Отчёт по лабораторной работе №6. Основы работы с Midnight Commander (mc). Структура программы на языке ассемблера NASM. Системные вызовы в ОС GNU Linux

дисциплина: Архитектура компьютера

Ибатулина Дарья Эдуардовна

Содержание

# 1 Цель работы

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

# 2 Задание

1. Создать каталог для программ на языке Ассемблера для лабораторной работы №6;
2. Создать в данном каталоге текстовый файл, ввести текст программы в соответствии с указаниями к лабораторной работе;
3. Оттранслировать его в объектный файл и проверить работу файла;
4. Создать копию вышеупомянутого файла;
5. Скачать из ТУИСа внешний файл с программой, переместить его в каталог для программ по лабораторной работе №6;
6. Ввести в новый созданный файл текст программы в соответствии с указаниями к лабораторной работе, подключив внешний файл, исправив предыдущий текст;
7. Оттранслировать его в объектный файл, затем в исполняемый файл и проверить его работу;
8. Создать ещё одну копию первого файла с программой;
9. Внести изменения в текст программы в соответствии с указаниями, не используя внешний файл;
10. Создать исполняемый файл и проверить его работу;
11. Создать ещё одну копию второго файла с программой;
12. Внести изменения в текст программы в соответствии с указаниями, используя внешний файл;
13. Создать исполняемый файл и проверить его работу.

# 3 Теоретическое введение

## 3.1 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 того, чтобы начать работу с программой, необходимо написать в терминале команду mc и нажать Ener.

Например, в табл. 1 приведены функциональные клавиши mc и их использование.

Таблица 1: Функциональные клавиши Midnight Commander

| Клавиша | Назначение |
| --- | --- |
| f1 | вызов контекстно-зависимой подсказки |
| f2 | вызов меню, созданного пользователем |
| 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 |

## 3.2 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 размеров 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.

## 3.3 Элементы программирования

### 3.3.1 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. Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать.

### 3.3.2 Описание инструкции int

Инструкция языка ассемблера intпредназначена 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).

### 3.3.3 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 **еах**\* значение *1*, а в регистр ***ebx*** код завершения *0*.

### 3.3.4 Подключение внешнего файла

Для упрощения написания программ часто встречающиеся одинаковые участки кода (такие как, например, вывод строки на экран или выход их 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.

# 4 Выполнение лабораторной работы

1. Сначала открываем терминал, вводим команду mc и видим открывшееся окно MidnightCommander (рис. 1):

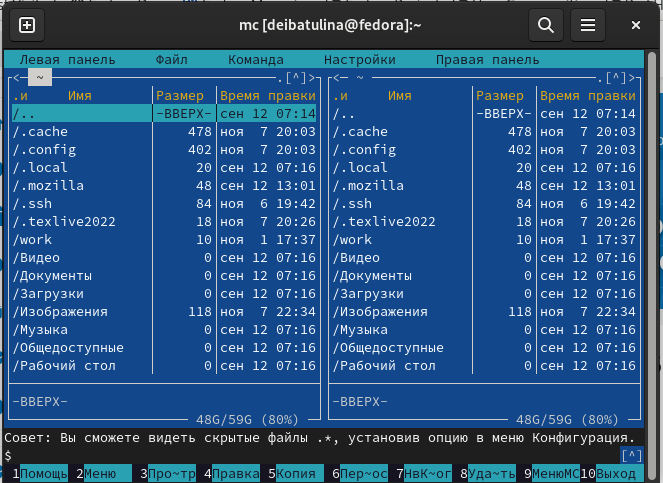


Рис. 1: Окно MidnightCommander

1. C помощью функциональной клавиши F7 создаём в каталоге для написания программ на Ассемблере папку для лабораторной работы №6 (рис. 2):

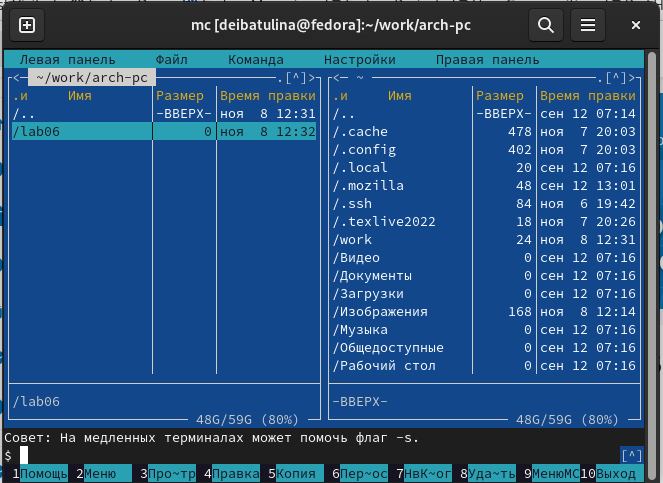


Рис. 2: Создание каталога lab06

1. Пользуясь строкой ввода, создаём файл lab6-1.asm (рис. 3):

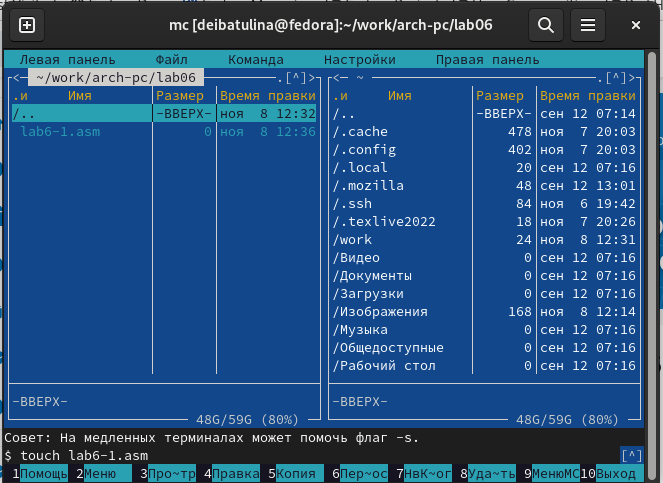


Рис. 3: Создание файла lab6-1.asm

1. С помощью клавиши F4 открываем данный файл для редактирования (рис. 4). По умолчанию файл открывается в текстовом редакторе mcedit:

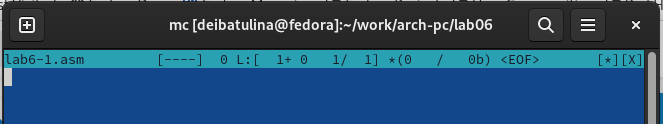


Рис. 4: Открытие файла lab6-1.asm для редактирования

1. Вводим в файл текст программы из листинга, как указано в лабораторной работе (рис. 5). Данная программа выводит сообщение на экран, ждёт ввода строки от пользователя:

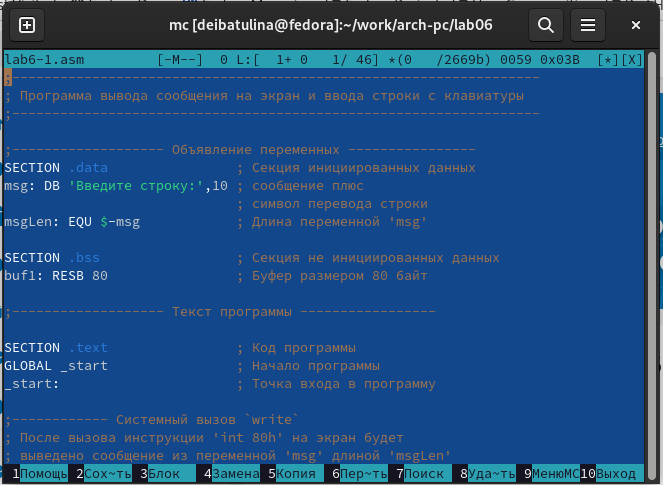


Рис. 5: Ввод текста программы в файл

1. Сохраняем изменения в файле с помощью клавиши F2 (рис. 6):

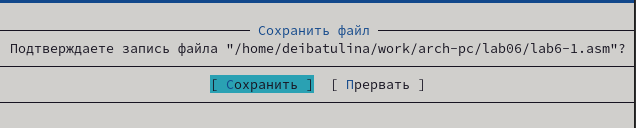


Рис. 6: Сохранение изменений в тексте программы

1. Открываем файл для просмотра с помощью клавиши F3, убеждаемся в том, что данный файл содержит текст программы (рис. 7):

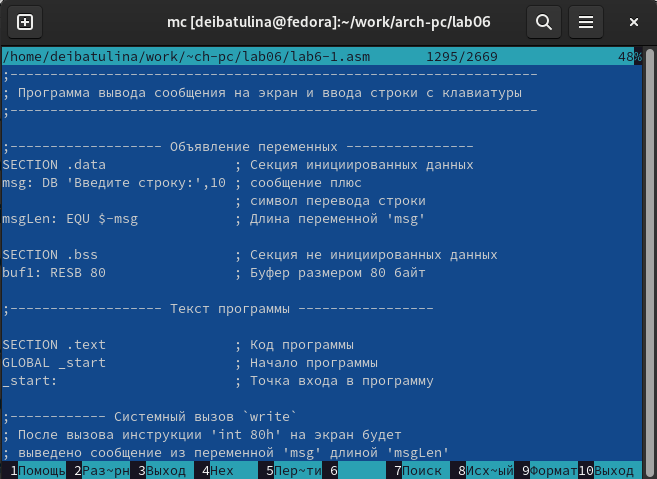


Рис. 7: Открытие файла для просмотра (чтения)

1. Транслируем файл в объектный, затем в исполняемый, проверяем корректность работы программы (рис. 8):

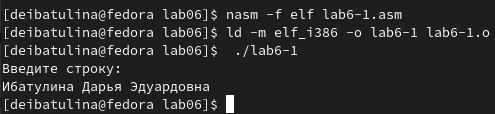


Рис. 8: Трансляция файла в исполняемый, проверка корректности работы программы

Заметим, что программа успешно работает.

1. Для дальнейшей работы необходимо подключить внешний файл. Программа также должна выводить на экран сообщение и ждать от пользователя ввода строки с клавиатуры. При этом часть инструкций чодержится во внешнем файле. Для этого скачиваем внешний файл со страницы курса в ТУИС (рис. 9):



Рис. 9: Скачивание внешнего файла со страницы курса в ТУИС

1. Для работы со внешним файлом он должен лежать в том же каталоге, что и файл с кодом программы. Перемещаем его в каталог lab06, пользуясь функциональной клавишей F5 (рис. 10):

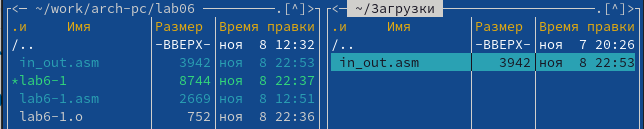


Рис. 10: Перемещение внешнего файла в каталог с программами

1. С помощью клавиши F5 создаём копию файла lsb6-1.asm, называем его lab6-2.asm (рис. 11):

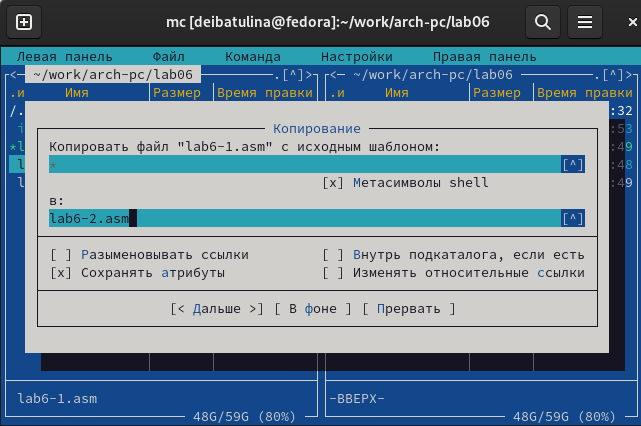


Рис. 11: Копирование файла lab6-1.asm

1. Далее вводим в этот файл текст программы, указанный в лабораторной работе (рис. 12), создаём исполняемый файл и проверяем его работу (рис. 13):

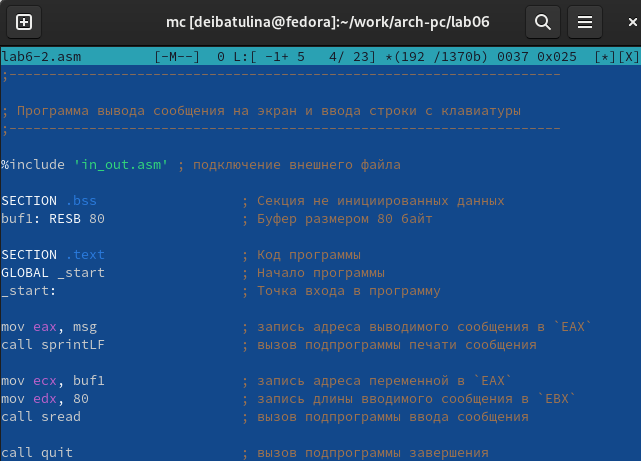


Рис. 12: Создание исполняемого файла с исправленным текстом программы

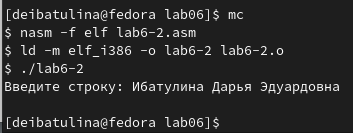


Рис. 13: Проверка работы исполняемого файла

1. Вносим изменения в файл lab6-2.asm, заменяя команду ‘sprintLF’ на ‘sprint’, создаём исполняемый файл и проверяем его работу (рис. 14):

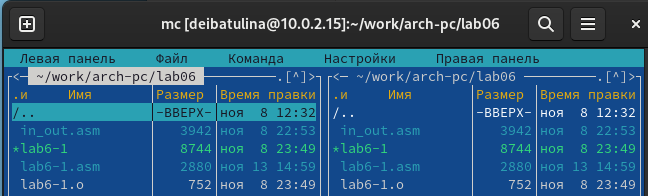


Рис. 14: Проверка работы исполняемого файла после замены одной команды

Мы видим, что различие есть: оно заключается в том, что в первом случае после ввода строки с клавиатуры пользователем курсор переносился на новую строку, а в исправленном файле - нет. Всё дело в том, что команда ‘sprintLF’ добавляет к сообщению символ перевода строки.

## 4.1 Выполнение заданий для самостоятельной работы

1. Напишем программу для того, чтобы она выводила сообщение на экран, требовала от пользователя ввода строки с клавиатуры, а затем выводила эту сроку на экран. Создаём копию файла lab6-1.asm с помощью клавиши F5, называем его lab6-1\_copy.asm (рис. 15):

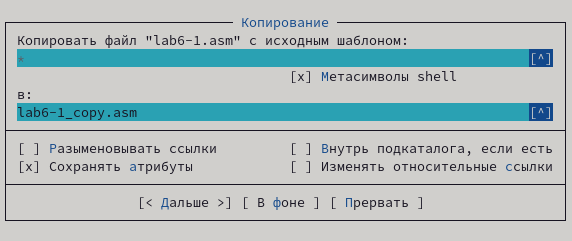


Рис. 15: Копирование файла lab6-1.asm, название его lab6-1\_copy.asm

1. Нажимаем F4, вносим в него текст программы:

**Листинг 1. Программа вывода сообщения на экран, ввода строки с клавиатуры и вывода данной строки на экран**

;------------------------------------------------------------------------------------  
; Программа вывода сообщения на экран, ввода строки с клавиатуры и её вывода на экран'  
;-------------------------------------------------------------------------------------  
  
;------------------- Объявление переменных ----------------  
SECTION .data ; Секция инициированных данных  
msg: DB 'Введите строку:',10 ; сообщение плюс  
; символ перевода строки  
  
msgLen: EQU $-msg ; Длина переменной 'msg'  
SECTION .bss ; Секция не инициированных данных  
buf1: RESB 80 ; Буфер размером 80 байт  
  
;------------------- Текст программы -----------------  
SECTION .text ; Код программы  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа в программу  
  
;------------ Cистемный 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  
  
;------------ Cистемный 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,buf1 ; Адрес строки 'buf1' в 'ecx'  
mov edx,buf1 ; Размер строки 'buf1' в 'edx'  
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

**!Примечание!** Чтобы вывести введённую пользователем строку на экран, после системного вызова *read* добавляем ещё один блок кода системный вызов *write*, меняем msg и msgLen на buf1, чтобы записать в регистры адрес и размер строки, введённой пользователем с клавиатуры, соответственно.

1. Создаём исполняемый файл, проверяем работу исполняемого файла (рис. 16):

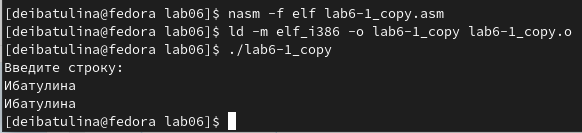


Рис. 16: Создание исполняемого файла lab6-1\_copy.asm, проверка его работы

1. Теперь нужно создать аналогичную программу, но с подключением внешнего файла *in\_out.asm*. Создаём копию файла lab6-2.asm с помощью клавиши F5, называем его lab6-2\_copy.asm (рис. 17):

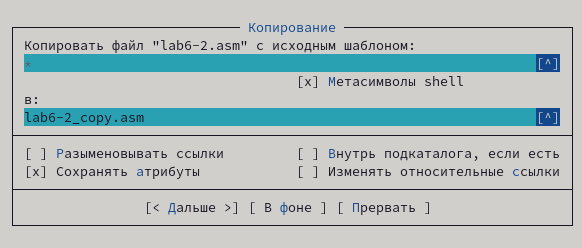


Рис. 17: Копирование файла lab6-2.asm, название его lab6-2\_copy.asm

1. Нажимаем F4, вносим в него текст программы:

**Листинг 2. Программа вывода сообщения на экран, ввода строки с клавиатуры и вывода данной строки на экран (с подключением внешнего файла)**

;--------------------------------------------------------------------  
; Программа вывода сообщения на экран и ввода строки с клавиатуры  
;---------------------------------------------------------------------  
  
%include 'in\_out.asm' ; подключение внешнего файла  
  
SECTION .data ; Секция инициированных данных  
msg: DB 'Введите строку: ',0h ; сообщение  
SECTION .bss ; Секция не инициированных 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 sprint ; 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  
mov eax,4 ; системный вызов для записи (sys\_write)  
mov ebx,1 ; описатель файла '1' - стандартный вывод  
mov ecx,buf1 ; адрес строки buf1 в ecx  
int 80h ; вызов ядра  
call quit ; вызов подпрограммы завершения

**!Примечание!** Чтобы вывести введённую пользователем строку на экран, после *call sread* добавляем ещё один блок кода системный вызов *sys\_write*, отвечающий за вызов вывода введённой строки, а также в регистр ecx записываем адрес строки buf1.

1. Создаём исполняемый файл, проверяем работу исполняемого файла (рис. 18):

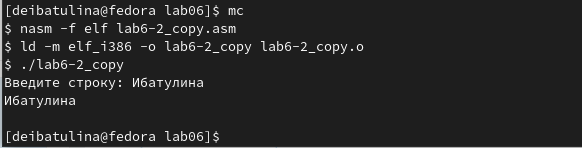


Рис. 18: Создание исполняемого файла lab6-2\_copy.asm, проверка его работы

# 5 Выводы

В процессе выполнениия лабораторной работы я приобрела практические навыки работы в Midnight Commander, а также освоила инструкции языка ассемблера mov и int.

# Список литературы

1. Лабораторная работа №6. Руководство.