

Лабораторная работа №9

НКАбд-02-23

Выборнов Дмитрий Валерьевич

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программ с помощью GDB.
3. Добавление точек останова.
4. Обработка аргументов командной строки в GDB.
5. Задание для самостоятельной работы.

3 Теоретическое введение

3.1 Отладка.

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки.

Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); • использование специальных программ-отладчиков. 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 строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного

графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

3.2 Подпрограммы.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `esp`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM.

Создаю новый файл и ввожу в него текст программы.

```
GNU nano 6.2
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

    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. 4.1: Первый шаг.

Создаю исполняемый файл и проверяю его работу.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 6
2x + 7 = 19
```

Рис. 4.2: Второй шаг.

Изменяю текст программы.

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ', 0
result: DB 'f(g(x)) = ', 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

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

    call quit

_calcul:
    call _subcalcul
```

Рис. 4.3: Третий шаг.

Запускаю изменённую программу и проверяю её работу.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-1-2.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1-2 lab9-1-2.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-1-2
Введите x: 5
f(g(x)) = 35
```

Рис. 4.4: Четвёртый шаг.

4.2 Отладка программ с помощью GDB.

Создаю новый файл и ввожу в него текст программы вывода Hello, world!

```
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. 4.5: Пятый шаг.

Получаю исполняемый файл, загружаю его в GDB и проверяю его при помощи команды run.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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) r
Starting program: /home/dvvybornov/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 1114) exited normally]
(gdb) _
```

Рис. 4.6: Шестой шаг.

Запускаю программу с брейкпоинтом.

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 12.
(gdb) r
Starting program: /home/dvvybornov/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:12
12      mov eax, 4
(gdb) _
```

Рис. 4.7: Седьмой шаг.

Изучаю диссимилированный код программы с двумя разными синтаксисами.

```

(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. 4.8: Восьмой шаг.

Основные различия отображения синтаксиса машинных команд в режимах АТТ и Intel заключаются в отображении третьего столбца данных. Включаю режим псевдографики.

```
[ Register Values Unavailable ]

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
0x8049031 <_start+49> mov    ebx,0x0

native process 1117 In: _start L12 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 4.9: Девятый шаг.

4.3 Добавление точек останова.

Создаю ещё одну точку останова и проверяю информацию о них.

```

(gdb) layout regs
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:12
          breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 25.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:12
          breakpoint already hit 1 time
2        breakpoint keep y  0x08049031 lab9-2.asm:25
(gdb) _

```

Рис. 4.10: Десятый шаг.

4.4 Работа с данными программы в GDB.

Во время работы программы изменяются значения тех регистров, с которыми она взаимодействует, а также значение регистра `esp`.

Проверяю информацию о регистрах.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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

native process 1117 In: _start L18 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.11: Одиннадцатый шаг.

Просматриваю значение регистров msg1 и msg2.

```

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

```

Рис. 4.12: Двенадцатый шаг.

Изменяю несколько символов в msg1 и msg2.

```
(gdb) set {char} &msg1 = 'h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char} &msg2 = 'b'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "borld!\n\034"
(gdb) _
```

Рис. 4.13: Тринадцатый шаг.

Вывожу значение регистра eax в разных форматах.

```
(gdb) p /x $edx
$2 = 0x8
(gdb) p /t $edx
$3 = 1000
(gdb) p /c $edx
$4 = 8 '\b'
(gdb) _
```

Рис. 4.14: Четырнадцатый шаг.

Изменяю значение регистра ebx.

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

Рис. 4.15: Пятнадцатый шаг.

В первый раз изменение не сработало, так как в ebx хранилось численное значение.

4.5 Обработка аргументов командной строки в GDB.

Копирую нужный файл из восьмой лабораторной работы, создаю исполняемый файл и загружаю его в GDB.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2
'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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...
```

Рис. 4.16: Шестнадцатый шаг.

Запускаю программу.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 6.
(gdb) run
Starting program: /home/dvvybornov/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:6
6      pop ecx
```

Рис. 4.17: Семнадцатый шаг.

Просматриваю остальные позиции стека.


```

Breakpoint 1, _start () at lab9-3.asm:6
6      pop ecx
(gdb) x/x $esp
0xffffd160: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd2c6: "/home/dvvybornov/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2f1: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd303: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd314: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd316: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) _

```

Рис. 4.18: Восемнадцатый шаг.

Шаг изменения адреса равен 4, так как для каждого объекта в стеке отводится 4 бита.

4.6 Задание для самостоятельной работы.

4.6.1 №1

Преобразовываю программу из восьмой лабораторной работы.

```

#include 'in_out.asm'

SECTION .data
msg1 db "Результат: ", 0
msg2 db "Функция: 5(2 + x)", 0
SECTION .text
GLOBAL _start
_start:

    mov eax, msg2
    call sprintf

    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end

    pop eax
    call atoi

    call _calc

    loop next

_end:
    mov eax, msg1
    call sprintf
    mov eax, esi
    call iprintLF
    call quit

_calc:

```

Рис. 4.19: Первый шаг первого задания.

Проверяю работу изменённой программы.

```

dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-4 8 18 28
Функция: 5(2 + x)
Результат: 300

```

Рис. 4.20: Второй шаг первого задания.

4.6.2 №2

Копирую текст нужной программы в новый файл.

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

Рис. 4.21: Первый шаг второго задания.

Запускаю эту программу при помощи GDB.

```

--Register group: general
eax      0x804a000      134520832      ecx      0x4          4
edx      0x0           0          ebx      0xa         10
esp      0xffffd1a0    0xffffd1a0    ebp      0x0         0x0
esi      0x0           0          edi      0xa         10
eip      0x8049105     0x8049105 <_start+29> eflags   0x206         [ PF IF ]
cs       0x23          35          ss       0x2b         43
ds       0x2b          43          es       0x2b         43

0x80490f4 <_start+12> mov     ecx,0x4
0x80490f9 <_start+17> mul     ecx
0x80490fb <_start+19> add     ebx,0x5
0x80490fe <_start+22> mov     edi,ebx
0x8049100 <_start+24> mov     eax,0x804a000
> 0x8049105 <_start+29> call    0x804900f <sprint>
0x804910a <_start+34> mov     eax,edi
0x804910c <_start+36> call    0x8049086 <iprintLF>
0x8049111 <_start+41> call    0x80490db <quit>

native process 2232 In: _start L17 PC: 0x8049105
(gdb) layout regs
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-5.asm, line 8.
(gdb) run
Starting program: /home/dvvybornov/work/arch-pc/lab09/lab9-5

Breakpoint 1, _start () at lab9-5.asm:8
(gdb) si
(gdb)

```

Рис. 4.22: Второй шаг второго задания.

Все проблемы этой программы связаны с тем, как в ней использованы регистры.

Исправляю текст программы.

```

#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edx, eax

mov eax, div
call sprint
mov eax,edx
call iprintLF

call quit

```

Рис. 4.23: Третий шаг второго задания.

Проверяю работу программы.

```

dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25

```

Рис. 4.24: Четвёртый шаг второго задания.

Теперь программа даёт правильный результат.

5 Выводы

Выполнив эту лабораторную работу, я приобрёл навыки написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.