

Отчёт по лабораторной работе №6

Олег Россохин

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Основы работы с Midnight Commander	6
3	Выполнение лабораторной работы	7
3.1	Подключение внешнего файла in_out.asm	14
4	Самостоятельная работа	20
5	Вопросы для самопроверки	24
	Список литературы	26

Список иллюстраций

Список таблиц

1 Цель работы

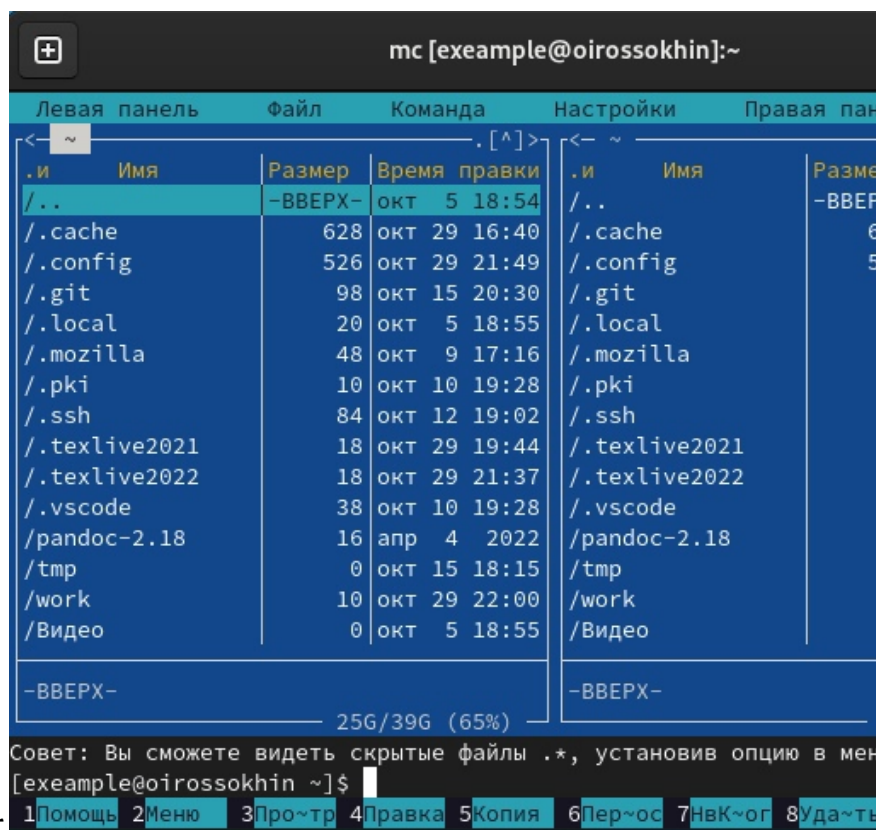
Приобретение практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера `mov` и `int`.



2 Теоретическое введение

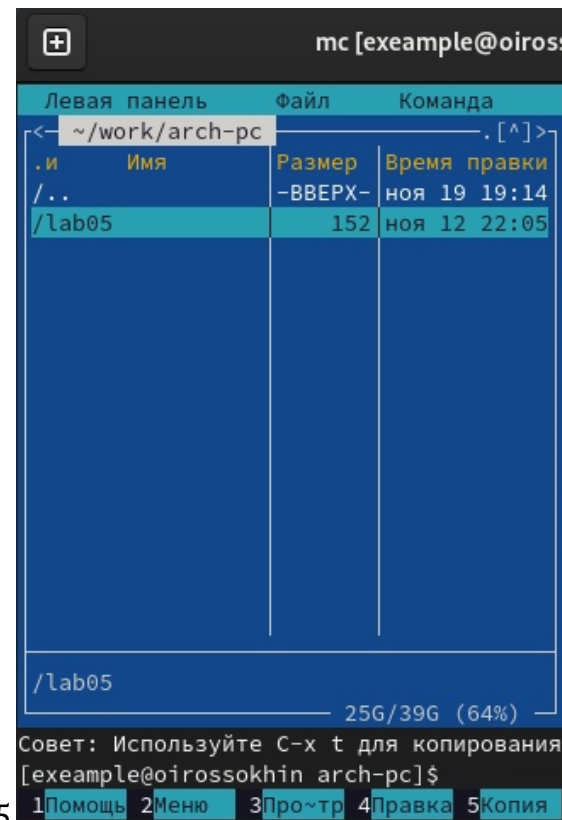
2.1 Основы работы с Midnight Commander

Midnight Commander — 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 по сравнению с терминалом.

3 Выполнение лабораторной работы

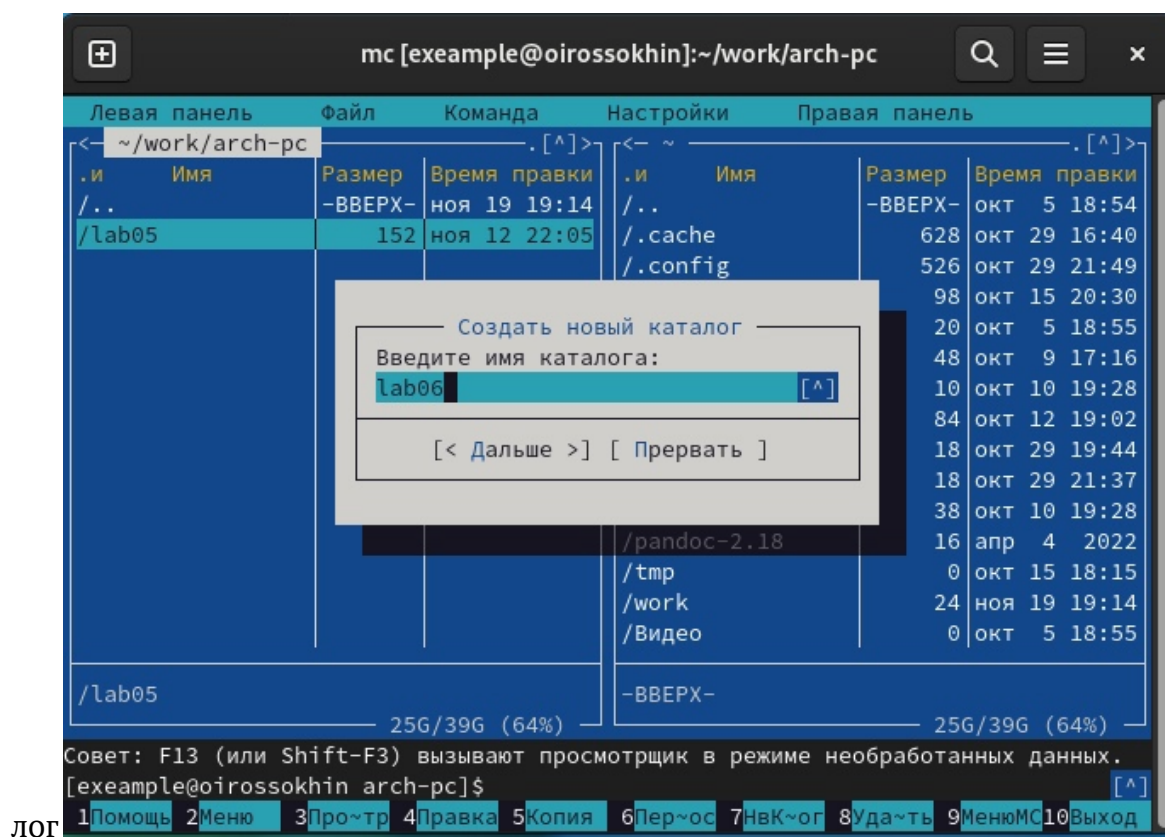


1. Откроем Midnight Commander
2. Пользуясь клавишами ,  и Enter перейдем в каталог ~/work/arch-pc, со-

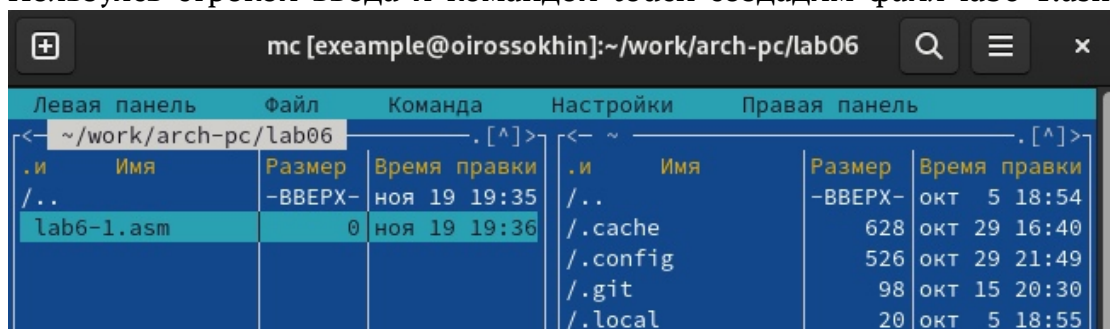


зданный при выполнении лабораторной работы №5

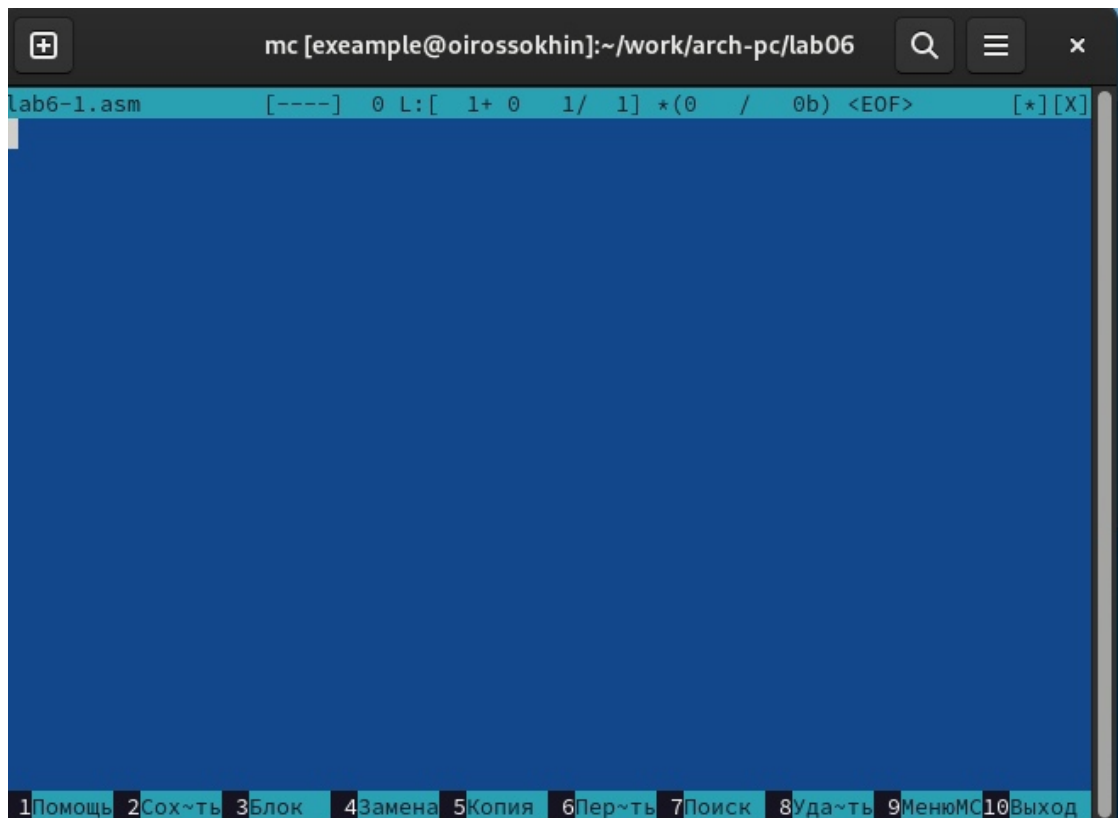
3. С помощью клавиши F7 создадим папку lab06 и перейдем в созданный ката-



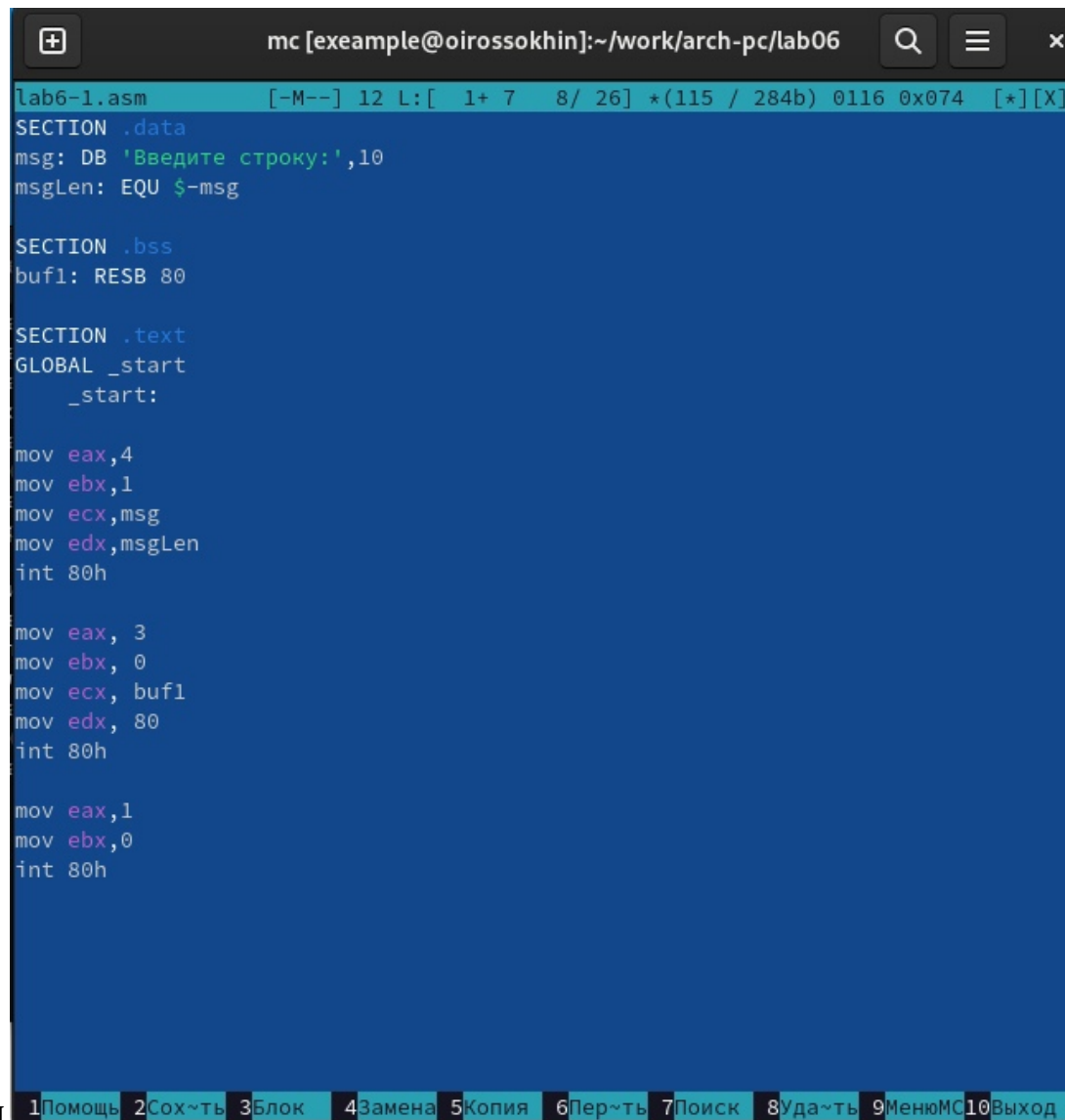
4. Пользуясь строкой ввода и командой touch создадим файл lab6-1.asm



5. С помощью клавиши F4 откроем файл lab6-1.asm для редактирования



6. Введем текст программы вывода сообщения на экран, сохраним изменения



```
mc [example@oirossokhin]:~/work/arch-pc/lab06
lab6-1.asm [-M--] 12 L: [ 1+ 7 8/ 26] *(115 / 284b) 0116 0x074 [*][X]
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,1
mov ebx,0
int 80h

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюMC10Выход
```

и закроем файл

7. С помощью клавиши F3 откроем файл lab6-1.asm для просмотра и убедимся,

```
mc [example@oirossokhin]:~/work/arch-pc
lab6-1.asm [-M--] 12 L: [ 1+ 7 8/ 26] *(115 / 2
SECTION .data
msg: DB 'Введите строку:',10
msgLen: EQU $-msg

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

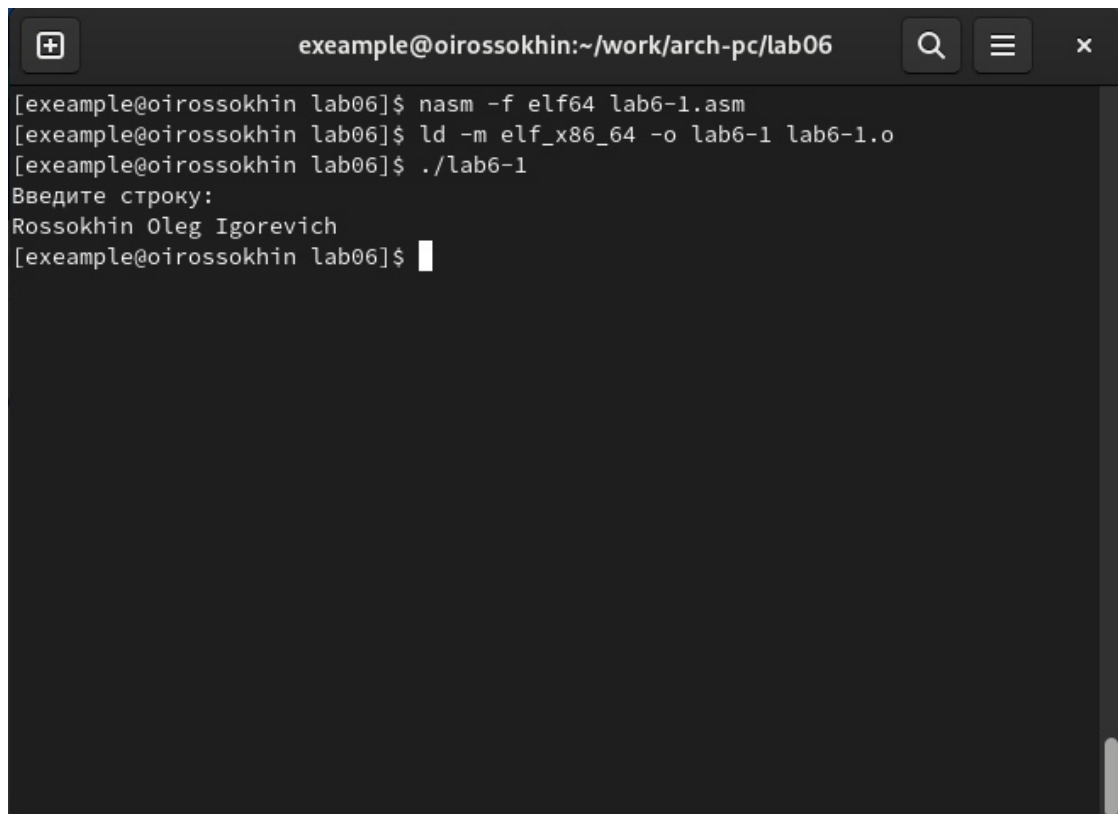
mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h

mov eax, 3
mov ebx, 0
mov ecx, buf1
mov edx, 80
int 80h

mov eax,1
mov ebx,0
int 80h
```

что файл содержит текст программы

8. Оттранслируем текст программы lab6-1.asm в объектный файл. Далее выполним компоновку объектного файла и запустим получившийся исполняемый файл. Программа напишет 'Введите строку:' и будет ожидать ввода текста с клавиатуры. На запрос введу свои ФИО.

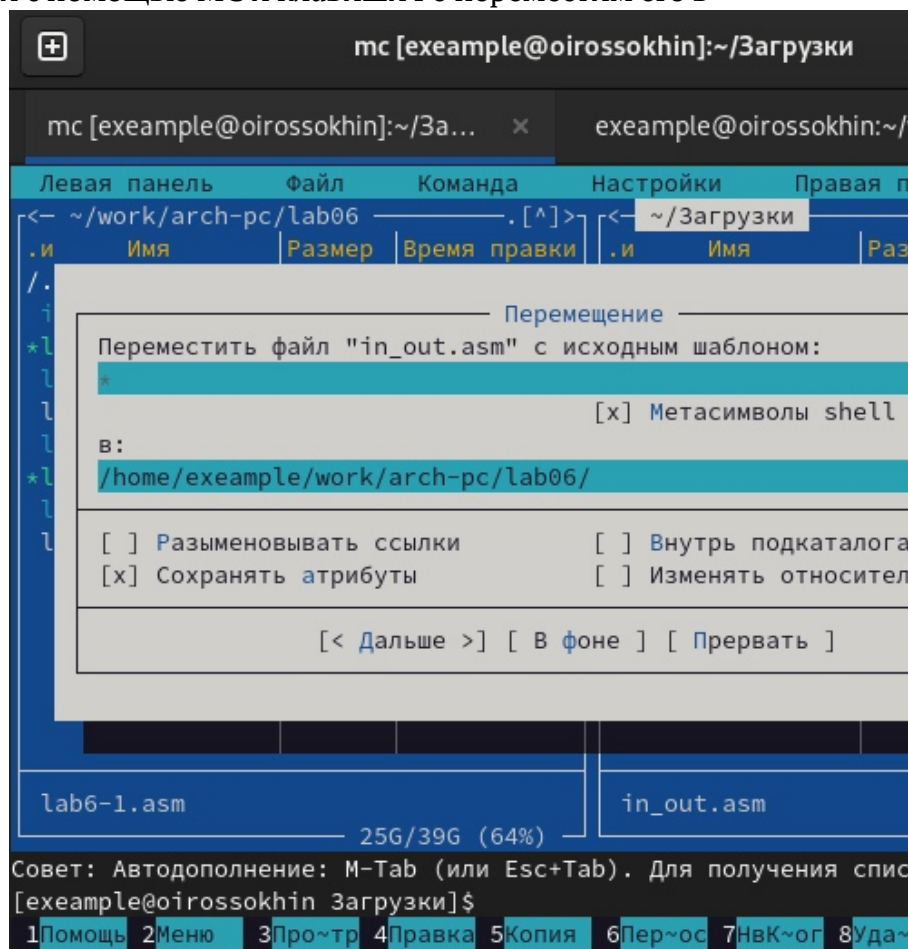


A terminal window with a dark background and light text. The title bar at the top shows a plus icon on the left, the text "exeample@oirossokhin:~/work/arch-pc/lab06" in the center, and search, menu, and close icons on the right. The terminal content shows a series of commands and their outputs. The first three lines are commands: "nasm -f elf64 lab6-1.asm", "ld -m elf_x86_64 -o lab6-1 lab6-1.o", and "./lab6-1". The fourth line is the prompt "Введите строку:". The fifth line is the input "Rossokhin Oleg Igorevich". The sixth line is the prompt "[exeample@oirossokhin lab06]\$ " followed by a cursor. A vertical scrollbar is visible on the right side of the terminal window.

```
[exeample@oirossokhin lab06]$ nasm -f elf64 lab6-1.asm
[exeample@oirossokhin lab06]$ ld -m elf_x86_64 -o lab6-1 lab6-1.o
[exeample@oirossokhin lab06]$ ./lab6-1
Введите строку:
Rossokhin Oleg Igorevich
[exeample@oirossokhin lab06]$
```

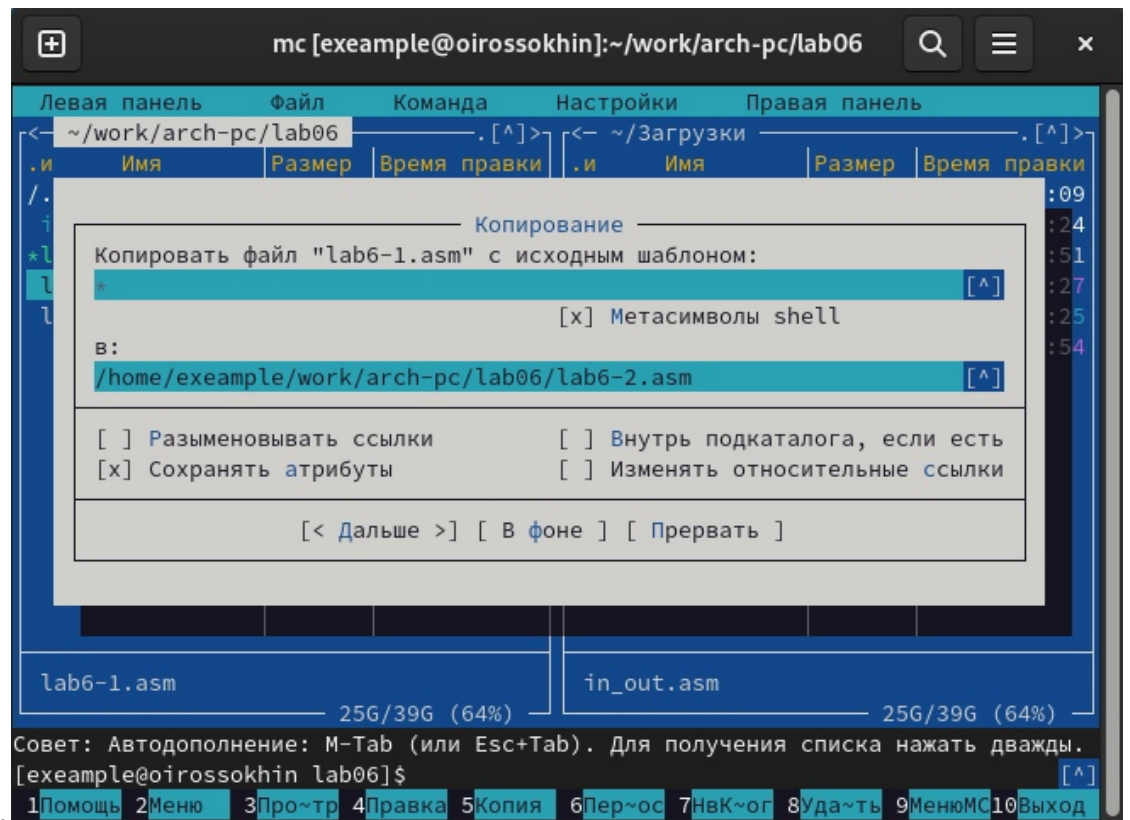
3.1 Подключение внешнего файла in_out.asm

9-10. Скачаем файл in_out.asm и с помощью МС и клавиши F6 переместим его в



каталог с программой lab6-1.asm.

11. С помощью клавиши F5 создадим копию файла lab6-1.asm с именем lab6-



2.asm.

12. Исправим текст программы в файле lab6-2.asm с использованием подпрограмм из внешнего файла in_out.asm. Создадим исполняемый файл и прове-

```
mc [exeample@oirossokhin]:~/work/arch-pc/lab06
lab6-2.asm [-M--] 9 L:[ 1+19 20/ 20] *(224 / 224b) <EOF> [*][X]
#include 'in_out.asm'

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

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

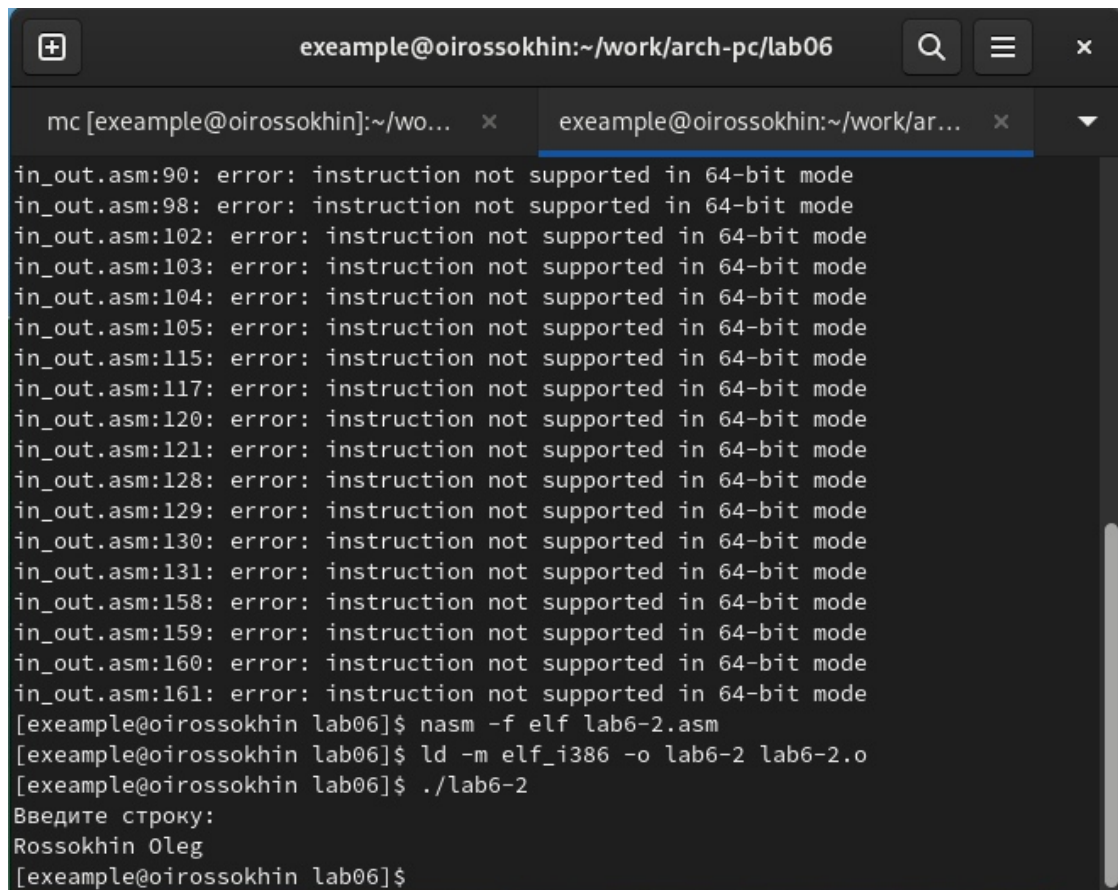
mov eax, msg
call sprintLF

mov ecx, buf1
mov edx, 80
call sread

call quit
```

1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюMC10Выход

рим его работу.

A terminal window with a dark background. The title bar shows the user 'example@oirossokhin' and the directory '~/work/arch-pc/lab06'. There are search, menu, and close icons on the right. Two tabs are open: 'mc [example@oirossokhin]:~/wo...' and 'example@oirossokhin:~/work/ar...'. The terminal output shows a series of 16 error messages from 'in_out.asm' lines 90 to 161, all stating 'error: instruction not supported in 64-bit mode'. Below these, the user runs 'nasm -f elf lab6-2.asm', 'ld -m elf_i386 -o lab6-2 lab6-2.o', and './lab6-2'. The prompt 'Введите строку:' is followed by the name 'Rossokhin Oleg', and the terminal ends at the prompt '[example@oirossokhin lab06]\$'.

13. В файле lab6-2.asm заменим подпрограмму sprintLF на sprint. Создадим ис-



```
mc [example@oirossokhin]:~/wo... x
lab6-2.asm [-M--] 11 L: [ 1+13
#include 'in_out.asm'

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

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

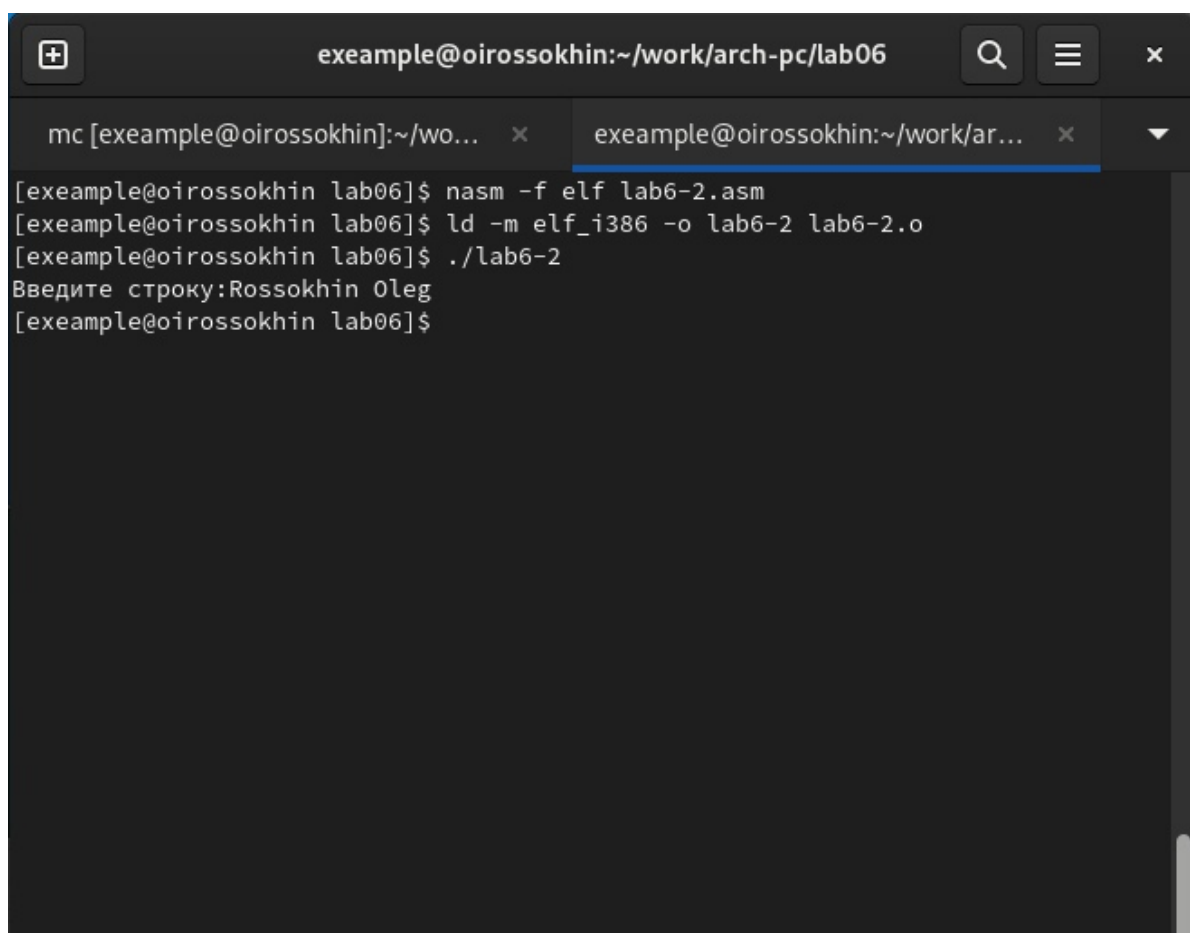
mov eax, msg
call sprint

mov ecx, buf1
mov edx, 80
call sread

call quit
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия

полняемый файл и проверим его работу. *В чем разница?*



The image shows a terminal window with a dark background. The title bar at the top reads "example@oirossokhin:~/work/arch-pc/lab06". There are two tabs open: "mc [example@oirossokhin]:~/wo..." and "example@oirossokhin:~/work/ar...". The terminal content shows the following commands and output:

```
[example@oirossokhin lab06]$ nasm -f elf lab6-2.asm
[example@oirossokhin lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[example@oirossokhin lab06]$ ./lab6-2
Введите строку:Rossokhin Oleg
[example@oirossokhin lab06]$
```

заметим, что разница двух команд `sprint` и `sprintLF` заключается в переносе строки.

4 Самостоятельная работа

1-2. Создайте копию файла lab6-1.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 введённую строку на экран. Затем получите исполняемый файл и проверьте его работу. На приглашение ввести строку введите свою фами-

Открыть + lab6-1copy.asm ~/work/arch-pc/lab06

```
SECTION .data
msg: DB 'Введите строку:',10

msgLen: EQU $-msg

SECTION .bss
buf1 RESB 80

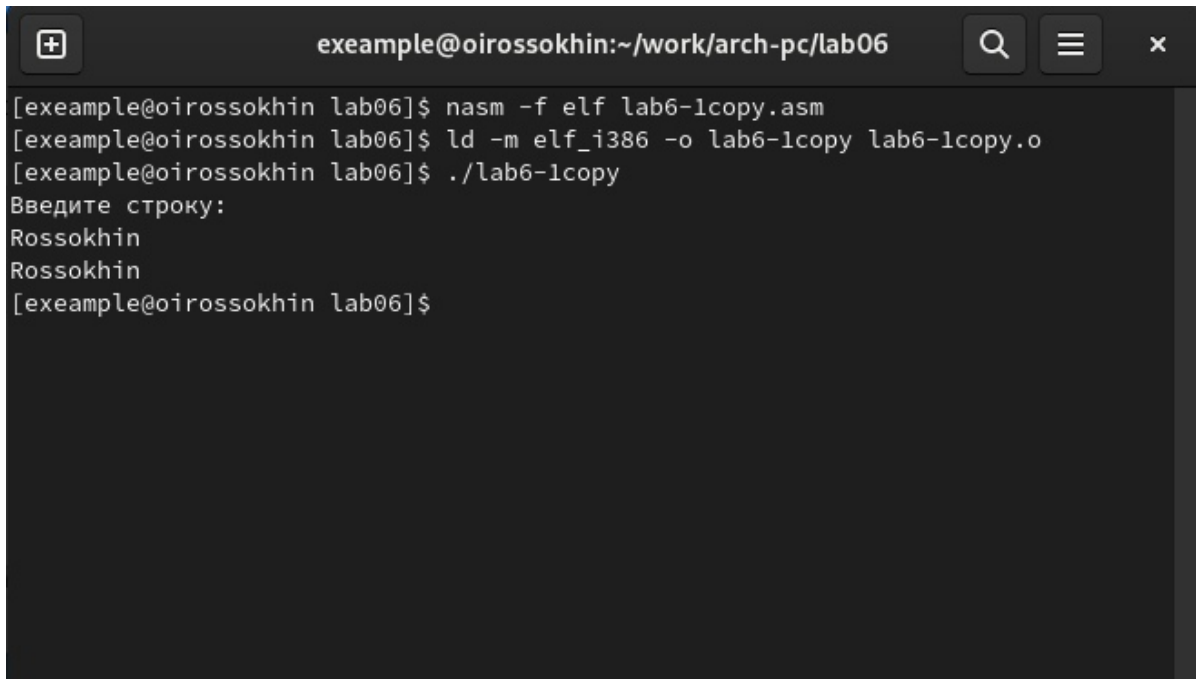
SECTION .txt
GLOBAL _start
_start:

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,msgLen
int 80h
```

ЛИЮ.



```
example@oirossokhin:~/work/arch-pc/lab06
[example@oirossokhin lab06]$ nasm -f elf lab6-1copy.asm
[example@oirossokhin lab06]$ ld -m elf_i386 -o lab6-1copy lab6-1copy.o
[example@oirossokhin lab06]$ ./lab6-1copy
Введите строку:
Rossokhin
Rossokhin
[example@oirossokhin lab06]$
```

3-4. Создайте копию файла lab6-2.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.

Создайте исполняемый файл и проверьте его работу.

Открыть ▾

+

lab6-2copy.asm
~/work/arch-pc/lab06

🔍 ☰ ✕

```
%include 'in_out.asm'|

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

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,msg
call sprintLF

mov ecx,msg
mov edx,80
call sread

mov eax,msg
call sprint

call quit
```

+

example@oirossokhin:~/work/arch-pc/lab06

🔍 ☰ ✕

```
[example@oirossokhin lab06]$ nasm -f elf lab6-2copy.asm
[example@oirossokhin lab06]$ ld -m elf_i386 -o lab6-2copy lab6-2copy.o
[example@oirossokhin lab06]$ ./lab6-2copy
Введите строку:
Rossokhin
Rossokhin
[example@oirossokhin lab06]$
```

5 Вопросы для самопроверки

1. Каково назначение mc?

1. Midnight Commander - программа, позволяющая просмотреть структуру каталогов и выполнить основные операции по управлению файловой системой.

2. Какие операции с файлами можно выполнить как с помощью команд **bash**, так и с помощью меню (комбинаций клавиш) **mc**? Приведите несколько примеров.

1. Как с помощью команды **cd** в **bash** можно перейти в нужный каталог, так и в **mc** это можно сделать с помощью клавиш.
2. Копирование файла осуществляется как в **bash** командой **cp**, так и в **mc** клавишей F5
3. Перемещение файла в **bash** осуществляется командой **mv**, а в **mc** - клавишей F6.

3. Какова структура программы на языке ассемблера NASM?

3. Программа на языке ассемблера NASM, как правило, состоит из трёх секций: **SECTION .text**(код программы); **SECTION .data**(секция инициализированных данных); **SECTION .bss**(секция неинициализированных данных).

4. Для описания каких данных используются секции `bss` и `data` в языке ассемблера NASM?
 1. `SECTION .data` содержит переменные, для которых задано начальное значение.
 2. `SECTION .bss` используется содержит переменные, для которых **не** задано начальное значение.
5. Для чего используются компоненты `db`, `dw`, `dd`, `dq` и `dt` языка ассемблера NASM?
 1. `db` - определяет переменную размером в 1 байт;
 2. `dw` - определяет переменную размером в 2 байта(слово);
 3. `dd` - определяет переменную размером в 4 байта(двойное слово);
 4. `dq` - определяет переменную размером в 8 байт;
 5. `dt` - определяет переменную размером в 10 байт.
6. Какое произойдёт действие при выполнении инструкции `mov eax, esi`?
 1. `mov eax, esi` перешлет значение регистра **esi** в регистр **eax**
7. Для чего используется инструкция `int 80h`?
 1. Инструкция `int 80h` используется для выполнения системного вызова 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. # Выводы

По итогам проделанной работы мы приобрели навыки работы с Midnight Commander, а также освоили некоторые инструкции языка ассемблер.

Список литературы

::: {#refs} :::
https://esystem.rudn.ru/pluginfile.php/1584633/mod_resource/content/1/%D0%9B%D