

# **Лабораторная работа №9**

**Понятие подпрограммы.Отладчик GDB.**

Виеру Женифер

# Содержание

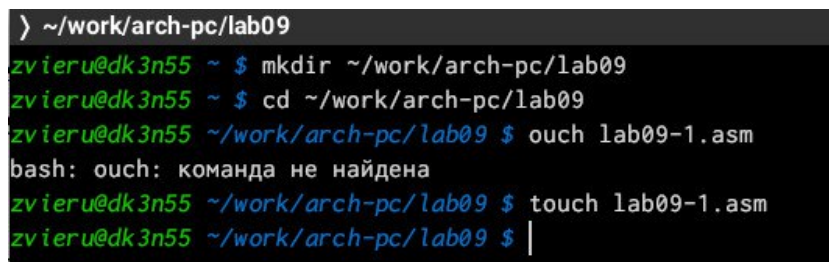
<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>4</b>
<b>3</b>	<b>Выводы</b>	<b>14</b>

# 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки..

## 2 Выполнение лабораторной работы

Создала каталог для программ лабораторной работы № 9, перешла в него и создала файл lab09-1.asm. Потом открыла его с помощью команды `ms` и горячей клавиши F4 и написала в нем следующий текст(рис. 2.1).



```
> ~/work/arch-pc/lab09
zvieru@dk3n55 ~ $ mkdir ~/work/arch-pc/lab09
zvieru@dk3n55 ~ $ cd ~/work/arch-pc/lab09
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ouch lab09-1.asm
bash: ouch: команда не найдена
zvieru@dk3n55 ~/work/arch-pc/lab09 $ touch lab09-1.asm
zvieru@dk3n55 ~/work/arch-pc/lab09 $ |
```

Рис. 2.1: Создание lab09-1.asm

Потом написала программу программу вычисления арифметического выражения  $f(x) = 2x + 7$  с помощью подпрограммы `_calcul`. (рис. 2.2).

```

lab09-1.asm      [----] 3 L:[ 1+ 4 5/ 30] *(91 / 419b) 0084 0x054 [*][X
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret

```

Рис. 2.2: Текст программы

Создала исполняемый файл запустила его и проверила его работу (рис. 2.3).

```

zvieru@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ./lab09.1
bash: ./lab09.1: Нет такого файла или каталога
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
2x+7=13

```

Рис. 2.3: Запуск lab09-1.asm

Изменила текст программы так, чтобы она вычисляла  $f(g(x))$  (рис. 2.4).

```

lab09-1.asm      [-M--] 21 L:[ 1+ 0 1/ 57] *(21 / 5
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
prim1: DB 'f(x) = 2x+7',0
prim2: DB 'g(x) = 3x-1',0
result: DB 'f(g(x))= ',0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,prim1
call sprintLF

mov eax,prim2
call sprintLF

mov eax,msg
call sprint

mov ecx,x
mov edx,80
call sread

mov eax,x
call atoi

call _calcul

mov eax,result
call sprint
mov eax,[res]
call iprintLF

call quit

```

Рис. 2.4: Текст программы

Создала исполняемый файл запустила его и проверила его работу (рис. 2.5).

```

zvieru@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 5
f(g(x))= 35
zvieru@dk3n55 ~/work/arch-pc/lab09 $ |

```

Рис. 2.5: Запуск lab09-1.asm

Создала файл lab09-2.asm с текстом программы вывода сообщения Hello world!(рис. 2.6).

```

lab09-2.asm [----] 8 L:[ 1+27 28/ 28] *(300 / 300b) <EOF>
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2

SECTION .text
global _start
_start:

mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len

int 0x80

mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len

int 0x80

mov eax, 1
mov ebx, 0

int 0x80

```

Рис. 2.6: Текст программы

Создала исполняемый файл и загрузила его файл в отладчик gdb и запустила его(рис. 2.7).

```

zvieru@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
zvieru@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
zvieru@dk3n55 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: ~/work/arch-pc/lab09/lab09-2
Starting program: /afs/.dk.sci.pfu.edu.ru/home/z/v/zvieru/work/arch-pc/lab09/lab09-2
Starting program: ~/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 6386) exited normally]
(gdb)

```

Рис. 2.7: Запуск lab09-1.asm

Для более подробного анализа программы установила брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустила её(рис. 2.8).

```

(gdb) run
Starting program: ~/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:12
12 mov eax, 4
No source file named _start
Breakpoint 1 at 0x8049000.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (_start)
Breakpoint 1 at 0x8049000: file lab09-2.asm , line 12.
(gdb) run
Starting program: ~/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:12
12 mov eax, 4) pending.
(gdb) █

```

Рис. 2.8: Запуск

Посмотрела дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`(рис. 2.9).



```

(gdb) disassemble _start
Dump of assembler code for function _start:
0x08049000 <+0>:    mov     $0x4,%eax
0x08049005 <+5>:    mov     $0x1,%ebx
0x0804900a <+10>:   mov     $0x804a000,%ecx
0x0804900f <+15>:   mov     $0x8,%edx
0x08049014 <+20>:   int     $0x80
0x08049016 <+22>:   mov     $0x4,%eax
0x0804901b <+27>:   mov     $0x1,%ebx
0x08049020 <+32>:   mov     $0x804a008,%ecx
0x08049025 <+37>:   mov     $0x7,%edx
0x0804902a <+42>:   int     $0x80
0x0804902c <+44>:   mov     $0x1,%eax
0x08049031 <+49>:   mov     $0x0,%ebx
0x08049036 <+54>:   int     $0x80
End of assembler dump.

```

Рис. 2.9: disassemble \_start

Переключилась на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel(рис. 2.10).

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
0x08049000 <+0>:    mov     eax,0x4
0x08049005 <+5>:    mov     ebx,0x1
0x0804900a <+10>:   mov     ecx,0x804a000
0x0804900f <+15>:   mov     edx,0x8
0x08049014 <+20>:   int     0x80
0x08049016 <+22>:   mov     eax,0x4
0x0804901b <+27>:   mov     ebx,0x1
0x08049020 <+32>:   mov     ecx,0x804a008
0x08049025 <+37>:   mov     edx,0x7
0x0804902a <+42>:   int     0x80
0x0804902c <+44>:   mov     eax,0x1
0x08049031 <+49>:   mov     ebx,0x0
0x08049036 <+54>:   int     0x80
End of assembler dump.

```

Рис. 2.10: Текст программы

Включила режим псевдографики для более удобного анализа программы (рис. 2.11).

```
gdb lab09-2

[ Register Values Unavailable ]

0x8049000 <_start>    mov    eax,0x4
0x8049005 <_start+5>    mov    ebx,0x1
0x804900a <_start+10>   mov    ecx,0x804a000
0x804900f <_start+15>   mov    edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov    eax,0x4
0x804901b <_start+27>   mov    ebx,0x1
0x8049020 <_start+32>   mov    ecx,0x804a008
0x8049025 <_start+37>   mov    edx,0x7
0x804902a <_start+42>   int     0x80
0x804902c <_start+44>   mov    eax,0x1
0x8049031 <_start+49>   mov    ebx,0x0
0x8049036 <_start+54>   int     0x80

exec No process in: L?? PC: ??
(gdb) layout regs
(gdb) █
```

Рис. 2.11: Запуск layout regs

Проверила установления точки останова по имени метки (`_start`) с помощью команды `info breakpoints` (рис. 2.12).

```
0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x80
0x804902c <_start+44>   mov     eax,0x1
0x8049031 <_start+49>   mov     ebx,0x0
0x8049036 <_start+54>   int     0x80

exec No process in: L?? PC: ??
(gdb) layout regs
(gdb) info breakpoints
Num   Type      Disp Enb Address      What
1     breakpoint keep  y   <PENDING>   _start
Breakpoint 1 at 0x8049000: file lab09-2.asm , line 12.
(gdb) run
Starting program: ~/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:12
12 mov eax, 4
(gdb) █
```

Рис. 2.12: Текст программы

С помощью команды `info` посмотрела значение переменной `msg1` и `msg2`(рис. 2.13).

```
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 2.13: Значение переменной `msg1` и `msg2`

С помощью команды `set` изменила значение переменной `msg1` (рис. 2.14).

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
```

Рис. 2.14: Изменение переменной `msg1`

С помощью команды `set` изменила значение переменной `msg2` (рис. 2.15).

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb)
```

Рис. 2.15: Изменение переменной msg2

Потом я вывела значение регистров есх еах (рис. 2.16).

```
(gdb) p/f $msg1
$1 = void
(gdb) p/s $eax
$2 = 1
(gdb) p/t $eax
$3 = 1
(gdb) p/c $ecx
$4 = 8 '\b'
(gdb) p/x $ecx
$5 = 0x804a008
```

Рис. 2.16: Значение регистров есх еах

Я изменила значение регистра ебх. Команда выводит два разных значения

так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные. (рис. 2.17).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50

(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2

(gdb)
```

Рис. 2.17: Значение регистра ebx

Я посмотрела все позиции стека. По первому адрему хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того (рис. 2.18).

```
(gdb) x/x $esp
0xfffffc300: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xfffffc57d: "/afs/.dk.sci.pfu.edu.ru/home/z/v/zvieru/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xfffffc5c0: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xfffffc5d2: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xfffffc5e3: "2"
(gdb) x/s *(void**)($esp + 20)
0xfffffc5e5: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 2.18: Все позиции стека

## **3 Выводы**

Выполнив данную лабораторную работу я приобрела навыки написания программ использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.