

5. Лабораторная работа №5. Основы работы с *Midnight Commander* (mc). Структура программы на языке ассемблера NASM. Системные вызовы в ОС GNU Linux


5.1. Цель работы

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

5.2. Теоретическое введение

5.2.1. 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  (рис. 5.1).

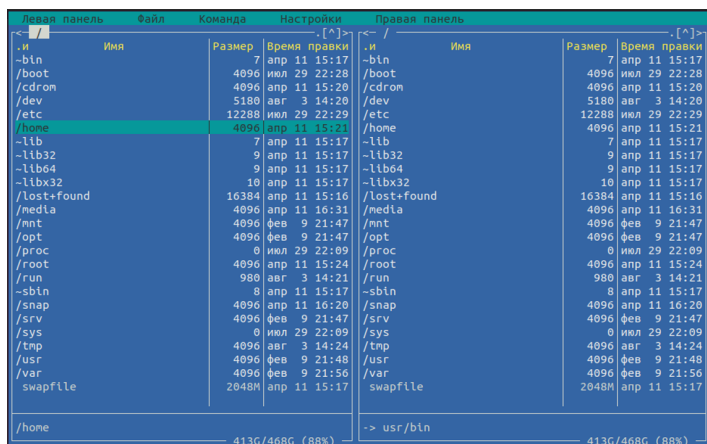


Рис. 5.1. Окно Midnight Commander

В Midnight Commander используются функциональные клавиши **F1**—**F10**, к которым привязаны часто выполняемые операции (табл. 5.1).

Таблица 5.1. Функциональные клавиши Midnight Commander

Функциональные клавиши	Выполняемое действие
F1	вызов контекстно-зависимой подсказки
F2	вызов меню, созданного пользователем

Функциональные клавиши	Выполняемое действие
F3	просмотр файла, на который указывает подсветка в активной панели
F4	вызов встроенного редактора для файла, на который указывает подсветка в активной панели
F5	копирование файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый на второй панели
F6	перенос файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый 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

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 расширений `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].

5.2.2. Структура программы на языке ассемблера NASM

Программа на языке ассемблера NASM, как правило, состоит из трёх секций: секция кода программы (`SECTION .text`), секция иницированных (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 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:

- `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 (четырёхбайтное 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:

```
<имя> DB <операнд> [, <операнд>] [, <операнд>]
```

Таблица 5.2. Примеры

Пример	Пояснение
<code>a db 10011001b</code>	определяем переменную <code>a</code> размером 1 байт с начальным значением, заданным в двоичной системе счисления (на двоичную систему счисления указывает также буква <code>b</code> (binary) в конце числа)
<code>b db ' ! '</code>	определяем переменную <code>b</code> в 1 байт, инициализируемую символом <code> ! </code>

Пример	Пояснение
c db "Hello"	определяем строку из 5 байт
d dd -345d	определяем переменную d размером 4 байта с начальным значением, заданным в десятичной системе счисления (на десятичную систему указывает буква d (decimal) в конце числа)
h dd 0f1ah	определяем переменную h размером 4 байта с начальным значением, заданным в шестнадцатеричной системе счисления (h — hexadecimal)

Для объявления неиницированных данных в секции `.bss` используются директивы `resb`, `resw`, `resd` и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти. Примеры их использования приведены в табл. 5.3.

Таблица 5.3. Директивы для объявления неиницированных данных

Директива	Назначение директивы	Аргумент	Назначение аргумента
resb	Резервирование заданного числа однобайтовых ячеек	string resb 20	По адресу с меткой string будет расположен массив из 20 однобайтовых ячеек (хранение строки символов)
resw	Резервирование заданного числа двухбайтовых ячеек (слов)	count resw 256	По адресу с меткой count будет расположен массив из 256 двухбайтовых слов
resd	Резервирование заданного числа четырёхбайтовых ячеек (двойных слов)	x resd 1	По адресу с меткой x будет расположено одно двойное слово (т.е. 4 байта для хранения большого числа)

5.2.3. Элементы программирования

5.2.3.1. 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.

Таблица 5.4. Варианты использования `mov` с разными операндами

Тип операндов	Пример	Пояснение
<code>mov <reg>,<reg></code>	<code>mov eax,ebx</code>	пересылает значение регистра <code>ebx</code> в регистр <code>eax</code>
<code>mov <reg>,<mem></code>	<code>mov cx,[eax]</code>	пересылает в регистр <code>cx</code> значение из памяти, указанной в <code>eax</code>
<code>mov <mem>,<reg></code>	<code>mov rez,ebx</code>	пересылает в переменную <code>rez</code> значение из регистра <code>ebx</code>
<code>mov <reg>,<const></code>	<code>mov eax,403045h</code>	пишет в регистр <code>eax</code> значение 403045h

Тип операндов	Пример	Пояснение
<code>mov <mem>, <const></code>	<code>mov byte[rez], 0</code>	записывает в переменную <code>rez</code> значение 0

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.

5.2.3.2. 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`.

5.2.3.3. 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.

5.3. Порядок выполнения лабораторной работы

1. Откройте *Midnight Commander*

```
user@dk4n31:~$ mc
```

2. Пользуясь клавишами ↑, ↓ и Enter перейдите в каталог `~/work/arch-rc` созданный при выполнении лабораторной работы №5 (рис. 5.2).

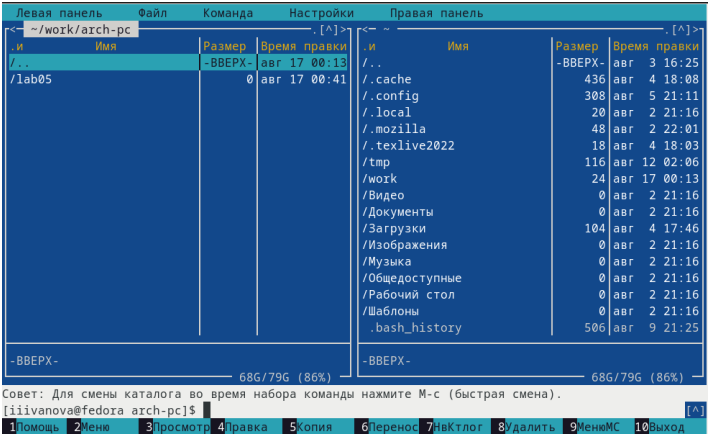


Рис. 5.2. Окно Midnight Commander. Смена текущего каталога

3. С помощью функциональной клавиши F7 создайте папку `lab06` (рис. 5.3) и перейдите в созданный каталог.

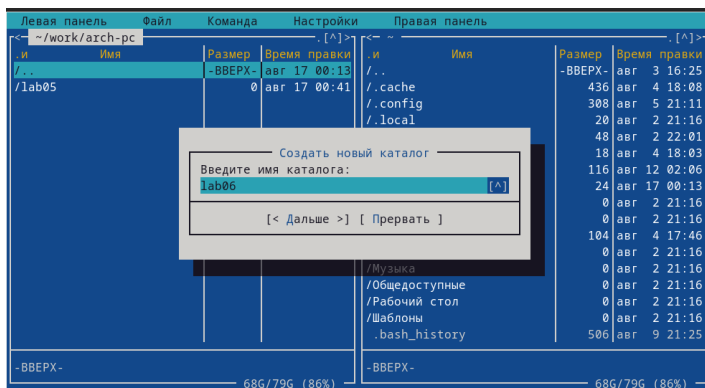


Рис. 5.3. Окно Midnight Commander. Создание каталога

4. Пользуясь строкой ввода и командой `touch` создайте файл `lab6-1.asm` (рис. 5.4).

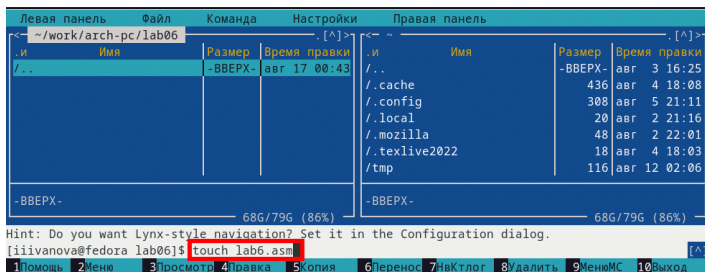


Рис. 5.4. Окно Midnight Commander. Создание файла

5. С помощью функциональной клавиши **F4** откройте файл `lab6-1.asm` для

редактирования во встроенном редакторе. Как правило в качестве встроенного редактора *Midnight Commander* используется редакторы nano (рис. 5.5) или mcedit (рис. 5.6).

```

GNU nano 6.0                                lab6.asm                                Изменён
-----
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;
-----
;----- Объявление переменных -----
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

;----- 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

^G Справка      ^O Записать   ^W Поиск      ^K Вырезать   ^I Выполнить  ^C Позиция    M-U Отмена
^X Выход        ^R ЧитФайл  ^U Замена     ^V Вставить   ^J Выровнять  ^_/ К строке  M-E Повтор
  
```

Рис. 5.5. Окно Midnight Commander. Редактор nano

```

lab6.asm      [----]  0 L:[ 1+26 27/ 46] *(1531/2733b) 0032 0x020      [*](X)
;----->
; 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'
1Помощь  2Сохранить  3Блок  4Замена  5Копия  6Пере-ить  7Поиск  8Удалить  9МенюМС  10Выход

```

Рис. 5.6. Окно Midnight Commander. Редактор mcedit

- Введите текст программы из листинга 6.1 (можно без комментариев), сохраните изменения и закройте файл.

Листинг 6.1. Программа вывода сообщения на экран и ввода строки с клавиатуры

```

;----->
; 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 иницированных 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

```

Для редактора `mcedit`: **F2** используется для сохранения изменений, **F10** для выхода из редактора.

Для редактора `nano`: **Ctrl**+**x** (выход) > **Y** (сохранить изменения) > **Enter**.

- С помощью функциональной клавиши **F3** откройте файл `lab6-1.asm` для просмотра. Убедитесь, что файл содержит текст программы.
- Оттранслируйте текст программы `lab6-1.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 lab6-1.asm
user@dk4n31:~$ ld -m elf_i386 -o lab6-1 lab6-1.o
user@dk4n31:~$ ./lab6-1
Введите строку:
Имя пользователя
user@dk4n31:~$

```

Скрыть панели *Midnight Commander*, за которыми доступен для работы командный интерпретатор оболочки, можно с помощью комбинации **Ctrl**+**o** (или через меню **Команда** > **Отключить панели**).

5.3.1. Подключение внешнего файла `in_out.asm`

Для упрощения написания программ часто встречающиеся одинаковые участки кода (такие как, например, вывод строки на экран или выход из программы) можно оформить в виде подпрограмм и сохранить в отдельные файлы, а во всех нужных местах поставить вызов нужной подпрограммы. Это позволяет сделать основную программу более удобной для написания и чтения.

NASM позволяет подключать внешние файлы с помощью директивы `%include`, которая предписывает ассемблеру заменить эту директиву содержимым файла. Подключаемые файлы также написаны на языке ассемблера. Важно отметить, что директива `%include` в тексте программы должна стоять раньше, чем встречаются вызовы подпрограмм из подключаемого файла. Для вызова подпрограммы из внешнего файла используется инструкция `call`, которая имеет следующий вид

```
call <function>
```

где `function` имя подпрограммы.

Для выполнения лабораторных работ используется файл `in_out.asm`¹, который содержит следующие подпрограммы [3]:

- `slen` – вычисление длины строки (используется в подпрограммах печати сообщения для определения количества выводимых байтов);
- `sprint` – вывод сообщения на экран, перед вызовом `sprint` в регистр `eax` необходимо записать выводимое сообщение (`mov eax, <message>`);
- `sprintLF` – работает аналогично `sprint`, но при выводе на экран добавляет к сообщению символ перевода строки;
- `sread` – ввод сообщения с клавиатуры, перед вызовом `sread` в регистр `eax` необходимо записать адрес переменной в которую введенное сообщение буд записано (`mov eax, <buffer>`), в регистр `ebx` – длину вводимой строки (`mov ebx, <N>`);

¹Файл `in_out.asm` можно скачать на странице курса в ТУИС

- `iprint` – вывод на экран чисел в формате ASCII, перед вызовом `iprint` в регистр `eax` необходимо записать выводимое число (`mov eax, <int>`);
- `iprintLF` – работает аналогично `iprint`, но при выводе на экран после числа добавляет к символ перевода строки;
- `atoi` – функция преобразует ascii-код символа в целое число и записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax, <int>`);
- `quit` – завершение программы.

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.

В одной из панелей `mc` откройте каталог с файлом `lab6-1.asm`. В другой панели каталог со скаченным файлом `in_out.asm` (для перемещения между панелями используйте `Tab`). Скопируйте файл `in_out.asm` в каталог с файлом `lab6-1.asm` с помощью функциональной клавиши `F5` (рис. 5.7).

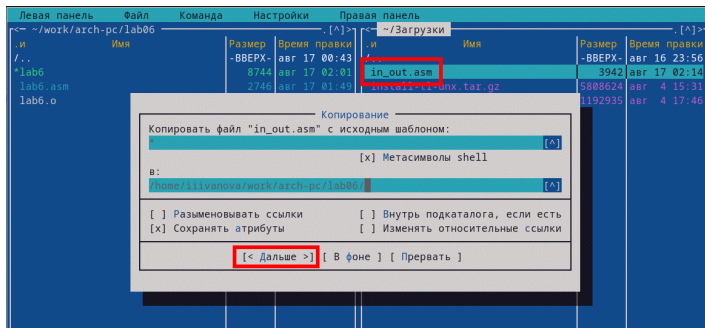


Рис. 5.7. Окно Midnight Commander. Копирование файла

11. С помощью функциональной клавиши **F6** создайте копию файла `lab6-1.asm` с именем `lab6-2.asm`. Выделите файл `lab6-1.asm`, нажмите клавишу **F6**, введите имя файла `lab6-2.asm` и нажмите клавишу **Enter** (рис. 5.8).

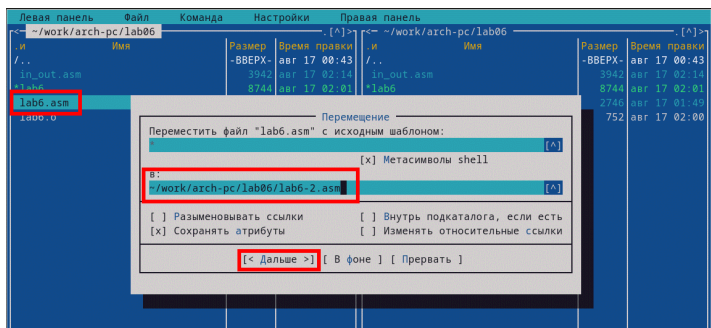


Рис. 5.8. Окно Midnight Commander. Создание копии файла

12. Исправьте текст программы в файле `lab6-2.asm` с использованием подпрограмм из внешнего файла `in_out.asm` (используйте подпрограммы `sprintLF`, `sread` и `quit`) в соответствии с листингом 6.2. Создайте исполняемый файл и проверьте его работу.

Листинг 6.2. Программа вывода сообщения на экран и ввода строки с клавиатуры с использованием файла `in_out.asm`

```

;-----
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;-----

%include 'in_out.asm' ; подключение внешнего файла

```

```

SECTION .data                ; Секция иницированных данных
msg:  DB 'Введите строку: ',0h ; сообщение

SECTION .bss                 ; Секция не иницированных 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

```

13. В файле lab6-2.asm замените подпрограмму sprintf на sprint. Создайте исполняемый файл и проверьте его работу. В чем разница?

5.4. Задание для самостоятельной работы

1. 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.

2. Получите исполняемый файл и проверьте его работу. На приглашение ввести строку введите свою фамилию.
3. 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 введённую строку на экран.

Не забудьте, подключаемый файл `in_out.asm` должен лежать в том же каталоге, что и файл с программой, в которой он используется.

4. Создайте исполняемый файл и проверьте его работу.

5.5. Содержание отчёта

Отчёт должен включать:

- Титульный лист с указанием номера лабораторной работы и ФИО студента.
- Формулировка цели работы.
- Описание результатов выполнения лабораторной работы:
 - описание выполняемого задания;
 - скриншоты (снимки экрана), фиксирующие выполнение заданий лабораторной работы;
 - комментарии и выводы по результатам выполнения заданий.
- Описание результатов выполнения заданий для самостоятельной работы:
 - описание выполняемого задания;
 - скриншоты (снимки экрана), фиксирующие выполнение заданий;
 - комментарии и выводы по результатам выполнения заданий;

- листинги написанных программ (текст программ).
- Выводы, согласованные с целью работы.

Отчёт по выполнению лабораторной работы оформляется в формате Markdown. В качестве отчёта необходимо предоставить отчёты в 3 форматах: pdf, docx и md. А также файлы с исходными текстами написанных при выполнении лабораторной работы программ (файлы *.asm). Файлы необходимо загрузить на странице курса в ТУИС в задание к соответствующей лабораторной работе и загрузить на Github.

5.6. Вопросы для самопроверки

1. Каково назначение `mc`?
2. Какие операции с файлами можно выполнить как с помощью команд `bash`, так и с помощью меню (комбинаций клавиш) `mc`? Приведите несколько примеров.
3. Какова структура программы на языке ассемблера NASM?
4. Для описания каких данных используются секции `bss` и `data` в языке ассемблера NASM?
5. Для чего используются компоненты `db`, `dw`, `dd`, `dq` и `dt` языка ассемблера NASM?
6. Какое произойдёт действие при выполнении инструкции `mov eax, esi`?
7. Для чего используется инструкция `int 80h`?