Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Ким Денис Вячеславович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Приобрести навыки написания программ с использованием подпрограмм и познакомиться с методами отладки при помощи GDB и его основными возможностями.

2 Задание

В ходе данной работы мне предстоит познакомиться с понятием отладки, методами отладки, основными возможностями отладчика GDB. Также я научусь пользоваться отладкой программ с помощью данного отладчика.

3 Теоретическое введение

Здесь описываются теоретические аспекты, связанные с выполнением работы.

Например, в табл. 3.1 приведено краткое описание стандартных каталогов Unix.

Таблица 3.1: Описание некоторых каталогов файловой системы GNU Linux

Имя каталога	Описание каталога
/	Корневая директория, содержащая всю файловую
/bin	Основные системные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям
/etc	Общесистемные конфигурационные файлы и файлы конфигурации установленных программ
/home	Содержит домашние директории пользователей, которые, в свою очередь, содержат персональные настройки и данные пользователя
/media	Точки монтирования для сменных носителей
/root	Домашняя директория пользователя root
/tmp	Временные файлы
/usr	Вторичная иерархия для данных пользователя

Более подробно про Unix см. в [1–4].

4 Выполнение лабораторной работы

Создаём каталог для выполнения лабораторной работы № 9, переходим в него и создаём файл lab9-1.asm: (рис. 4.1).

Рис. 4.1: Создание нового каталога и файла в нём

В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы _calcul. Изучаем его(рис. 4.2).

```
lab9-1.asm          [-M--] 21 L:[  1+ 0   1/ 29] *(21
%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
```

Рис. 4.2: Текст программы из листинга

Проверяем работу файла: (рис. 4.3).

```
dvkim@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
dvkim@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
dvkim@dk3n55 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 3
2x+7=13
dvkim@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 4.3: Запуск программы

Изменяем текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul:(рис. 4.4).

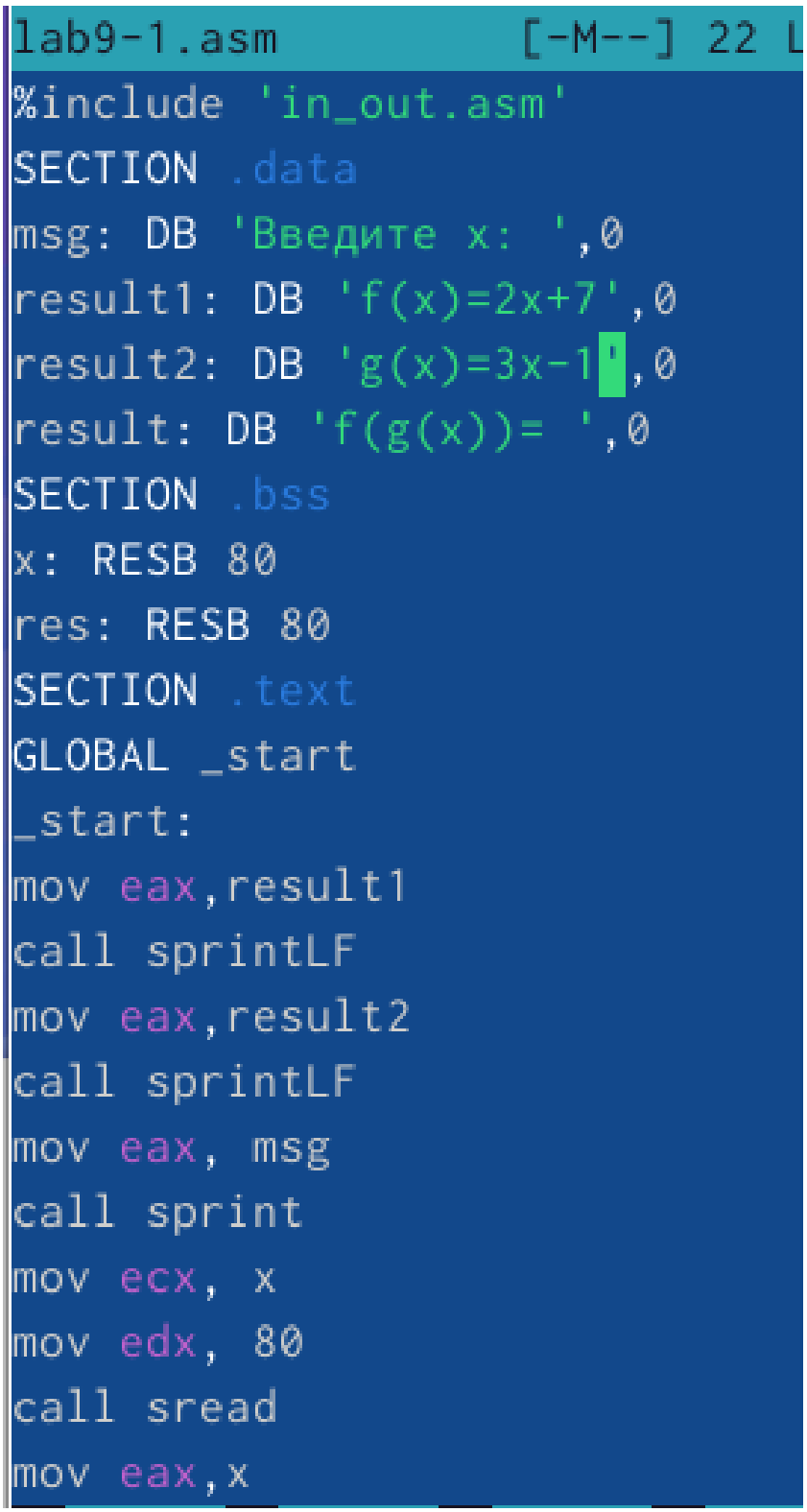

Рис. 4.4: Изменение текста программы

Проверяем работу файла программы с внесёнными изменениями: (рис. 4.5).

```
dvkim@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
dvkim@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
dvkim@dk3n55 ~/work/arch-pc/lab09 $ ./lab9-1
f(x)=2x+7
g(x)=3x-1
Введите x: 3
f(g(x))= 23
```

Рис. 4.5: Запуск программы

Создаём файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!). Загружаем исполняемый файл в отладчик gdb: (рис. 4.6).

```
dvkim@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-2.lst lab9-2.asm
dvkim@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-2 lab9-2.o
dvkim@dk3n55 ~/work/arch-pc/lab09 $ gdb lab9-2
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 lab9-2...
(gdb)
```

Рис. 4.6: Загрузка исполняемого файл в отладчик gdb

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run: (рис. 4.7).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvkim/work/arch-pc/lab09/lab9
-2
Hello, world!
[Inferior 1 (process 4734) exited normally]
(gdb)
```

Рис. 4.7: Проверка работы программы

Для более подробного анализа программы устанавливаем брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаем её. (рис. 4.8).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvkim/work/arch-pc/lab09/lab9
-2

Breakpoint 1, _start () at lab9-2.asm:9
9       mov eax, 4
(gdb)
```

Рис. 4.8: Установка брейкпоинта и запуск программы

Посмотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки _start (рис. 4.9).

```
(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.
(gdb)
```

Рис. 4.9: Дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel. Отличия состоят в том, что в дисассимилированном отображении используются % и $, Intel их не использует: (рис. 4.10).

```
(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.
(gdb)
```

Рис. 4.10: Ввод команды set disassembly-flavor intel

Включаем режим псевдографики для более удобного анализа программы (рис. 9.2): (рис. 4.11).

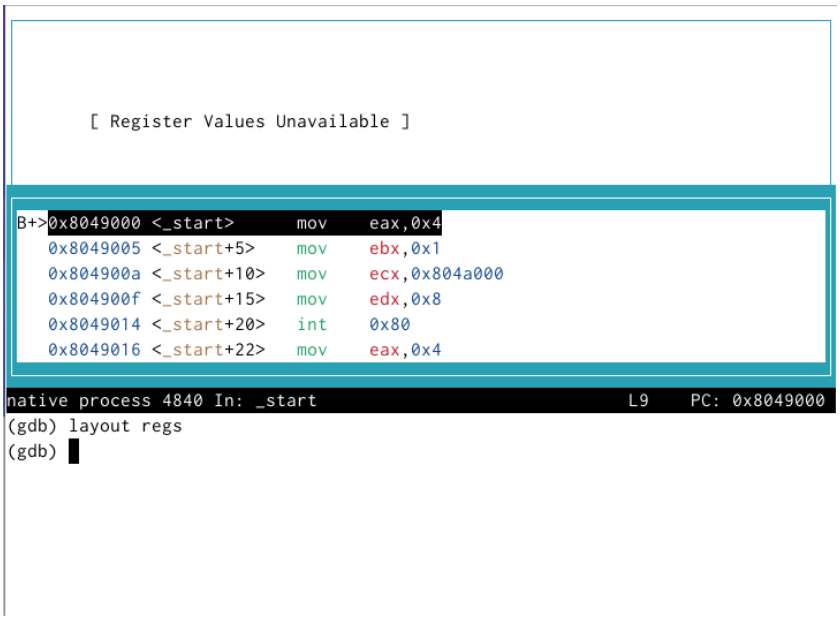

Рис. 4.11: Включение режима псевдографики

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверяем это с помощью команды info breakpoints: (рис. 4.12).

```
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb)
```

Рис. 4.12: Ввод команды info breakpoints

Установим еще одну точку останова по адресу инструкции. Определяем адрес предпоследней инструкции (mov ebx,0x0) и устанавливаем точку останова: (рис. 4.13).

```
   0x804902a <_start+42>   int    0x80
   0x804902c <_start+44>   mov    eax,0x1
   0x8049031 <_start+49>   mov    ebx,0x0
b+ 0x8049036 <_start+54>   int    0x80
   0x8049038               add    BYTE PTR [eax],al
   0x804903a               add    BYTE PTR [eax],al
native process 4840 In: _start                          L9    PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049036
Breakpoint 2 at 0x8049036: file lab9-2.asm, line 21.
(gdb)
```

Рис. 4.13: Установка точки останова

Посмотрим информацию о всех установленных точках останова: (рис. 4.14).

```
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x08049036 lab9-2.asm:21
(gdb)
```

Рис. 4.14: Ввод команды info breakpoints

Выполняем 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. (рис. 4.15).

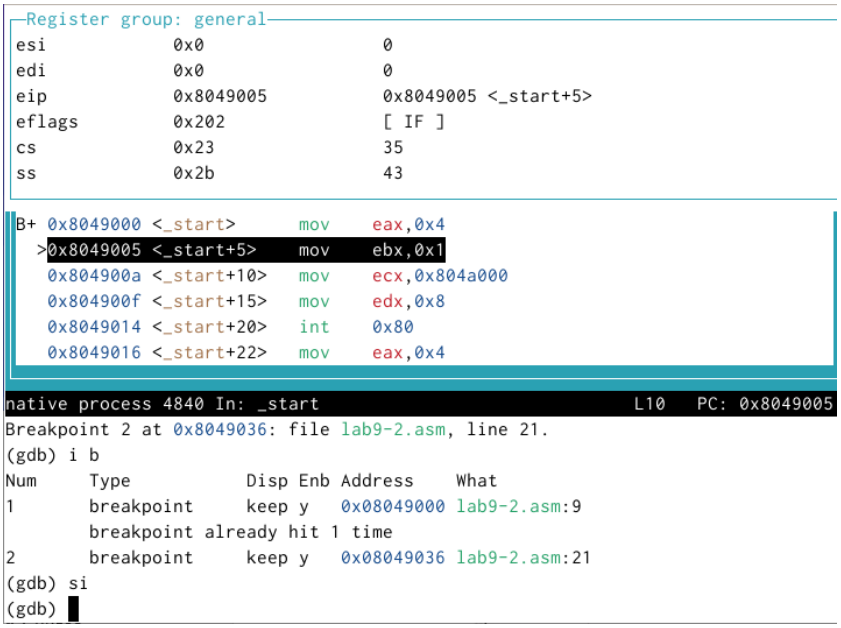

Рис. 4.15: Использование команды stepi

Посмотреть содержимое регистров также можно с помощью команды info registers: (рис. 4.16).

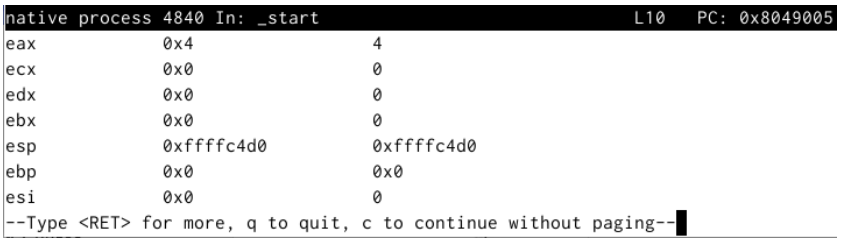

Рис. 4.16: Ввод команды info registers

Посмотрите значение переменной msg1 по имени: (рис. 4.17).

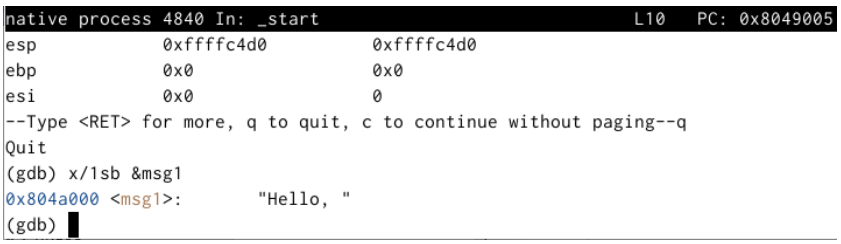

Рис. 4.17: Просмотр значения переменной msg1

Посмотрим значение переменной msg2 по адресу. Адрес переменной можно

определить по дизассемблированной инструкции. В данном случае я просмотрел значение переменной по имени. (рис. 4.18).

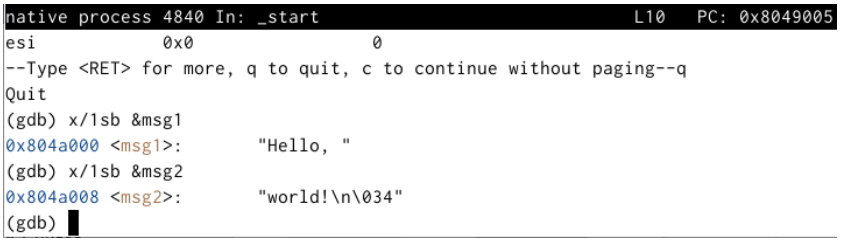

Рис. 4.18: Просмотр значения переменной msg2

Изменяем первый символ переменной msg1 (рис. 9.5): (рис. 4.19).

Рис. 4.19: Изменение первого символа переменной

Заменяем любой символ во второй переменной msg2: (рис. 4.20).

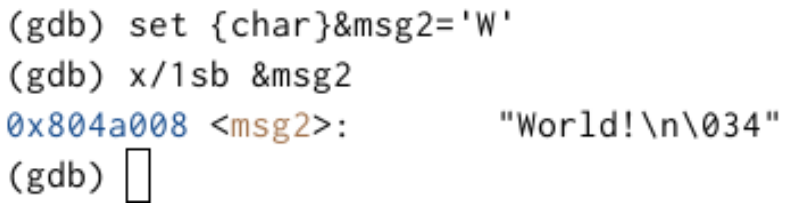

Рис. 4.20: Изменение первого символа другой переменной

Выводим в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx: (рис. 4.21).

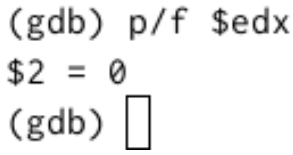

Рис. 4.21: Вывод значения регистра edx

С помощью команды set изменяем значение регистра ebx. Разница состоит в

том, что команда выводит два разных значания, значения разнятся, так как в первый раз вносится значение 2, а второй регистр равен двум: (рис. 4.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$3 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2
(gdb)
```

Рис. 4.22: Изменение значения регистра ebx

Завершаем выполнение программы с помощью команды continue и выходим из GDB с помощью команды quit (рис. 4.23).

```
(gdb) c
Continuing.
hello, World!

Breakpoint 2, _start () at lab9-2.asm:20
(gdb) q
```

Рис. 4.23: Завершение выполнения программы и выход

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8: (рис. 4.24).

```
dvkim@dk8n76 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/ar
ch-pc/lab09/lab9-3.asm
dvkim@dk8n76 ~/work/arch-pc/lab09 $
```

Рис. 4.24: Копирование файла

Создаём исполняемый файл. (рис. 4.25).

```
dvkim@dk8n76 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-3.lst lab9-3.asm
dvkim@dk8n76 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-3 lab9-3.o
```

Рис. 4.25: Создание исполняемого файла

Загружаем исполняемый файл в отладчик, указав заданные аргументы. Для начала установим точку останова перед первой инструкцией в программе и запустим её: (рис. 4.26).

```
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvkim/work/arch-pc/lab09/lab9
-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:5
5       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.26: Загрузка исполняемого файла в отладчик

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число, равное количеству аргументов командной строки. Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адесу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] – второго и т.д. Элементы расположены с интервалом в 4, так как стек может хранить до 4 байт: (рис. 4.27).

```
(gdb) x/x $esp
0xffffc460:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffc6bb:     "/afs/.dk.sci.pfu.edu.ru/home/d/v/dvkim/work/arch-pc/lab09/lab9-
3"
(gdb) x/s *(void**)($esp + 8)
0xffffc6fc:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffc70e:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffc71f:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffc721:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.27: Просмотр остальных позиций стека

Выполняем задания для самомстоятельной работы. Преобразуем программу из лабораторной работы №8, реализовав вычисление значения функции f(x) как подпрограмму: (рис. 4.28).

```
lab9-4.asm         [-M--] 13 L:[  1+12  13/ 32] *(187 / 362b) 0010 0x00A  [*][X]
%include 'in_out.asm'
SECTION .data
func DB 'f(x)=15x+2',0
a DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
mov eax,func
call sprintLF
next:
cmp ecx,0
jz _end
pop eax
call atoi
call pod
add esi,eax
loop next
_end:
 1Помощь 2Сохран 3Блок  4Замена 5Копия  6Пер~ть 7Поиск  8Уда~ть 9МенюMC 10Выход
```

Рис. 4.28: Изменение текста программы

Проверяем правильность: (рис. 4.29).

```
dvkim@dk8n76 ~/work/arch-pc/lab09 $ ./lab9-4 1
f(x)=15x+2
Результат: 17
dvkim@dk8n76 ~/work/arch-pc/lab09 $
```

Рис. 4.29: Запуск программы

В листинге 9.3 приведена программа вычисления выражения (3 + 2) 4 + 5. При запуске данная программа дает неверный результат - выражение равно 25, но программа выдаёт 10: (рис. 4.30).

```
dvkim@dk8n76 ~/work/arch-pc/lab09 $ nasm -f elf lab9-5.asm
dvkim@dk8n76 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-5 lab9-5.o
dvkim@dk8n76 ~/work/arch-pc/lab09 $ ^C
dvkim@dk8n76 ~/work/arch-pc/lab09 $ ./lab9-5
Результат: 10
dvkim@dk8n76 ~/work/arch-pc/lab09 $
```

Рис. 4.30: Получение неправильного ответа

Запускаем отладчик и анализируем: (рис. 4.31).

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:     mov    ebx,0x3
   0x080490ed <+5>:     mov    eax,0x2
   0x080490f2 <+10>:    add    ebx,eax
   0x080490f4 <+12>:    mov    ecx,0x4
   0x080490f9 <+17>:    mul    ecx
   0x080490fb <+19>:    add    ebx,0x5
   0x080490fe <+22>:    mov    edi,ebx
   0x08049100 <+24>:    mov    eax,0x804a000
   0x08049105 <+29>:    call   0x804900f <sprint>
   0x0804910a <+34>:    mov    eax,edi
   0x0804910c <+36>:    call   0x8049086 <iprintLF>
   0x08049111 <+41>:    call   0x80490db <quit>
End of assembler dump.
(gdb)
```

Рис. 4.31: Запуск отладчика

Исправляем ошибки в регистрах (например, в одной строчке они перепутаны): (рис. 4.32).

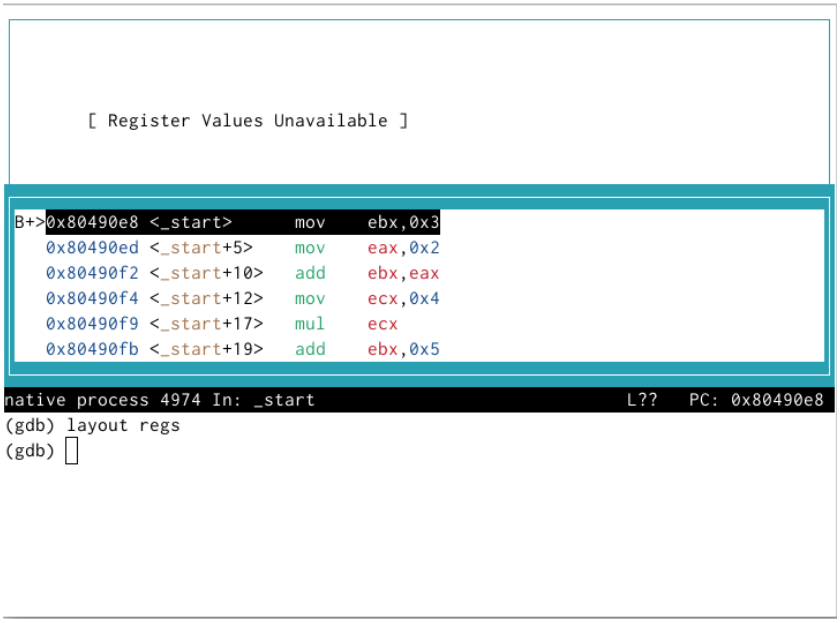

Рис. 4.32: Исправление ошибок

Запускаем программу и получаем правильный ответ: (рис. 4.33).

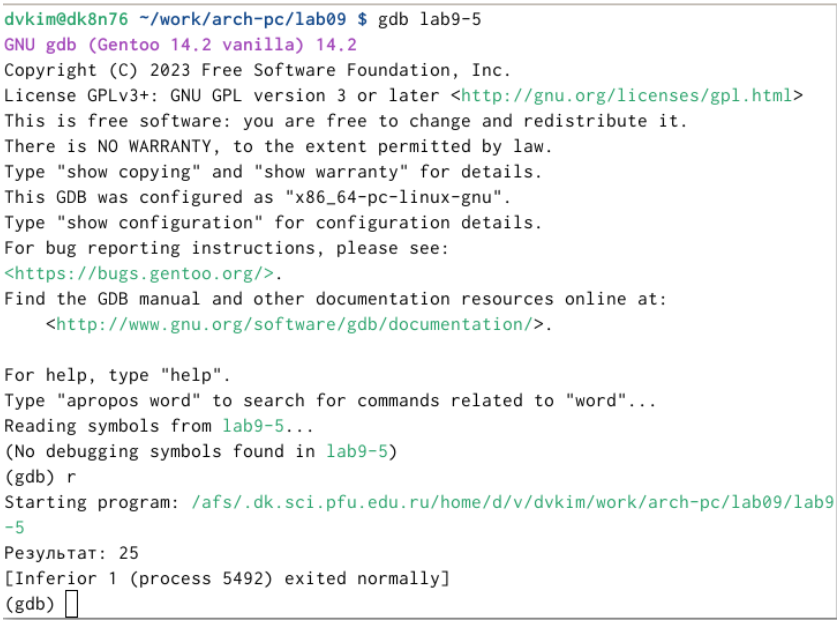

Рис. 4.33: Запуск программы и получение правильного ответа

5 Выводы

В ходе данной работы я приобрёл навыки написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.
2. Robbins A. Bash Pocket Reference. O’Reilly Media, 2016. 156 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Newham C. Learning the bash Shell: Unix Shell Programming. O’Reilly Media, 2005. 354 с.