

# **Лабораторная работа №9**

**Понятие подпрограммы.Отладчик GDB.**

Бурыкина Софья Дмитриевна

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
4	Самостоятельная работа	14
5	Выводы	16

## Список иллюстраций

3.1	Создала исполняемый файл и проверила его работу . . . . .	7
3.2	Результат программы . . . . .	7
3.3	Изменённая программа . . . . .	8
3.4	Проверила его работу, запустив её в оболочке GDB . . . . .	8
3.5	Запуск программы . . . . .	9
3.6	Переключилась на отображение команд с Intel'овским синтаксисом, .	9
3.7	Включила режим псевдографики для более удобного анализа программы . . . . .	10
3.8	Установка точки останова по адресу инструкции . . . . .	11
3.9	Значение . . . . .	11
3.10	Значение . . . . .	11
3.11	Вывод команды . . . . .	12
3.12	Запуск программы с аргументом 1 2 3, установка брейкпоинта . .	13
3.13	Позиция стека . . . . .	13
4.1	Результат работы . . . . .	14
4.2	Запуск файла . . . . .	14
4.3	Регистр ebx=5, а должен быть eax=5 . . . . .	15
4.4	Ошибка в сложение . . . . .	15
4.5	Проверка программы . . . . .	15

## **Список таблиц**

# **1 Цель работы**

Приобретение навыков написания программ с использованием подпрограмм.

## 2 Теоретическое введение

Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Более подробно об Unix см. в [1–6].

### 3 Выполнение лабораторной работы

Ввела программу с листинга 9.1(рис. 3.1).

```
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
2x+7=13
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создала исполняемый файл и проверила его работу

Изменила текст программы добавив подпрограмму \_subcalcul (рис. 3.2).(рис. 3.3).

```
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
Результат = 11
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $
```

Рис. 3.2: Результат программы

```
lab09-1.asm [-M--] 13 L: [ 1+25 26/ 43] *(511 / 802b) 0010 0x00A
#include <unistd.h>
SECTION .data
msg: DB "Введите x: ",0
result: DB "Результат = ",0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _subcalcul
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintf
call quit
;---xxX-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
mov [res], eax
ret
```

Рис. 3.3: Изменённая программа

Создала файл lab09-2.asm с текстом программы из Листинга 9.2.(рис. ??).(рис.

3.4).

```
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ touch lab09-2.asm
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ mc
```

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/d/sdburikhkina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3656) exited normally]
(gdb)
```

Рис. 3.4: Проверила его работу, запустив её в оболочке GDB

Установила брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустила её.(рис. 3.5 ). (рис. ?? ). (рис. 3.6 ). (рис. 3.7).



```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/d/sdburikhina/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
```

Рис. 3.5: Запуск программы

```
(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.
```

```
(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. 3.6: Переключилась на отображение команд с Intel'овским синтаксисом,

```

чшил правка вид Закладки модули Настройка справка
Новая вкладка Разделить окно Копировать Вставить Найти

Register group: general
eax 0x0 0 ecx 0x0 0
edx 0x0 0 ebx 0x0 0
esp 0xffffc300 0xffffc300 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x8049000 0x8049000 <_start> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B> 0x8049000 <_start> mov eax,0x1
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

native process 3837 In: _start L9 PC: 0x8049000
(gdb) disassemble _start
(gdb) layout asm
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049000 lab09-2.asm:9
3 breakpoint keep y 0x08049000 lab09-2.asm:9
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/s/d/sdburikhina/work/arch-
pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
(gdb)

```

Рис. 3.7: Включила режим псевдографики для более удобного анализа программ

Посмотрела информацию о всех установленных точках (рис. 3.8).

```

Register group: general
eax 0x0 0 ecx 0x0 0
edx 0x0 0 ebx 0x0 0
esp 0xffffc300 0xffffc300 ebp 0x0 0
esi 0x0 0 edi 0x0 0
eip 0x8049000 0x8049000 <_start> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+> 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
b+ 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80

native process 3837 In: _start L9 PC: 0x8049000
2 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
3 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 4 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
3 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
4 breakpoint keep y 0x8049031 lab09-2.asm:20
(gdb)

```

Рис. 3.8: Установка точки останова по адресу инструкции

Посмотрим значение переменной msg1 по имени (рис. 3.9). (рис. 3.10).

```

(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 3.9: Значение

```

(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sd 0x804a008
0x804a008 <msg2>: 119
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhlllo, "

```

Рис. 3.10: Значение

С помощью команды set изменила значение регистра ebx. (рис. 3.11).

```
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x2      2
esp      0xffffc300 0xffffc300  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags  0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

B> 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
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80

native process 3837 In: _start L9 PC: 0x8049000
0x804a000 <msg1>: "Hello, "
(gdb) x/isd 0x804a008
0x804a008 <msg2>: 119
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/lsb &msg1
0x804a000 <msg1>: "hhlllo, "
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) <
```

Рис. 3.11: Вывод команды

Я копирую файл lab8-2.asm в папку с лабораторной номер 9 и называю его lab09-3.asm, создаю исполняемый файл и запускаю его через gdb. Эта программа должна находить произведение аргументов. Также ставлю точку останова на месте start (рис. 3.12).

```

sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ gdb --args lab09-3 1 2 3
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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 lab09-3...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/d/sdburikhina/work/arch-pc/lab09/lab09-3 1 2 3
1
2
3
[Inferior 1 (process 5007) exited normally]
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.o, line 8.

```

Рис. 3.12: Запуск программы с аргументом 1 2 3, установка брейкпоина

Рассматриваю позиции стека, так как у меня всего три аргумента на шаге (+20) выдаёт ошибку (рис. 3.13).

```

(gdb) x/x $esp
0xffffc2f0: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffc581: "/afs/.dk.sci.pfu.edu.ru/home/s/d/sdburikhina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc5c9: "1"
(gdb) x/s *(void**)(esp + 12)
0xffffc5cb: "2"
(gdb) x/s *(void**)(esp + 16)
0xffffc5cd: "3"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.13: Позиция стека

## 4 Самостоятельная работа

Создаю файл lab09-3.asm для выполнения первого задания из самостоятельной работы. Ввожу текст листинга 9.1 для удобства. Пишу программу согласно 7 варианту лабораторной работы номер 8 (рис. 4.1).

```
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf lab09-3.asm
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
sdburikhina@dk3n62 ~/work/arch-pc/lab09 $ ./lab09-3
Введите x: 1
3(x+2)=9
```

Рис. 4.1: Результат работы

Создаю файл lab09-4.asm для выполнения второго пункта лабораторной работы, ввожу туда текст листинга 9.3, сохраняю, создаю исполняемый файл и запускаю его. Убеждаюсь, что результат неверный. Ответ должен быть 25 (рис. 4.2).

ecx	0x2	0
ebx	0x5	5
ebp	0x0	0x0
edi	0x0	0
eflags	0x206	[ PF IF ]
ss	0x2b	43
es	0x2b	43
gs	0x0	0

Рис. 4.2: Запуск файла

Запускаю отладчик, смотрю как изменяется регистры eax,ebx,ecx пошагово с

помощью команды `si`. Замечаю что на третьем шаге регистр `ebx` имеет значение 5, а на четвёртом шаге `ecx` и `eax` перемножаются, что даёт неверный результат. Следовательно изменяю программу так, чтобы результат сложения записывался и `eax`(рис. 4.3).

<code>ecx</code>	<code>0x2</code>	<code>0</code>
<code>ebx</code>	<code>0x5</code>	<code>5</code>
<code>ebp</code>	<code>0x0</code>	<code>0x0</code>
<code>edi</code>	<code>0x0</code>	<code>0</code>
<code>eflags</code>	<code>0x206</code>	<code>[ PF IF ]</code>
<code>ss</code>	<code>0x2b</code>	<code>43</code>
<code>es</code>	<code>0x2b</code>	<code>43</code>
<code>gs</code>	<code>0x0</code>	<code>0</code>

Рис. 4.3: Регистр `ebx`=5, а должен быть `eax`=5

Снова неверный результат, нахожу ошибку в именовании регистрах, вижу ошибку в сложение и меняю это (рис. 4.4).

Register group: general					
<code>eax</code>	<code>0x19</code>	<code>25</code>	<code>ecx</code>	<code>0x4</code>	<code>4</code>
<code>edx</code>	<code>0x0</code>	<code>0</code>	<code>ebx</code>	<code>0x3</code>	<code>3</code>
<code>esp</code>	<code>0xffffc300</code>	<code>0xffffc300</code>	<code>ebp</code>	<code>0x0</code>	<code>0x0</code>
<code>esi</code>	<code>0x0</code>	<code>0</code>	<code>edi</code>	<code>0x0</code>	<code>0</code>
<code>eip</code>	<code>0x80490fe</code>	<code>0x80490fe &lt;_start&gt;</code>	<code>eflags</code>	<code>0x202</code>	<code>[ IF ]</code>
<code>cs</code>	<code>0x23</code>	<code>35</code>	<code>ss</code>	<code>0x2b</code>	<code>43</code>
<code>ds</code>	<code>0x2b</code>	<code>43</code>	<code>es</code>	<code>0x2b</code>	<code>43</code>
<code>fs</code>	<code>0x0</code>	<code>0</code>	<code>gs</code>	<code>0x0</code>	<code>0</code>

Рис. 4.4: Ошибка в сложение

Создаю исполняемый файл и запускаю его (рис. 4.5).

```
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ ./lab09-4
Результат: 25
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ mc
sdburikhkina@dk3n62 ~/work/arch-pc/lab09 $ mc
```

Рис. 4.5: Проверка программы

## 5 Выводы

Приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями. # Список литературы{unnumbered}

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016. URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.