

Отчет по лабораторной работе №5

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

Ашуров Захид Фамил оглы

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	11
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Открытие Midnight Commander	11
4.2	Переход в каталог созданный при выполнении лабораторной работы №4	11
4.3	Создание папки lab05	11
4.4	Проверка создания папки lab05	12
4.5	Переход в созданную папку	12
4.6	Создание файла lab5-1.asm	12
4.7	Ввод текста программы	13
4.8	Сохранение текста программы	13
4.9	Просмотрим файл lab5-1.asm	14
4.10	Транслирование текста программы lab5-1 в объектный файл . . .	14
4.11	Запуск исполняемого файла	14
4.12	Скачка файла in_out.asm	15
4.13	Проверка корректности места файла	15
4.14	Создание копии файла lab5-1.asm	15
4.15	Редактирование текста программы в файле lab5-2.asm	16
4.16	Замена подпрограммы	16
4.17	Транслирование текста программы	16
4.18	Запуск файла	17

Список таблиц

1 Цель работы

Приобрести практические навыки работы в Midnight Commander. Освоить инструкцию языка ассемблера `mov` и `int`

2 Задание

Подключение внешнего файла in_out.asm

3 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.

Дополнительную информацию о Midnight Commander можно получить по команде man mc и на странице проекта.

- 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 (учетверённое слово);
- DT (define ten bytes) — определяет переменную размером в 10 байт

Директивы используются для объявления простых переменных и для объявления масси-

вов. Для определения строк принято использовать директиву DB в связи с особенностями хранения данных в оперативной памяти. Синтаксис директив определения данных следующий:

```
DB [ , ] [ , ]
```

Для объявления неиницированных данных в секции .bss используются директивы resb,

resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать за- данное количество ячеек памяти.

- 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`). 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`:

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

Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать. Использование следующих 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.

- 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` (принимать и 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.

4 Выполнение лабораторной работы

Открываем Midnight Commander (Рис. 4.1).



Рис. 4.1: Открытие Midnight Commander

Перейти в каталог созданный при выполнении работы №4 (Рис. 4.2).

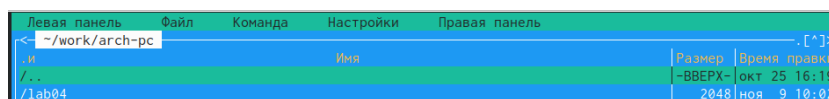


Рис. 4.2: Переход в каталог созданный при выполнении лабораторной работы №4

Создаем папку lab05 (Рис. 4.3).

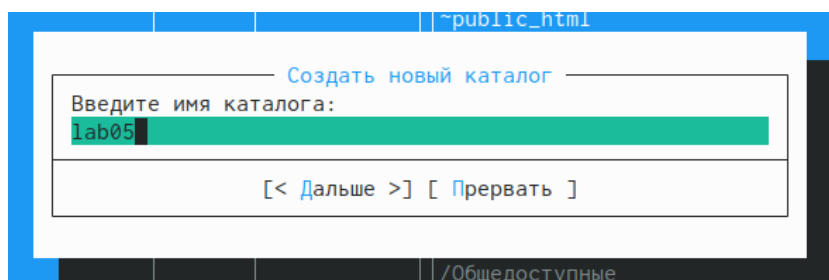


Рис. 4.3: Создание папки lab05

Удостоверимся в корректности выполнения (Рис. 4.4).

Левая панель				Правая панель			
Файл		Команда		Настройки		Правая панель	
~/work/arch-pc		[.*]>		[.*]>		[.*]>	
Имя	Размер	Время	правки	Имя	Размер	Время	правки
./	-ВВЕРХ-	окт 25	16:19	./	-ВВЕРХ-	сен 5	17:27
/lab04	2048	ноя 9	10:02	./cache	2048	окт 10	17:01
/lab05	2048	ноя 10	15:46	./config	4096	ноя 10	15:22
				./emacs.d	2048	окт 26	09:30
				./gnupg	2048	ноя 10	15:16
				./local	2048	сен 12	09:02

Рис. 4.4: Проверка создания папки lab05

Переходим в созданную папку (Рис. 4.5).

Левая панель				Правая панель			
Файл		Команда		Настройки		Правая панель	
~/work/arch-pc/lab05							
Имя	Размер	Время	правки	Имя	Размер	Время	правки
./	-ВВЕРХ-	ноя 9	10:10				

Рис. 4.5: Переход в созданную папку

Создаем файл lab5-1.asm командой touch (Рис. 4.6).

```
Совет: Вы можете просматривать файлы RPM, нажав Enter на файле RPM.
zfashurov@dk5n51 ~/work/arch-pc/lab04/lab05 $ touch lab5-1.asm
```

Рис. 4.6: Создание файла lab5-1.asm

Вводим текст программы (Рис. 4.7).

```

/afs/.dk.sci.pfu.edu.ru/home/z/f/zf-ork/arch-pc/lab04/lab05/lab5-1.asm 2179/2179 100%
;----- Объявление переменных -----
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку:',10 ; сообщение плюс
; символ перевода строки
msgLen: EQU $-msg ; Длина переменной 'msg'
SECTION .bss ; Секция не инициализированных данных
buf1: RESB 80 ; Буфер размером 80 байт

;----- Текст программы -----
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу

;----- Системный вызов 'write' -----
; После вызова инструкции 'int 80h' на экран будет
; выведено сообщение из переменной 'msg' длиной 'msgLen'
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла 1 - стандартный вывод
mov ecx,msg ; Адрес строки 'msg' в 'ecx'
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,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра

```

Рис. 4.7: Ввод текста программы

Сохраняем текст программы (Рис. 4.8).

Сохранить файл	
дтверждаете запись файла "/afs/.dk.sci.pfu.edu.ru/home/z/f/zfashurov/work/arch-pc/lab04/lab05/lab5-1.asm	
[Сохранить]	[Прервать]

Рис. 4.8: Сохранение текста программы

Посмотрим файл lab5-1.asm с помощью функциональной клавиши F3 (Рис. 4.9)

```

;-----
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;-----
;----- Объявление переменных -----
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку:',10 ; сообщение плюс
; символ перевода строки
msgLen: EQU $-msg ; Длина переменной 'msg'
SECTION .bss ; Секция не инициализированных данных
buf1: RESB 80 ; Буфер размером 80 байт
;----- Текст программы -----
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
;----- Системный вызов 'write' -----
; После вызова инструкции 'int 80h' на экран будет
; выведено сообщение из переменной 'msg' длиной 'msgLen'
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла 1 - стандартный вывод
mov ecx,msg ; Адрес строки 'msg' в 'ecx'
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 ; Дескриптор 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,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра

```

Рис. 4.9: Просмотрим файл lab5-1.asm

Оттранслируем текст программы lab5-1.asm в объектный файл (Рис. 4.10).

```

zfashurov@dk5n51 ~/work/arch-pc/lab05 $ nasm -f elf lab5-1.asm
zfashurov@dk5n51 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1 lab5-1.o

```

Рис. 4.10: Транслирование текста программы lab5-1 в объектный файл

Запустим получившийся исполняемый файл (Рис. 4.11).

```

zfashurov@dk5n51 ~/work/arch-pc/lab05 $ ./lab5-1
Введите строку:
Ашуров Захид Фамил оглы

```

Рис. 4.11: Запуск исполняемого файла

Скачиваем файл in_out.asm со страницы курса в ТУИС. (Рис. 4.12).



Рис. 4.12: Скачка файла in_out.asm

Проверяем чтоб файл in_out.asm лежал в нужно каталоге (Рис. 4.13).

< ~/work/arch-pc/lab05 .[^]>				< ...а компьютера/arch-pc/labs/lab05/report -.[^]>			
Имя	Размер	Время правки		Имя	Размер	Время правки	
./..	-ВВЕРХ-	ноя 9 10:10		./..	-ВВЕРХ-	окт 10 16:48	
in_out.asm	2048	ноя 10 16:48		/bib	2048	окт 10 16:48	

Рис. 4.13: Проверка корректности места файла

Создаем копию файла lab5-1.asm с именем lab5-2.asm (Рис. 4.14).

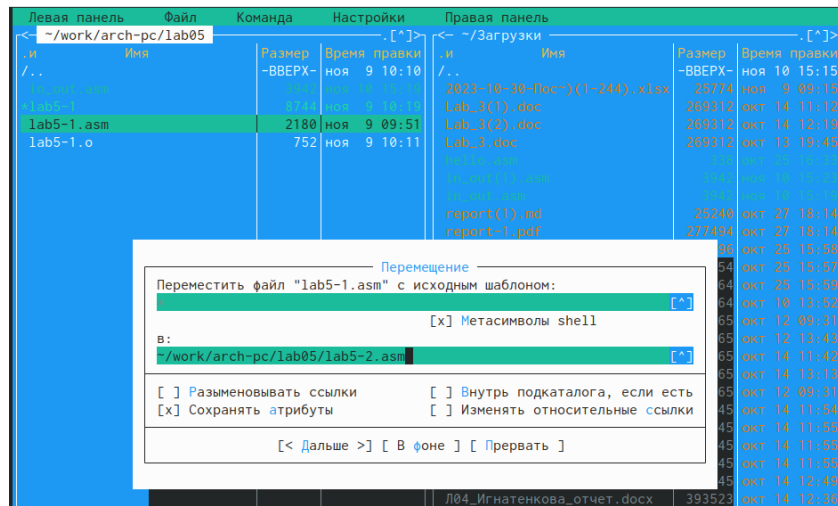


Рис. 4.14: Создание копии файла lab5-1.asm

Редактируем текст программы в файле lab5-2.asm (Рис. 4.15).

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/z/f/zfashurov/work/arch-pc/lab05/lab5-2.asm Изменён
; Программа вывода сообщения на экран и ввода строки с клавиатуры
-----
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку: ',0h ; сообщение
SECTION .bss ; Секция не инициализированных данных
buf1: RESB 80 ; Буфер размером 80 байт
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
mov eax, msg ; запись адреса выводимого сообщения в 'EAX'
call sprintf ; вызов подпрограммы печати сообщения
mov ecx, buf1 ; запись адреса переменной в 'EAX'
mov edx, 80 ; запись длины вводимого сообщения в 'EBX'
call sread ; вызов подпрограммы ввода сообщения
call quit ; вызов подпрограммы завершения
```

Рис. 4.15: Редактирование текста программы в файле lab5-2.asm

Заменяем подпрограмму sprintf на print (Рис. 4.16).

```
lab05: mc — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/z/f/zfashurov/work/arch-pc/lab05/lab5-2.asm
; Программа вывода сообщения на экран и ввода строки с клавиатуры
-----
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку: ',0h ; сообщение
SECTION .bss ; Секция не инициализированных данных
buf1: RESB 80 ; Буфер размером 80 байт
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
mov eax, msg ; запись адреса выводимого сообщения в 'EAX'
call print ; вызов подпрограммы печати сообщения
mov ecx, buf1 ; запись адреса переменной в 'EAX'
mov edx, 80 ; запись длины вводимого сообщения в 'EBX'
call sread ; вызов подпрограммы ввода сообщения
call quit ; вызов подпрограммы завершения
```

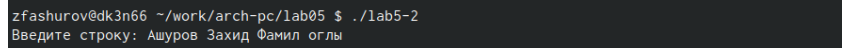
Рис. 4.16: Замена подпрограммы

Оттранслируем текст программы (Рис. 4.17).

```
zfashurov@dk3n66 ~/work/arch-pc/lab05 $ nasm -f elf lab5-2.asm
zfashurov@dk3n66 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-2 lab5-2.o
```

Рис. 4.17: Транслирование текста программы

Запускаем файл (Рис. 4.18).



```
zfashurov@dk3n66 ~/work/arch-pc/lab05 $ ./lab5-2
Введите строку: Ашуров Захид Фамил оглы
```

Рис. 4.18: Запуск файла

- В чем разница?

Разница между первым исполняемым и вторым файлом в том, что запуск первого запрашивает ввод с новой строки, а при запуске второго файла запрашивает ввод без переноса на новую строку. В этом и заключается различие между подпрограммой `sprintLF` и `sprint`.

5 Выводы

При выполнении лабораторной работы я приобрел практические навыки работы в Midnight Commander, а также освоил инструкции языка ассемблера `mov` и `int`

Список литературы