

Архитектура компьютера

Отчёт по лабораторной работе №9

Чилеше Лупупа

Содержание

| | | |
|---|-------------------------------------|----|
| 1 | Цель работы | 2 |
| 2 | Задание | 2 |
| 3 | Теоретическое введение..... | 2 |
| 4 | Выполнение лабораторной работы..... | 3 |
| 5 | Выводы..... | 10 |
| | Список литературы | 10 |

1 Цель работы

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

2 Задание

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm
2. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу
3. Создайте файл lab09-2.asm с текстом программы из Листинга 9.2
4. Проверьте работу программы, запустив ее в оболочке GDB

3 Теоретическое введение

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. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки.

После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново. 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 строки, 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. Наиболее популярные виды точек останова: • Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом); • Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его). Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

4 Выполнение лабораторной работы

- 1) Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm

```
lupupachileshe@ubuntu:~$ mkdir ~/work/arch-pc/lab09
lupupachileshe@ubuntu:~$ cd ~/work/arch-pc/lab09
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ touch lab09-1.asm
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ ls
lab09-1.asm
lupupachileshe@ubuntu:~/work/arch-pc/lab09$
```

1) Создание файла

- 2) Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Создаю исполняемый файл и проверяю его работу (2-3)

```
GNU nano 6.2 /home/lupupachleshe/work/arch-pc/lab09/lab09-1.asm *
#include 'ln_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:
;-----
; Основная программа
;-----
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 iprintf
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
^O Help      ^O Write Out ^M Where Is   ^X Cut       ^T Execute   ^C Location  ^U Undo
^X Exit      ^R Read File ^A Replace    ^U Paste     ^J Justify   ^_ Go To Line ^E Redo
```

2)Текст файла

```
lupupachleshe@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lupupachleshe@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
lupupachleshe@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 23
2x+7=53
lupupachleshe@ubuntu:~/work/arch-pc/lab09$
```

3)Создание файла

- 3) Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Создаю файл и проверяю его работу (4-5)

```
mov eax, result
call sprint
mov eax, [res]
call iprintf
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul ;Выход подпрограммы _subcalcul
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 ; выход из подпрограммы
^O Help      ^O Write Out ^M Where Is   ^X Cut       ^T Execute   ^C Location  ^U Undo
^X Exit      ^R Read File ^A Replace    ^U Paste     ^J Justify   ^_ Go To Line ^E Redo
```

4)Текст программы изменённый

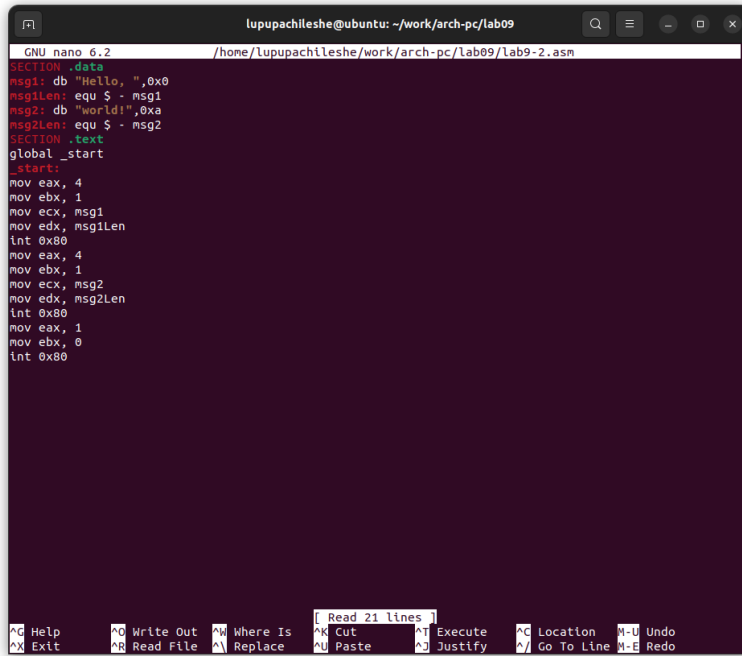
```
lupupachleshe@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lupupachleshe@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
lupupachleshe@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 23
2(3x-1)+7=143
lupupachleshe@ubuntu:~/work/arch-pc/lab09$
```

5)Работа файла

4) Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (6-8)

```
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ touch lab9-2.asm
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab9-2.asm
lupupachileshe@ubuntu:~/work/arch-pc/lab09$
```

6) Создание файла



```
GNU nano 6.2 /home/lupupachileshe/work/arch-pc/lab09/lab9-2.asm
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
```

7) Текст файла

```
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-2.asm
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ ./lab9-2
Hello, world!
lupupachileshe@ubuntu:~/work/arch-pc/lab09$
```

8) Работа файла

- 5) Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Скачиваю gdb и запускаю его (9-10)

```
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
lupupachileshe@ubuntu:~/work/arch-pc/lab09$ gdb lab9-2
```

9) Скачиваю gdb

```

lupupachleshe@ubuntu:~/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) run
Starting program: /home/lupupachleshe/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 41858) exited normally]
(gdb)

```

10) Запуск gdb и файла

- 6) Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её

```

(gdb) break _start
Breakpoint 1 at 0x00490000: file lab9-2.asm, line 12.
(gdb) run
Starting program: /home/lupupachleshe/work/arch-pc/lab09/lab9-2

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

```

11) Установка `break_point` и запуск программы с ней

- 7) Смотрю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00049000 <+0>:    mov     $0x4,%eax
0x00049005 <+5>:    mov     $0x1,%ebx
0x0004900a <+10>:   mov     $0x804a000,%ecx
0x0004900f <+15>:   mov     $0x8,%edx
0x00049014 <+20>:   int     $0x80
0x00049016 <+22>:   mov     $0x4,%eax
0x0004901b <+27>:   mov     $0x1,%ebx
0x00049020 <+32>:   mov     $0x804a008,%ecx
0x00049025 <+37>:   mov     $0x7,%edx
0x0004902a <+42>:   int     $0x80
0x0004902c <+44>:   mov     $0x1,%eax
0x00049031 <+49>:   mov     $0x0,%ebx
0x00049036 <+54>:   int     $0x80
End of assembler dump.
(gdb)

```

12) Дисассимилированный код

- 8) Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00049000 <+0>:    mov     eax,0x4
0x00049005 <+5>:    mov     ebx,0x1
0x0004900a <+10>:   mov     ecx,0x804a000
0x0004900f <+15>:   mov     edx,0x8
0x00049014 <+20>:   int     0x80
0x00049016 <+22>:   mov     eax,0x4
0x0004901b <+27>:   mov     ebx,0x1
0x00049020 <+32>:   mov     ecx,0x804a008
0x00049025 <+37>:   mov     edx,0x7
0x0004902a <+42>:   int     0x80
0x0004902c <+44>:   mov     eax,0x1
0x00049031 <+49>:   mov     ebx,0x0
0x00049036 <+54>:   int     0x80
End of assembler dump.
(gdb)

```

13) Отображение команд с Intel'овским синтаксисом

9) Включаю режим псевдографики для более удобного анализа программы

```
lupupachilleshe@ubuntu: ~/work/arch-pc/lab09
~Register group: general~
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      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

native process 42698 In: _start L12 PC: 0x8049000
(gdb) layout regs
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/lupupachilleshe/work/arch-pc/lab09/lab9-2
Breakpoint 1, _start () at lab9-2.asm:12
(gdb) |
```

14)Режим псевдографики

10) С помощью info breakpoints узнаю информацию об установленных точках останова

```
Breakpoint 1, _start () at lab9-2.asm:12
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:12
      breakpoint already hit 1 time
(gdb)
```

15)info breakpoints

11) Определяю адрес предпоследней инструкции (mov ebx,0x0) и устанавливаю точку останова. Смотрю информацию об установленных точках останова (16-17)

```
lupupachilleshe@ubuntu: ~/work/arch-pc/lab09
~Register group: general~
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      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

native process 42698 In: _start L12 PC: 0x8049000
lab9/lab9-2
Breakpoint 1, _start () at lab9-2.asm:12
(gdb) info breakpoints
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) |
```

16)Установка точки останова

```

(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab9-2.asm:12
2        breakpoint already hit 1 time
2        breakpoint keep y 0x08049031 lab9-2.asm:25
(gdb)

```

17) Вывод информации о точках останова

- 12) Выполняю 5 инструкций с помощью команды stepi (или si) и слежу за изменением значений регистров. Значения регистров eax, edx, ecx, esp, eip, cs, ds, ebx, ss, eflags, es изменяются

```

-Register group: general-
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      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

native process 42698 In: start L13 PC: 0x8049005
Breakpoint 3 at 0x08049031: file lab9-2.asm, line 25.
(gdb) i b
Undefined command: "ib". Try "help".
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab9-2.asm:12
2        breakpoint already hit 1 time
2        breakpoint keep y 0x08049031 lab9-2.asm:25
3        breakpoint keep y 0x08049031 lab9-2.asm:25
(gdb) si
(gdb)

```

18) 5 инструкций si

- 13) Посмотреть содержимое регистров также можно с помощью команды info registers

```

native process 42698 In: start L13 PC: 0x8049005
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

19) info registers

- 14) Смотрю значение переменной msg1 по имени и переменной msg2 по адресу (20-21)

```

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

```

20) Значение переменной msg1 по имени

```

(gdb) x 0x804a000
0x804a000 <msg2>: "world!\n\034"
(gdb)

```

21) Значение переменной msg2 по адресу

- 15) Изменяю первый символ переменной msg1

```

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

```

22) Изменение первого символа переменной msg1

16) Заменяю символ во второй переменной msg2

```
(gdb) set (char)0x004a008='w'
(gdb) x/1sb &msg2
0x004a008 <msg2>: "world!\n\034"
(gdb)
```

23)Изменение символа второй переменной

17) С помощью команды set изменяю значение регистра ebx. В первом случае выводит значение символа (его код), во втором - число

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

25)Изменяю значение регистра ebx

18) Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q)

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

Breakpoint 2, _start () at lab9-2.asm:25
(gdb) q
A debugging session is active.

    Inferior 1 [process 42698] will be killed.

Quit anyway? (y or n)
```

26)Завершаю программу

19) Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm, создаю исполняемый файл, загружаю исполняемый файл в отладчик, указав аргументы:

```
lupupachleshe@ubuntu: /work/arch-pc/lab8$ cp -f /work/arch-pc/lab8-2.asm -f /work/arch-pc/lab09/lab09-3.asm
cp: cannot stat '/home/lupupachleshe/work/arch-pc/lab8-2.asm': No such file or directory
lupupachleshe@ubuntu: /work/arch-pc/lab8$ cp -f /work/arch-pc/lab08/lab8-2.asm -f /work/arch-pc/lab09/lab09-3.asm
lupupachleshe@ubuntu: /work/arch-pc/lab8$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
lupupachleshe@ubuntu: /work/arch-pc/lab8$ ld -m elf_i386 -o lab09-3 lab09-3.o
lupupachleshe@ubuntu: /work/arch-pc/lab8$ gdb --args lab09-3 аргумент1 аргумент2 аргумент3
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:
<http://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 lab09-3...
(gdb)
```

27)Копирование, создание, загрузка файла

20) Для начала уставляю точку останова перед первой инструкцией в программе и запускаю её.

```
(gdb) b _start
Breakpoint 1 at 0x004a008: file lab09-3.asm, line 11.
(gdb) r
Starting program: /home/lupupachleshe/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент3

Breakpoint 1, _start () at lab09-3.asm:11
11      pop ecx          ; Извлекаем из стека в 'ecx' количество
(gdb)
```

28)Точка установка и запуск

- 21) Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)

```
(gdb) x/x $esp
0xffffd110: 0x00000005
(gdb)
```

29) Регистр esp

- 22) Смотрю остальные позиции стека. Шаг изменения равен 4 потому что шаг - int, а под этот тип данных выделяется 4 байта

```
(gdb) x/s *(void**)($esp + 4)
0xffffd0ea: "/home/lupupachleshe/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd0e8: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd0e6: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd0e4: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd0e2: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
esp: <error: Cannot access memory at address 0x0>
(gdb)
```

30) Остальные позиции стека

5 Выводы

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

Список литературы