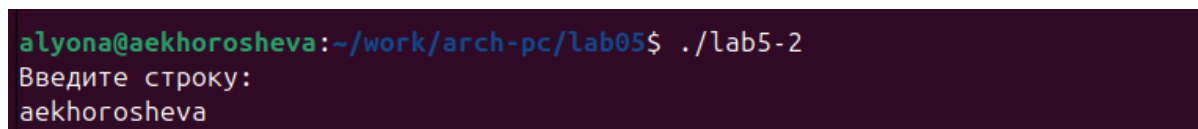


Рис. 4.5: “Запуск 2й программы”

6. Заходим в режим редактирования файла lab5-2.asm и меняем вызов подпрограммы *sprintLF* на *sprint*. После сохранения изменений и создания исполняемого файла можно запускать программу. В результате получаем, что до изменений ввод строки с клавиатуры производился *на следующей строке* (см. предыдущий скриншот), а теперь - *на той же строке*. В этом и есть отличие этих двух функций.

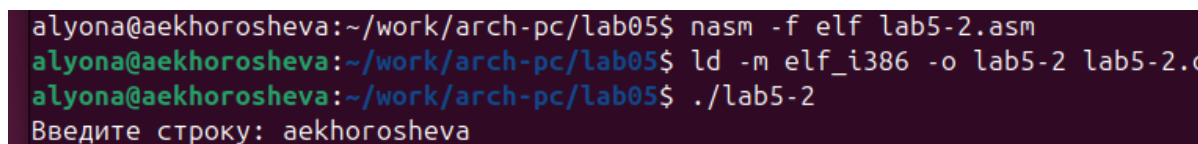
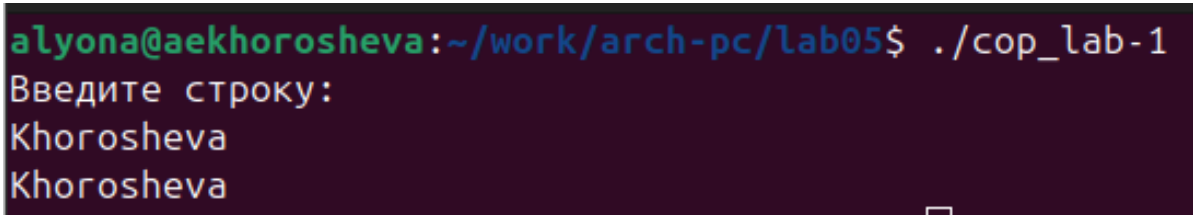


Рис. 4.6: “Запуск 2й программы с изменением функции на sprint”

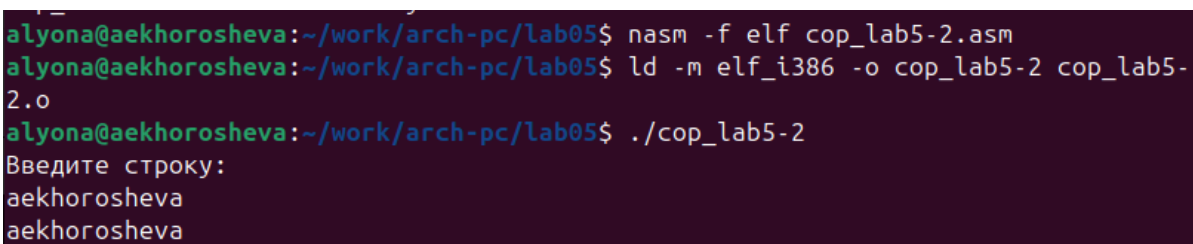
7. Создаём копию файла lab5-1.asm и называем её cop_lab5-1.asm. В тексте данной программы не используется внешний файл. Наша задача - изменить программу так, чтобы после ввода строки с клавиатуры на экран выводилась эта строка. Для этого создадим переменную(bufLen), которая будет хранить в себе длину вводимой строки, чтобы понимать сколько места она займёт - тогда, используя write, выведем исходную строку. После изменений текста программы, создаём исполняемый файл и проверяем корректность выполнения.



```
alyona@aeckhorosheva:~/work/arch-pc/lab05$ ./cop_lab-1
Введите строку:
Khorosheva
Khorosheva
```

Рис. 4.7: “Запуск 3й программы”

8. Теперь создаём копию другого файла - lab5-2.asm. Назовём её cop_lab5-2.asm. В этом тексте программы у нас используется внешний файл. Добавляем к основному тексту программы ввод переменной для хранения длины строки. Меняем программу логично предыдущему пункту, затем создаём исполняемый файл для запуска.



```
alyona@aeckhorosheva:~/work/arch-pc/lab05$ nasm -f elf cop_lab5-2.asm
alyona@aeckhorosheva:~/work/arch-pc/lab05$ ld -m elf_i386 -o cop_lab5-2 cop_lab5-2.o
alyona@aeckhorosheva:~/work/arch-pc/lab05$ ./cop_lab5-2
Введите строку:
aeckhorosheva
aeckhorosheva
```

Рис. 4.8: “Запуск 4й программы”

5 Выводы

В результате самостоятельной работы удалось двумя разными способами осуществить вывод

Получены навыки работы с файловым менеджером mc - функциональные клавиши и перемещение

Список литературы