

# **Отчёт по лабораторной работе №5**

**Дисциплина: архитектура компьютеров и операционные системы**

Хань Цзянтао

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
4.1	Начало работы с Midnight Commander .....	11
4.2	Подключение внешнего файла in_out.asm .....	14
4.3	Задания для самостоятельной работы.....	17
<b>5</b>	<b>Выводы</b>	<b>23</b>
	<b>Список литературы</b>	<b>24</b>

## Список иллюстраций

4.1	Интерфейс Midnight Commander .....	11
4.2	Создание каталога в Midnight Commander .....	12
4.3	Создание файла lab5-1.asm .....	12
4.4	Редактирование текстового файла в mc.....	13
4.5	Проверка наличия программы в файле .....	13
4.6	Запуск исполняющего файла.....	14
4.7	Копирование файла in_out.asm .....	14
4.8	Создание файла lab5-2.asm .....	15
4.9	Исправление текста программы.....	16
4.10	Запуск программы исполняемого файла .....	16
4.11	Изменения в коде программы .....	16
4.12	Запуск программы измененного файла.....	17
4.13	Создание копии файла .....	17
4.14	Внесение изменений в программу .....	18
4.15	Запуск исполняющего файла.....	19
4.16	Создание копии файла .....	20
4.17	Внесение изменений в программу .....	20
4.18	Запуск исполняющего файла.....	22

# Список таблиц

3.1	Функциональные клавиши F1 — F10 . . . . .	7
-----	---	---

# 1 Цель работы

Приобретение практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера `mov` и `int`.

## 2 Задание

1. Начало работы с Midnight Commander.
2. Подключение внешнего файла in\_out.asm.
3. Задания для самостоятельной работы.

## 3 Теоретическое введение

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. Для активации оболочки Midnight Commander достаточно ввести в командной строке `mc` и нажать клавишу `Enter`.

В табл. 3.1 приведено краткое описание функциональных клавиш `F1` — `F10`, к которым привязаны часто выполняемые операции в Midnight Commander.

Таблица 3.1: Функциональные клавиши `F1` — `F10`

Функ- цио- наль- ные клави- ши	Выполняемое действие
<code>F1</code>	вызов контекстно-зависимой подсказки
<code>F2</code>	вызов меню, созданного пользователем
<code>F3</code>	просмотр файла, на который указывает подсветка в активной панели
<code>F4</code>	вызов встроенного редактора для файла, на который указывает подсветка в активной панели
<code>F5</code>	копирование файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый на второй

Функ- цио- наль- ные клави- ши	Выполняемое действие
панели	
F6	перенос файла или группы отмеченных файлов из каталога, отображаемого
в актив- ной панели, в каталог, отобра- жае- мый на второй панели	
F7	создание подкаталога в каталоге, отображаемом в активной панели
F8	удаление файла (подкаталога) или группы отмеченных файлов
F9	вызов основного меню программы
F10	выход из программы

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).



Для объявления инициализированных данных в секции `.data` используются директивы `DB`, `DW`, `DD`, `DQ` и `DT`, которые резервируют память и указывают, какие значения должны храниться в этой памяти. 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. Инструкция языка ассемблера `mov` предназначена для дублирования данных источника в приёмнике. Инструкция языка ассемблера `int` предназначена для вызова прерывания с указанным номером. 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`. 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

### 4.1 Начало работы с Midnight Commander

Открываю Midnight Commander. (рис. 4.18).

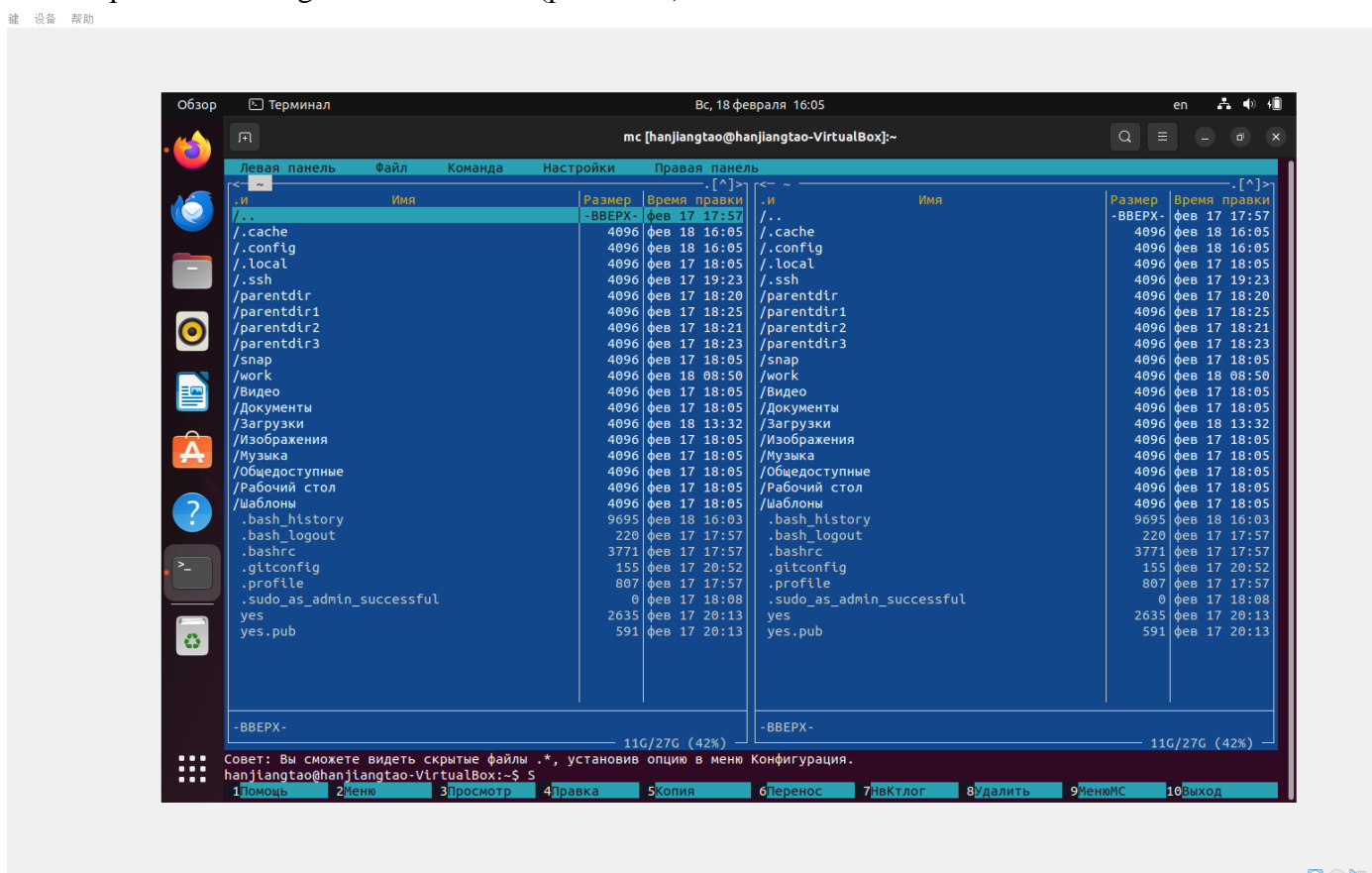


Рис. 4.1: Интерфейс Midnight Commander

Пользуясь клавишами вверх, вниз и Enter перехожу в каталог ~/work/arch-pc, созданный при выполнении лабораторной работы №4, с помощью функциональной клавиши F7 создаю папку lab05 и перехожу в созданный каталог. (рис.

4.18).

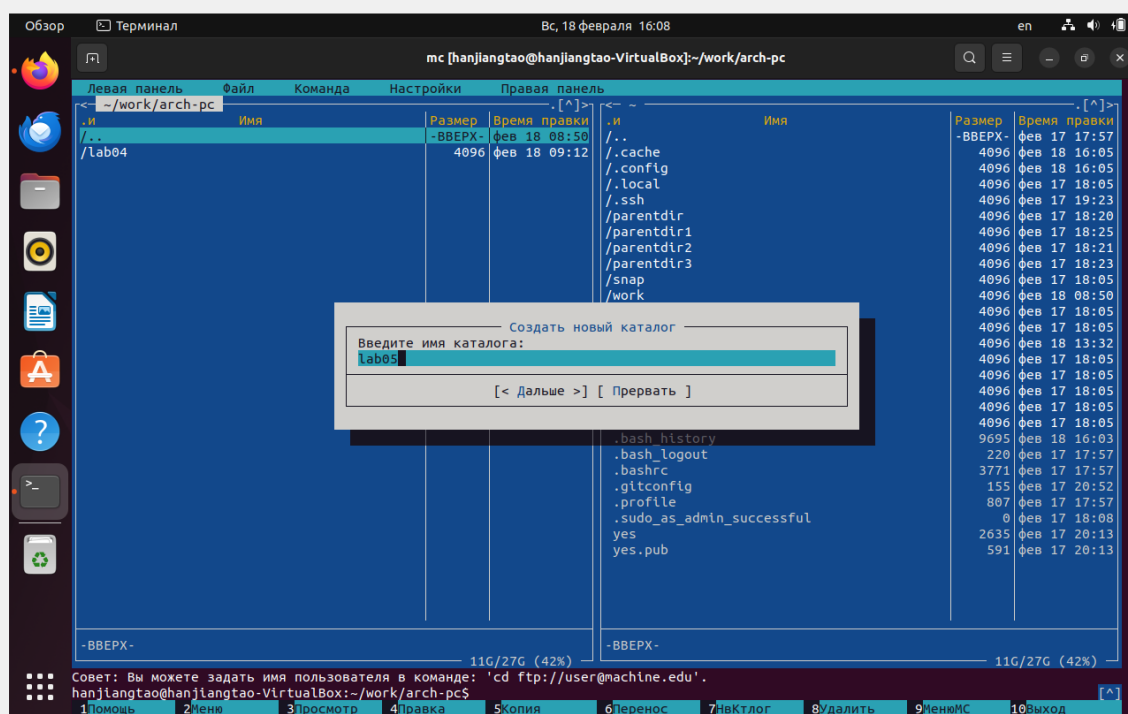


Рис. 4.2: Создание каталога в Midnight Commander

Пользуясь строкой ввода и командой touch создаю файл lab5-1.asm. (рис. 4.18).

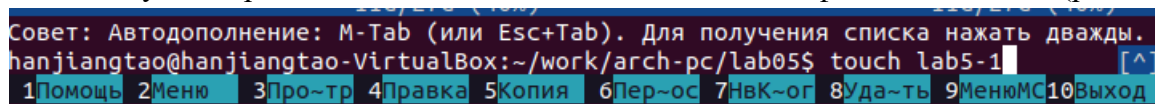


Рис. 4.3: Создание файла lab5-1.asm

С помощью функциональной клавиши F4 открываю файл lab5-1.asm для редактирования во встроенном редакторе, ввожу текст программы из листинга 5.1, сохраняю изменения с помощью функциональной клавиши F2 и закрываю файл. (рис. 4.18).

```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/la...
/home/hanjiangtao/work/~ch-pc/lab05/lab5-1.asm 234/285 82%
SECTION .data
msg: DB 'Введите строку:',10

msgLen: EQU $-msg

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h

mov eax, 3
mov edx, 0
mov edx, buf1
```

1Помощь 2Сверн 3Выход 4Hex 5Пер-ти 6 7Поиск 8Исх-ый 9Формат10Выход

Рис. 4.4:

Редактирование текстового файла в mc

С помощью функциональной клавиши F3 открываю файл lab5-1.asm для просмотра и убеждаюсь, что файл содержит текст программы. (рис. 4.18).

```
GNU nano 6.2 /home/hanjiangtao/work/arch-pc/lab05/lab5-1.asm
SECTION .data
msg: DB 'Введите строку:',10

msgLen: EQU $-msg

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h

mov eax, 3
```

Прочитано 30 строк 1

Рис. 4.5: Проверка наличия программы в файле

Компилирую текст программы lab5-1.asm в объектный файл, выполняю компоновку объектного файла и запускаю получившийся исполняемый файл, в строку ввожу свои ФИО. (рис. 4.18).

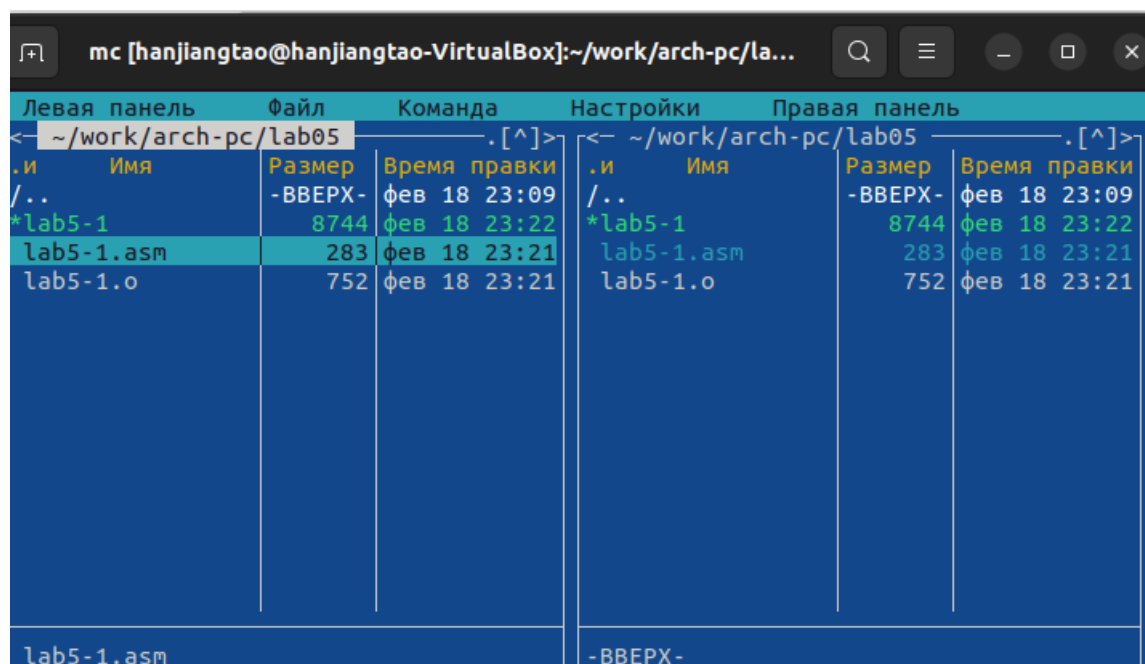


Рис. 4.6: Запуск исполняющего файла

## 4.2 Подключение внешнего файла in\_out.asm

Скачиваю файл in\_out.asm со страницы курса в ТУИС. В первой панели mc открываю каталог с файлом lab5-1.asm, а во второй панели каталог со скачанным файлом in\_out.asm. Копирую файл in\_out.asm в каталог с файлом lab5-1.asm с помощью функциональной клавиши F5. (рис. 4.18).



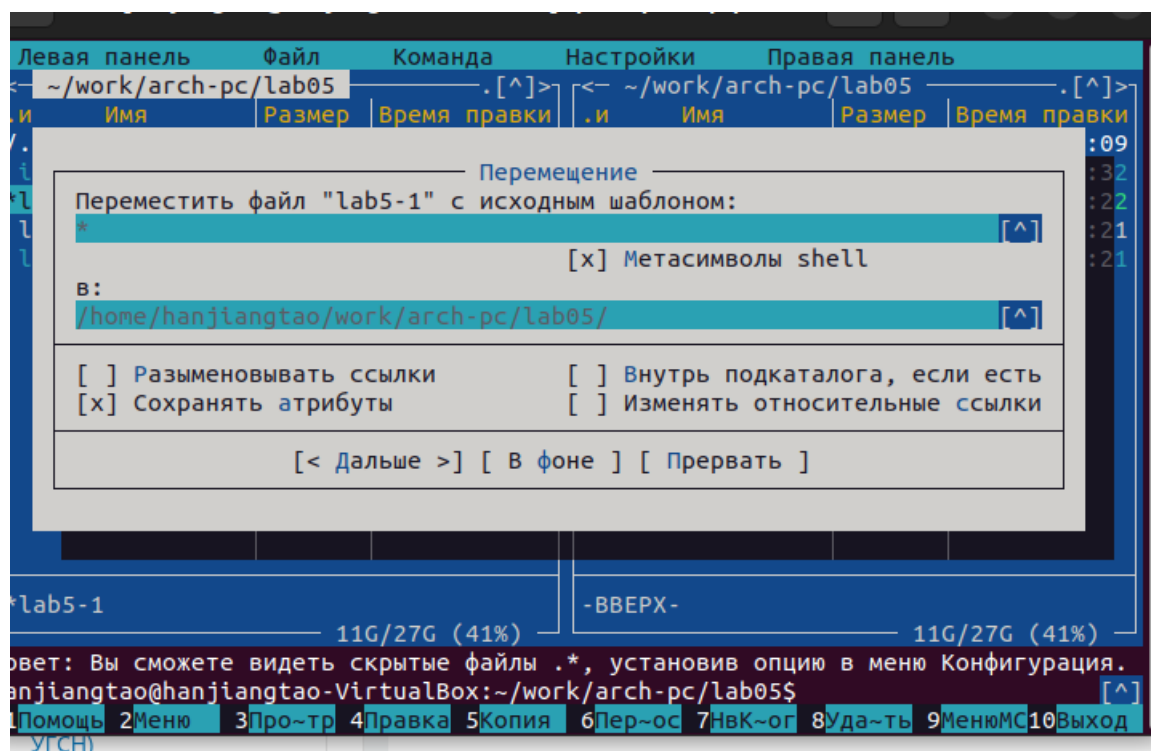


Рис. 4.7: Копирование файла in\_out.asm

С помощью функциональной клавиши F6 создаю копию файла lab5-1.asm с именем lab5-2.asm. (рис. 4.18).

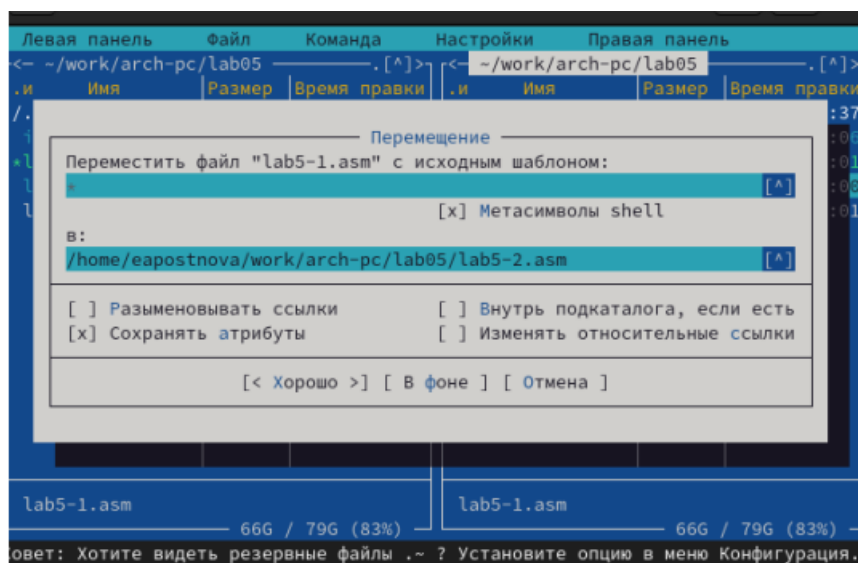


Рис. 4.8: Создание файла lab5-2.asm

Исправляю текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in\_out.asm в соответствии с листингом 5.2. (рис. 4.18).

```

mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/la...
home/hanjiangtao/work/~ch-pc/lab05/lab5-2.asm 219/219 100%
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите строку: ',0h

SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprintf
mov ecx, buf1
mov edx, 80
call read
call quit

```

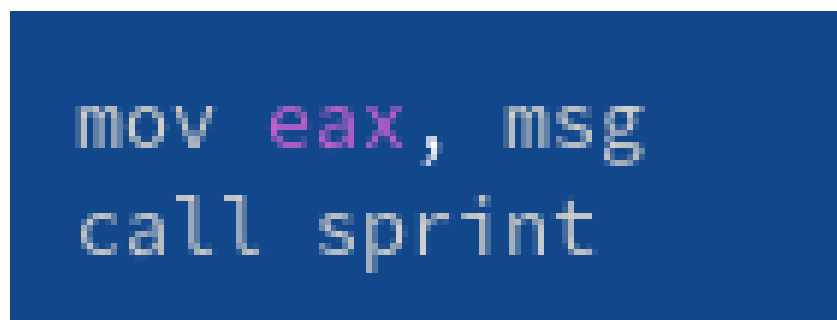
Рис. 4.9: Исправление текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.18).

mc [hanjiangtao@hanjiangtao-VirtualBox]:		
Левая панель	Файл	Команда
< ~/work/arch-pc/lab05 .[^]>		
.и	Имя	Размер
./..	-ВВЕРХ-	Время правки
in_out.asm	3942	фев 19 22:35
*lab5-1	8744	фев 18 13:32
lab5-1.asm	285	фев 18 23:22
lab5-1.o	752	фев 18 22:49
*lab5-2	9092	фев 18 23:21
lab5-2.asm	219	фев 18 23:41
lab5-2.o	1312	фев 18 23:39
lab5-2.asm		
11G/27G (40%)		
Совет: Установив переменную CDPATH, вы с		

Рис. 4.10: Запуск программы исполняемого файла

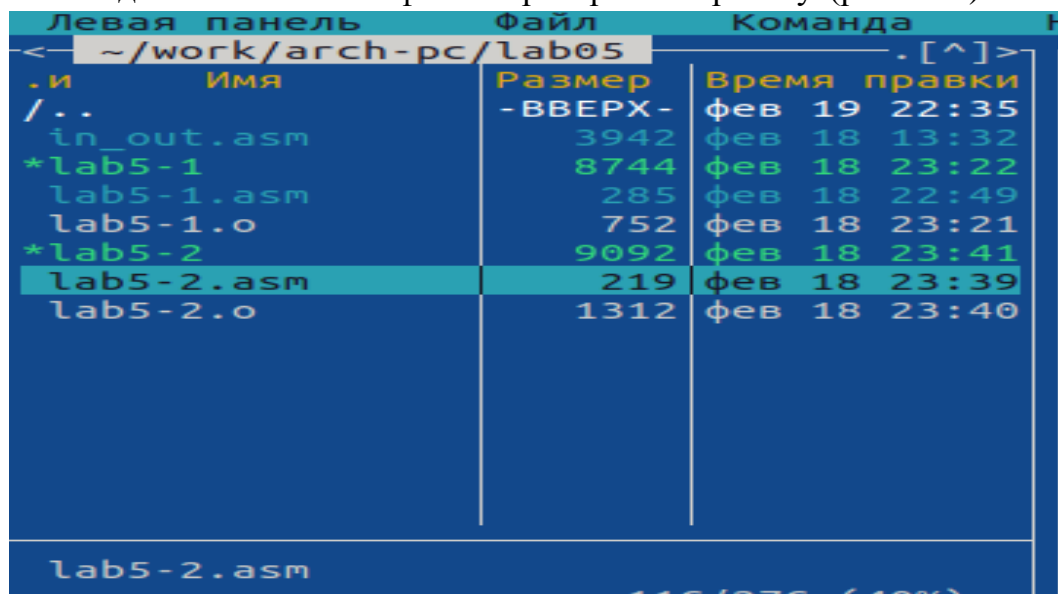
Затем в файле lab5-2.asm заменяю подпрограмму `sprintLF` на `sprint`. (рис. 4.18).

A screenshot of assembly code on a dark blue background. The code consists of two lines: 'mov eax, msg' and 'call sprint'. The word 'eax' in the first line is highlighted in a light blue color, while the rest of the code is in a light yellow color.

```
mov eax, msg  
call sprint
```

Рис. 4.11: Изменения в коде программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.18).



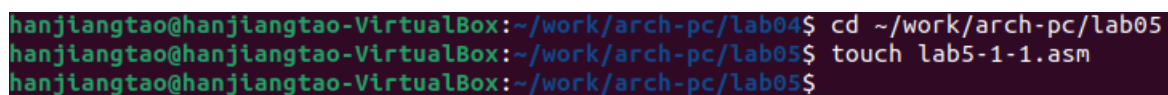
Имя	Размер	Время правки
in_out.asm	3942	фев 19 22:35
*lab5-1	8744	фев 18 13:32
lab5-1.asm	285	фев 18 23:22
lab5-1.o	752	фев 18 23:21
*lab5-2	9092	фев 18 23:41
lab5-2.asm	219	фев 18 23:39
lab5-2.o	1312	фев 18 23:40

Рис. 4.12: Запуск программы измененного файла

Разница состоит в том, что в изначальной программе ввод текста происходил с новой строки, в измененной же программе перехода на новую строку нет.

### 4.3 Задания для самостоятельной работы

1. Создаю копию файла lab5-1.asm. (рис. 4.18).



```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab04$ cd ~/work/arch-pc/lab05
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab05$ touch lab5-1-1.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab05$
```

Рис. 4.13: Создание копии файла

Вношу изменения в программу так, чтобы она выводила введенную строку на

экр. (рис. 4.18).

```

lab5-1-1.asm      [-М--]  0 L:[ 21+20  41/ 41] *(2486/2486b) <EOF>  [*]
mov edx,msgLen ; Размер строки 'msg' в 'edx'
int 80h ; Вызов ядра
; ----- системный вызов 'read' -----
; После вызова инструкции 'int 80h' программа будет ожидать ввода
; строки, которая будет записана в переменную 'buf1' размером 80 байт
mov eax, 3 ; Системный вызов для чтения (sys_read)
mov ebx, 0 ;Descriptor файла 0 - стандартный ввод
mov ecx, buf1 ; Адрес буфера под вводимую строку
mov edx, 80 ; Длина вводимой строки
int 80h ; Вызов ядра
; ----- Системный вызов 'exit' -----
; После вызова инструкции 'int 80h' программа завершит работу
mov eax,4 ; Системный вызов для выхода (sys_exit)
mov ebx,1 ; Выход с кодом возврата 0 (без ошибок)
mov ecx,buf1
mov edx,buf1
int 80h
mov eax,1
mov ebx,0
int 80h ; Вызов ядра

```

1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMC10Вых

Рис. 4.14: Внесение изменений в программу

Сам код:

SECTION .data

msg: DB 'Введите строку:',10

msgLen: EQU \$-msg

SECTION .bss

buf1: RESB 80

SECTION .text

GLOBAL \_start

\_start:

mov eax,4

mov ebx,1

mov ecx,msg

mov edx,msgLen

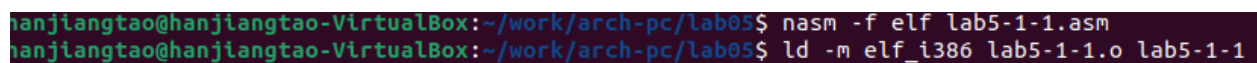
int 80h

mov eax, 3

mov ebx, 0

```
mov ecx, buf1
mov edx, 80
int 80h
mov eax, 4
mov ebx, 1
mov ecx, buf1
mov edx, buf1
int 80h
mov eax, 1
mov ebx, 0
int 80h
```

2. Получаю исполняемый файл и проверяю его работу. На приглашение ввести строку ввожу свою фамилию. (рис. 4.18).



```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab05$ nasm -f elf lab5-1-1.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab05$ ld -m elf_i386 lab5-1-1.o lab5-1-1
```

Рис. 4.15: Запуск исполняющего файла

Программа работает.

3. Создаю копию файла lab5-2.asm. (рис. 4.18).



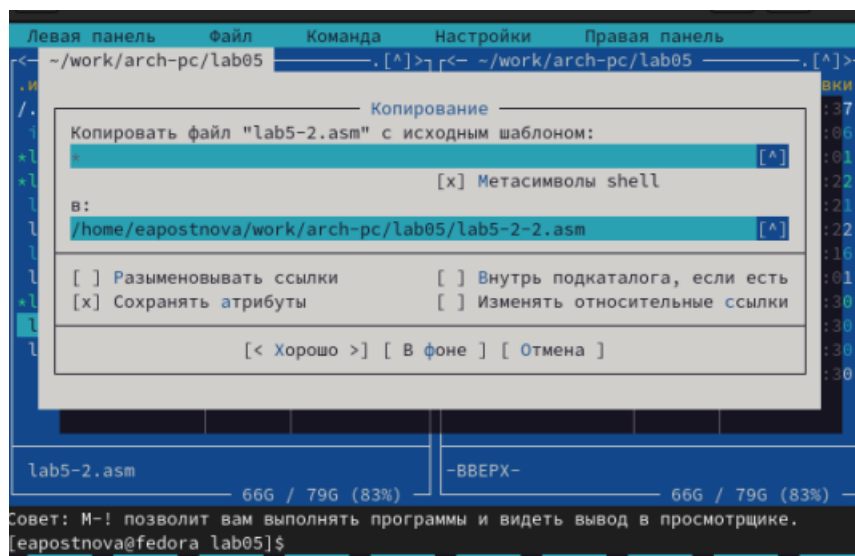


Рис. 4.16: Создание копии файла

Вношу изменения в программу с использованием подпрограмм из внешнего файла `in_out.asm` так, чтобы она выводила введенную строку на экран. (рис. 4.18).

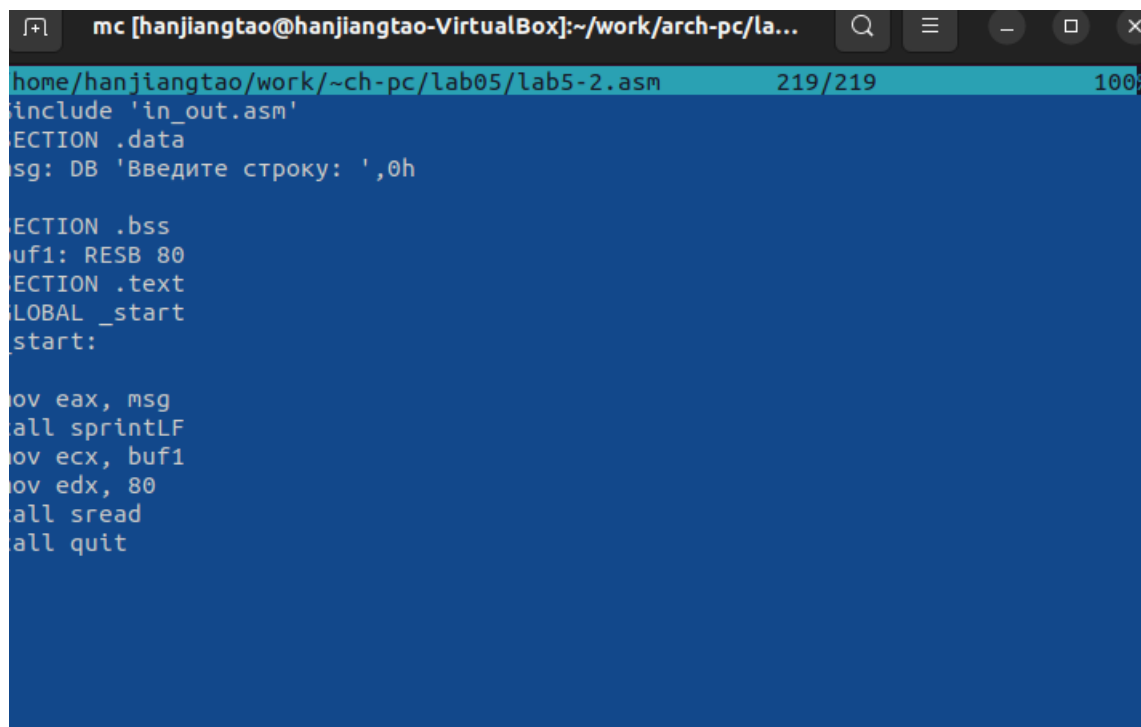


Рис. 4.17: Внесение изменений в программу

Сам код:

```

%include 'in_out.asm'

SECTION .data
msg: DB 'Введите строку:',0h

SECTION .bss
buf1: RESB 80

SECTION .text

GLOBAL _start

_start:

mov eax, msg

call sprint

mov ecx, buf1

mov edx, 80

call sread

mov eax, 4

mov ebx, 1

mov ecx, buf1

int 80h

call quit

```

4. Создаю исполняемый файл и проверяю его работу. (рис. 4.18).

Левая панель	Файл	Команда	Н
< ~/work/arch-pc/lab05			. [^]>
.и	Имя	Размер	Время правки
/..	-ВВЕРХ-		фев 19 22:35
in_out.asm	3942		фев 18 13:32
*lab5-1	8744		фев 18 23:22
lab5-1.asm	285		фев 18 22:49
lab5-1.o	752		фев 18 23:21
*lab5-2	9092		фев 18 23:41
lab5-2.asm	219		фев 18 23:39
lab5-2.o	1312		фев 18 23:40
lab5-2.asm			
116/276 (40%)			

Рис. 4.18: Запуск исполняющего файла

Программа работает.

## 5 Выводы

Благодаря данной лабораторной работе я приобрела навыки работы в Midnight Commander и освоила инструкции языка ассемблер `mov` и `int`, что поможет мне при выполнении последующих лабораторных работ.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM.— 2021.— URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix.— 2-е изд.— М.: МАКС Пресс, 2011.— URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).