

Отчет по лабораторной работе 9

Понятие подпрограммы.Отладчик GDB.

Зайцева П.Е.

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выводы	14

Список иллюстраций

2.1	Рис.1	7
2.2	Рис.2	7
2.3	Рис.3	8
2.4	Рис.4	8
2.5	Рис.5	9
2.6	Рис.6	9
2.7	Рис.7	10
2.8	Рис.8	10
2.9	Рис.9	11
2.10	Рис.10	11
2.11	Рис.11	12
2.12	Рис.12	12
2.13	Рис.13	13

Список таблиц

1 Цель работы

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

2 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, отрабатывает, 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.

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. # Выполнение лабораторной работы
Создала каталог для выполнения лабораторной работы No 9, перешла в него и создала файл lab09-1.asm

```
pezayjceva@dk8n54 ~ $ mkdir ~/work/arch-pc/lab09
pezayjceva@dk8n54 ~ $ cd ~/work/arch-pc/lab09
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ touch lab09-1.asm
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Рис.1

Ввела в файл lab09-1.asm текст программы из листинга 9.1. Создала исполняемый файл и проверила его работу.

```
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2x+7=9
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $
```

Рис. 2.2: Рис.2

Создала файл lab09-2.asm с текстом программы из Листинга 9.2. Получила исполняемый файл. Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ провела с ключом '-g'. Проверила работу программы, запустив ее в оболочке GDB с помощью команды run

```

pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ gdb lab09-2
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 lab09-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/p/e/pezayjceva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4892) exited normally]
(gdb)

```

Рис. 2.3: Рис.3

Включила режим псевдографики для более удобного анализа программы. В этом режиме есть три окна:

- В верхней части видны названия регистров и их текущие значения;
- В средней части виден результат дисассимилирования программы;
- Нижняя часть доступна для ввода команд.

```

[ 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

native process 4944 In: _start
(gdb) layout regs
(gdb)

```

Рис. 2.4: Рис.4

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверила это с помощью команды `info breakpoints`. Определила адрес предпоследней инструкции (`mov ebx,0x0`) и установила точку останова.

```
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 4944 In: _start
1 breakpoint keep y 0x08049000 lab09-2.asm:9
  breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint        keep y 0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
2      breakpoint        keep y 0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 2.5: Рис.5

Посмотрела содержимое регистров также можно с помощью команды `info registers` Посмотрела значение переменной `msg2` по адресу.

```
> 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

native process 4944 In: _start
fs          0x0          0
gs          0x0          0
(gdb) si
(gdb) x/1sb &msg1
No symbol "msg1" in current context.
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)
```

Рис. 2.6: Рис.6

С помощью команды `set` изменить значение регистра `ebx`.

```
> 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

native process 4944 In: _start
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
```

Рис. 2.7: Рис.7

Скопировала файл lab8-2.asm, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. Создала исполняемый файл.

```
pezayjeeva@dk8n54 ~/work/arch-pc/lab08 $ cd ~/work/arch-pc/lab09
pezayjeeva@dk8n54 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
pezayjeeva@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
pezayjeeva@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
pezayjeeva@dk8n54 ~/work/arch-pc/lab09 $
```

Рис. 2.8: Рис.8

Для загрузки в gdb программы с аргументами необходимо использовать ключ -args. Загрузила исполняемый файл в отладчик, указав аргументы. Для начала установила точку останова перед первой инструкцией в программе и запустила ее.

```

pezeyjceva@dk8n54 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/p/e/pezeyjceva/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\
3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Рис. 2.9: Рис.9

Посмотрела остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д.

```

(gdb) x/x $esp
0xffffc2b0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffc54a: "/afs/.dk.sci.pfu.edu.ru/home/p/e/pezeyjceva/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc591: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc5a3: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc5b4: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc5b6: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.10: Рис.10

Задание для самостоятельной работы.

Создала файл lab09-1.asm. Ввела программу из листинга 9.3. При запуске данная программа дает неверный результат. Проверила это.

```

pezhayceva@dk8n54 ~/work/arch-pc/lab09 $ touch lab09-4.asm
pezhayceva@dk8n54 ~/work/arch-pc/lab09 $ mc

pezhayceva@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
pezhayceva@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
ld: предупреждение: невозможно найти символ входа _start; начальный адрес не устанавливается
pezhayceva@dk8n54 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2x+7=9
pezhayceva@dk8n54 ~/work/arch-pc/lab09 $ ./lab09-4
bash: ./lab09-4: не удалось запустить бинарный файл: Ошибка формата выполняемого файла
pezhayceva@dk8n54 ~/work/arch-pc/lab09 $

```

Рис. 2.11: Рис.11

С помощью отладчика GDB, анализируя изменения значений регистров, определила ошибку и исправьте ее. Ошибка была в строках: `add ebx,eax mov ecx,4 mul ecx add ebx,5 mov edi,ebx`

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/p/e/pezhayceva/wc
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 2.12: Рис.12

Исправленный листинг: `%include 'in_out.asm' SECTION .data div: DB 'Результат:',0 SECTION .text GLOBAL _start _start: ; -- Вычисление выражения (3+2)*4+5 mov ebx,3 mov eax,2 add eax,ebx mov ecx,4 mul ecx add eax,5 mov edi,eax ; -- Вывод результата на экран mov eax,div call sprint mov eax,edi call iprintLF call quit`

Проверила его работу.

```
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $ ./lab09-4
Результат: 25
pezayjceva@dk8n54 ~/work/arch-pc/lab09 $
```

Рис. 2.13: Рис.13

3 Выводы

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