

Отчёт по лабораторной работе №5

Основы работы с Midnight Commander (mc). Структура программы на языке ассемблера NASM. Системные вызовы в ОС GNU Linux

Прокопьева Марина Евгеньевна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	9
4	Выводы	19
	Список литературы	20

Список иллюстраций

3.1	Открытие Midnight Commander	9
3.2	Окно Midnight Commander	10
3.3	Переход между каталогами	11
3.4	Создание каталога	11
3.5	Создание файла	12
3.6	Редактирование файла	12
3.7	Ввод текста	13
3.8	Просмотр файла	14
3.9	Запуск файла	15
3.10	Скачивание файла	15

Список таблиц

1 Цель работы

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

2 Теоретическое введение

Основы работы с Midnight Commander

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

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).

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 (учетверённое 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 неинициализированных данных в секции `.bss` используются директивы `resb`, `resw`, `resd` и другие, которые сообщают ассемблеру, что необходимо зарезервировать за данное количество ячеек памяти.

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`). В табл. 5.4 приведены варианты использования `mov` с разными операндами

ВАЖНО! Переслать значение из одной ячейки памяти в другую нельзя, для этого необходимо использовать две инструкции `mov`: `moveax, x` `movu, eax`. Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать. Использование следующих примеров приведет к ошибке: • `mov • mov al,1000h` — ошибка, попытка записать 2-байтное число в 1-байтный регистр; `eax,cx` — ошибка, размеры операндов не совпадают.

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`.

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.

3 Выполнение лабораторной работы

1. Открыла Midnight Commander

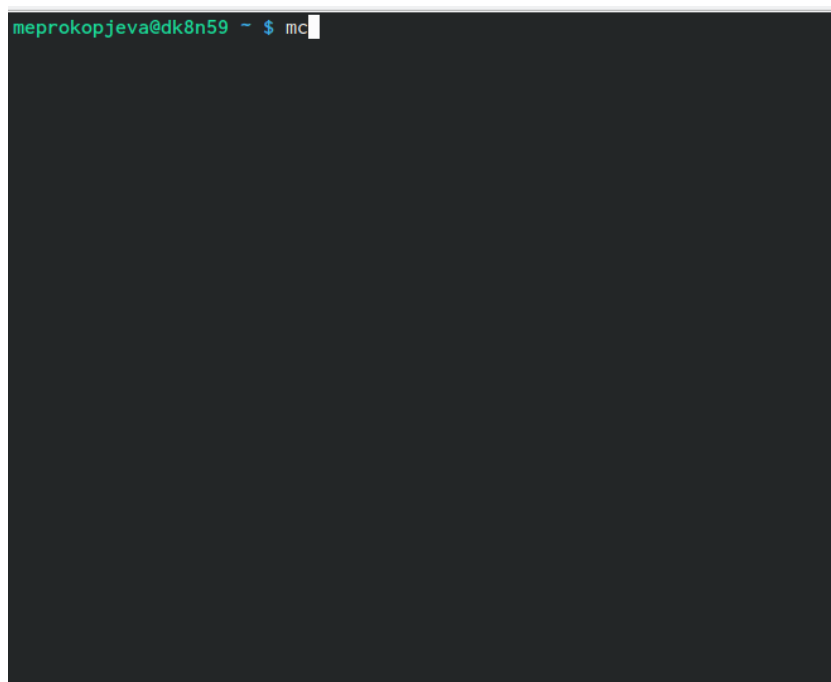


Рис. 3.1: Открытие Midnight Commander

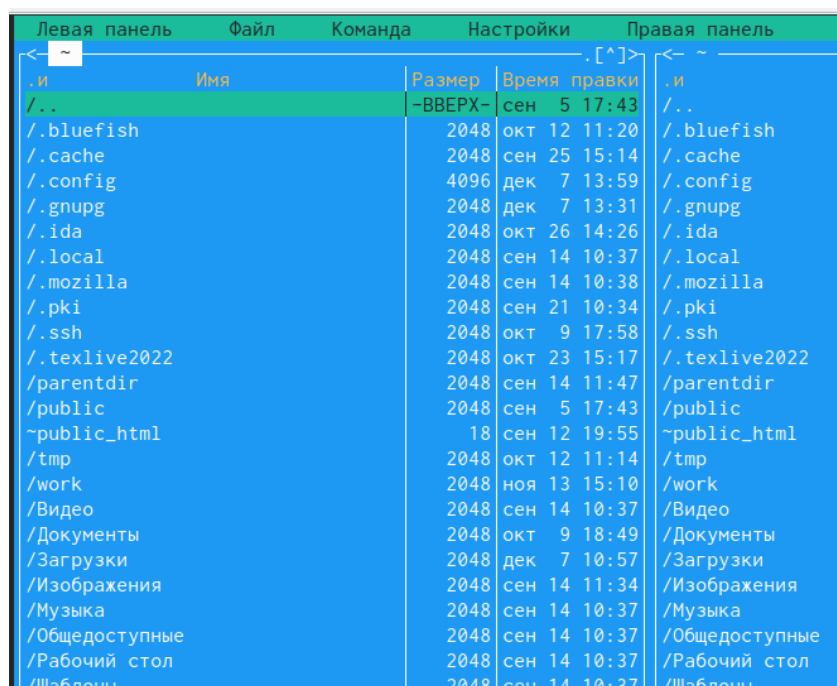


Рис. 3.2: Окно Midnight Commander

2. Пользуясь клавишами вверх, вниз и Enter перешла в каталог ~/work/arch-pc

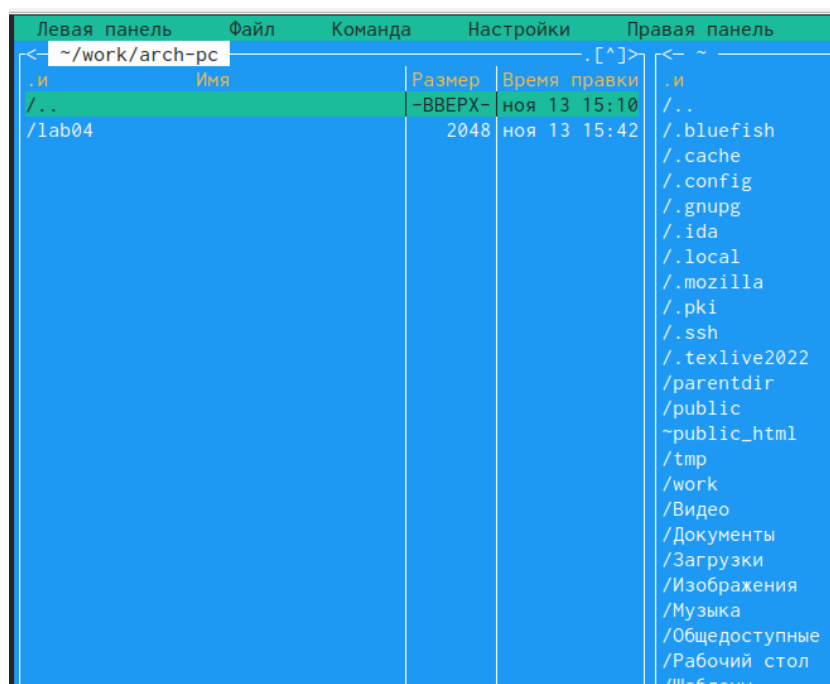


Рис. 3.3: Переход между каталогами

3. С помощью функциональной клавиши F7 создала папку lab05 и перешла в созданный каталог.

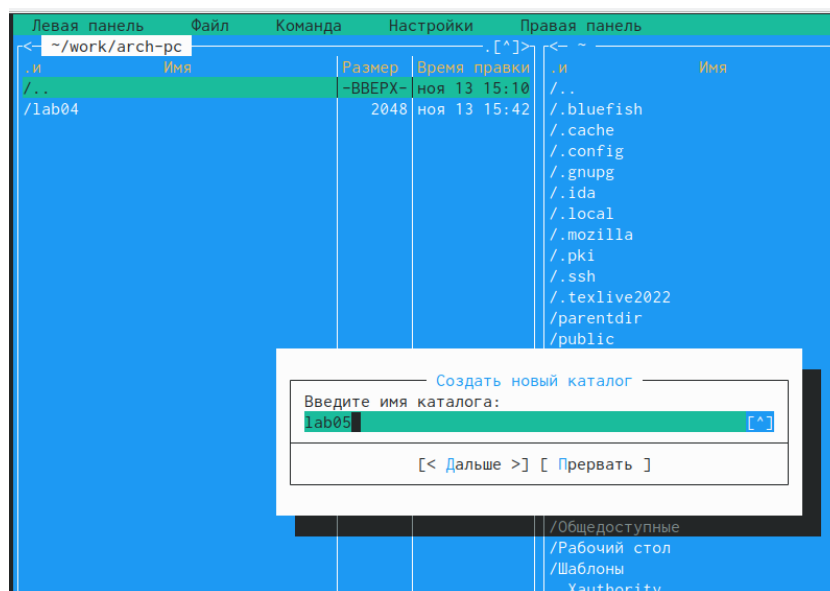


Рис. 3.4: Создание каталога

4. Пользуясь строкой ввода и командой touch создала файл lab5-1.asm

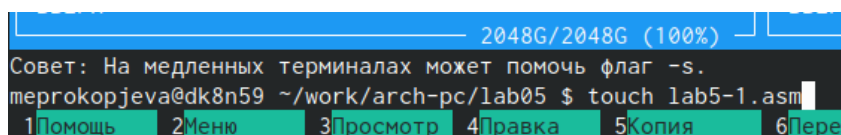


Рис. 3.5: Создание файла

5. С помощью функциональной клавиши F4 открыла файл lab5-1.asm для редактирования во встроенном редакторе.

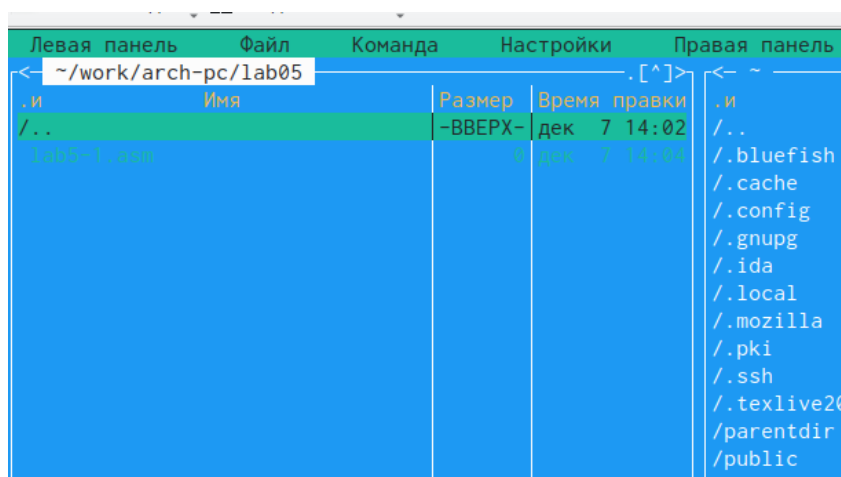


Рис. 3.6: Редактирование файла

6. Ввела текст программы из листинга 5.1 и сохранила изменения.

```
lab5-1.asm [----] 7 L: [ 1+25 26/ 26] *(277 / 27)
SECTION .data
msg: DB "введите строку", 10
msgLen: EQU $-msg

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start

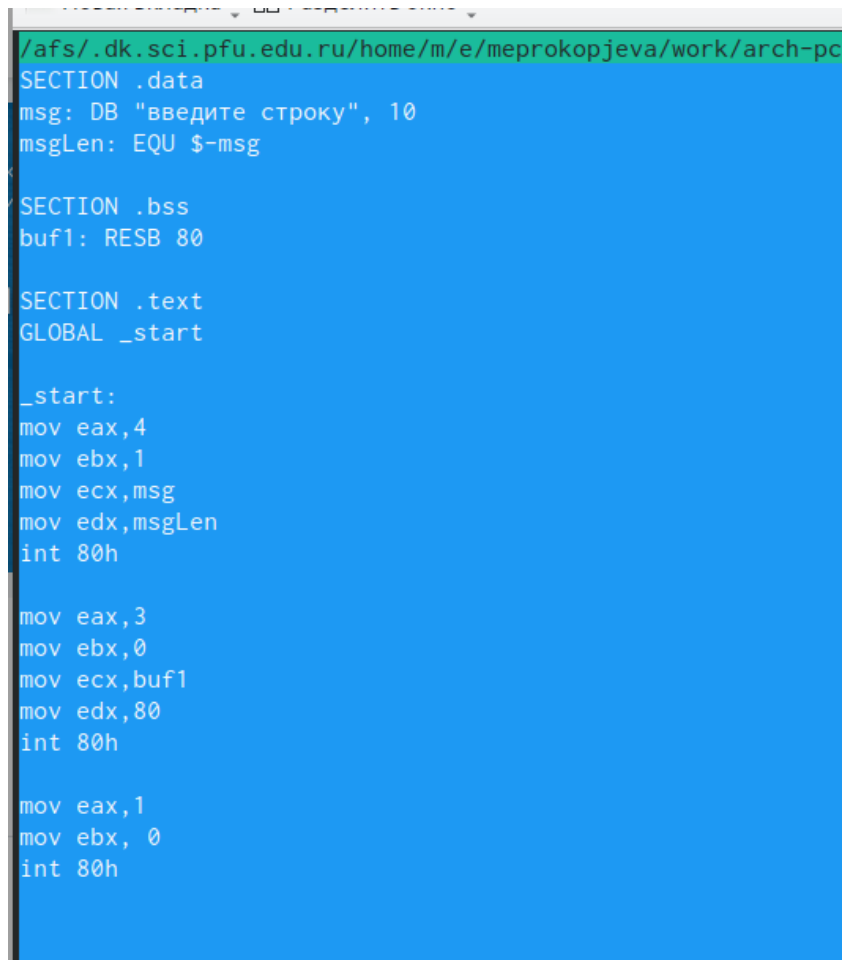
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, msgLen
int 80h

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

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

Рис. 3.7: Ввод текста

7. С помощью функциональной клавиши f3 открыла файл для просмотра и убедилась, что файл содержит тест программы



```
/afs/.dk.sci.pfu.edu.ru/home/m/e/meprokojjeva/work/arch-pc
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
```

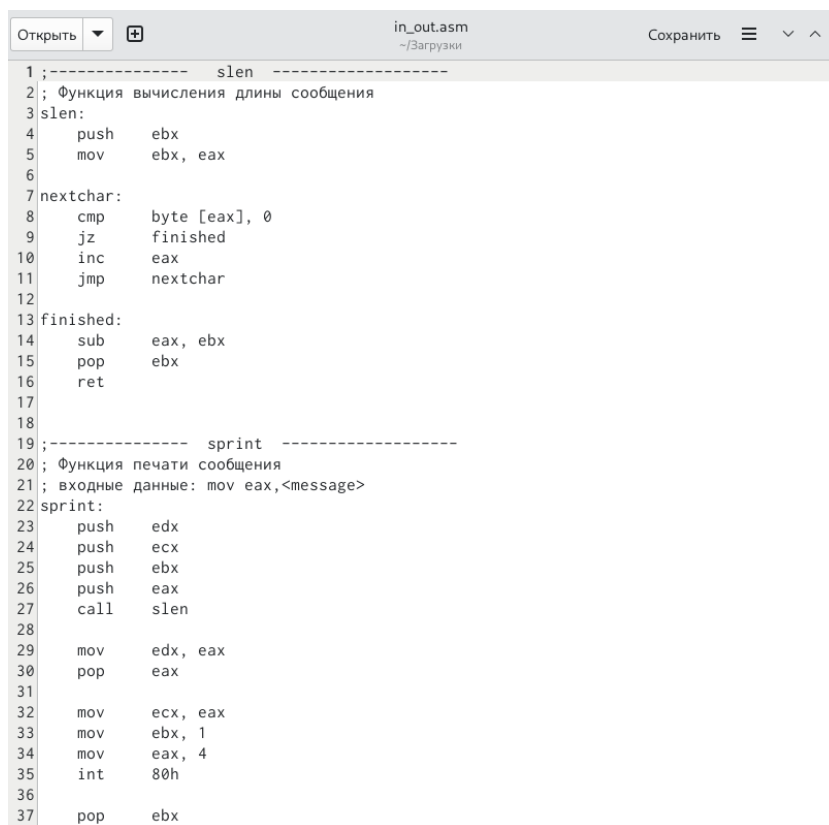
Рис. 3.8: Просмотр файла

8. Оттранслировала текст программы в объектный файл. Выполнила компо-
нировку объектного файла и запустила получившийся исполняемый файл.

```
meprokopjeva@dk8n59 ~ $ mc
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ touch lab5-1.asm
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ nasm -f elf lab5-1.asm
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1 lab5-1.o
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ ./lab5-1
введите строку
Прокопьева Марина
meprokopjeva@dk8n59 ~ $
```

Рис. 3.9: Запуск файла

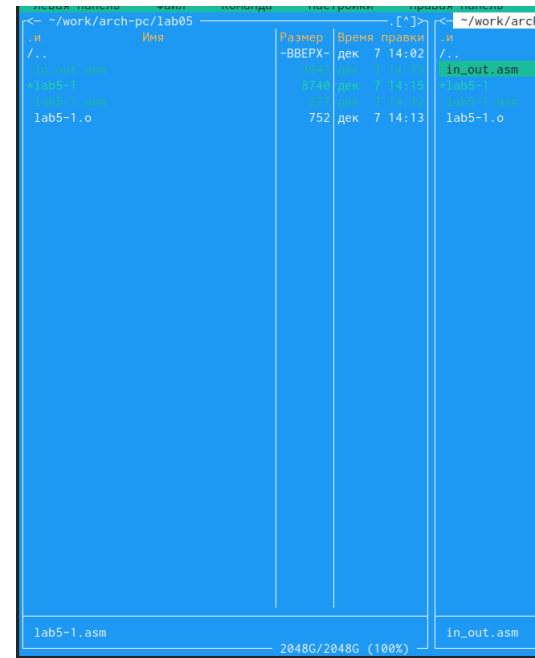
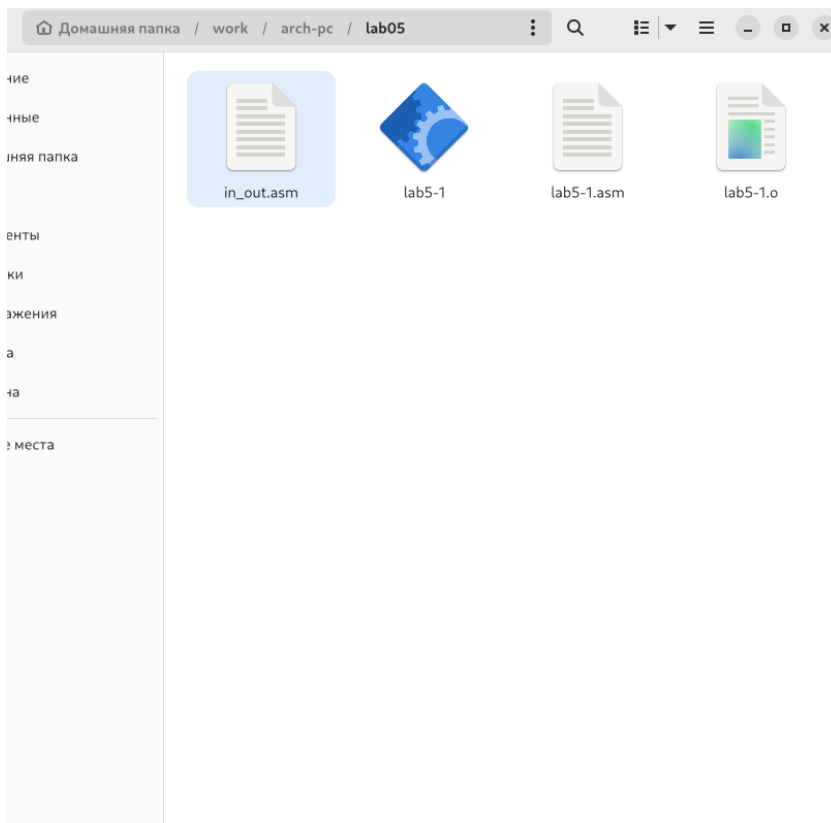
9. Скачала файл in_out.asm со страницы в туис.



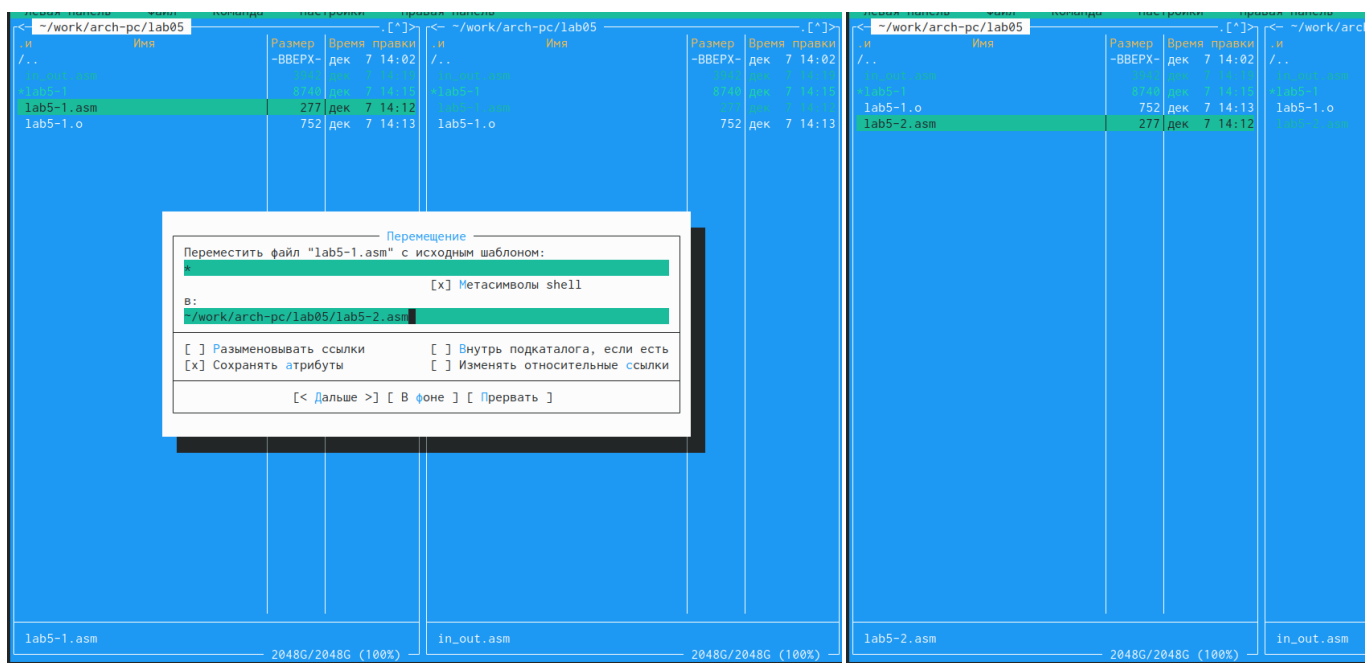
```
1 ;----- slen -----
2 ; Функция вычисления длины сообщения
3 slen:
4     push    ebx
5     mov     ebx, eax
6
7 nextchar:
8     cmp     byte [eax], 0
9     jz      finished
10    inc     eax
11    jmp     nextchar
12
13 finished:
14    sub     eax, ebx
15    pop     ebx
16    ret
17
18
19 ;----- sprint -----
20 ; Функция печати сообщения
21 ; входные данные: mov eax, <message>
22 sprint:
23     push    edx
24     push    ecx
25     push    ebx
26     push    eax
27     call    slen
28
29     mov     edx, eax
30     pop     eax
31
32     mov     ecx, eax
33     mov     ebx, 1
34     mov     eax, 4
35     int     80h
36
37     pop     ebx
```

Рис. 3.10: Скачивание файла

10. Переместила его в тот же каталог, что и файл с программой, в которой он используется.



11. С помощью функциональной клавиши F6 создала копию файла lab5-1.asm с именем lab5-2.asm.



- Исправила текст программы файла с использованием программ из внешнего файла in_out.asm. И проверила работу программы.

```
lab5-2.asm [-M--] 9 L: [ 1+18 19/ 19] *(232 / 232b) <EOF>
#include "in_out.asm"

SECTION .data
msg: DB "Загрузка программы", 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
```

```
meprokopjeva@dk8n59 ~/work/arch-pc/lab05
Введите строку:
Прокопьева Марина
meprokopjeva@dk8n59 ~/work/arch-pc/lab05
```

Самостоятельная работа

- Создала копию файла lab5-1.asm и внесла изменения, чтобы программа работала по алгоритму в задании. Проверила работу программы.

Имя	Размер	Время правки
lab5-1.asm	277	дек 7 14:41
lab5-1.o	752	дек 7 14:42
lab5-2.asm	232	дек 7 14:38
lab5-2.o	1312	дек 7 14:38
lab5-3.asm	277	дек 7 14:41

```
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ nasm -f elf64 lab5-1.asm
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ ld -m elf64 -o lab5-1.o lab5-1.o
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $ ./lab5-1
Введите строку:
Прокопьева Марина
meprokopjeva@dk8n59 ~/work/arch-pc/lab05 $
```

- Создала копию файла lab5-2.asm и внесла изменения, чтобы программа работала по алгоритму в задании. Проверила работу программы.

```

meprokojjeva@dk8n59 ~/work/arch-pc/lab05 $ nasm -f elf lab5-4.asm
meprokojjeva@dk8n59 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-4 lab5-4.o
meprokojjeva@dk8n59 ~/work/arch-pc/lab05 $ ./lab5-4
Введите строку:
Prokoreva
Prokoreva
meprokojjeva@dk8n59 ~/work/arch-pc/lab05 $

```

~/work/arch-pc/lab05		
.и	Имя	Разм
../		-BBE
	in_out.asm	3
*lab5-1		0
	lab5-1.asm	
	lab5-1.o	
*lab5-2		9
	lab5-2.asm	
	lab5-2.o	1
*lab5-3		0
	lab5-3.asm	
	lab5-3.o	
*lab5-4		9
	lab5-4.asm	
	lab5-4.o	1

4 Выводы

Приобрела практические навыки работы в Midnight Commander. Освоила инструкции языка ассемблера `mov` и `int`.

Список литературы