

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Нуруллаев Бахадур

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Реализация подпрограмм в NASM	10
4.2	Отладка программ с помощью GDB	12
4.2.1	Добавление точек останова	14
4.2.2	Работа с данными программы в GDB	15
5	Выводы	19
	Список литературы	20

Список иллюстраций

4.1	Создание файлов для лабораторной работы	10
4.2	Ввод текста программы из листинга 9.1	11
4.3	Запуск исполняемого файла	11
4.4	Изменение текста программы согласно заданию	12
4.5	Запуск исполняемого файла	12
4.6	Ввод текста программы из листинга 9.2	13
4.7	Включение режима псевдографики	14
4.8	Установка точек останова и просмотр информации о них . . .	15
4.9	До использования команды <code>stepi</code>	16
4.10	Просмотр значений переменных	17
4.11	Использование команды <code>set</code>	18

Список таблиц

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. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. 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.

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.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки

отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit` (или сокращённо `q`).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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

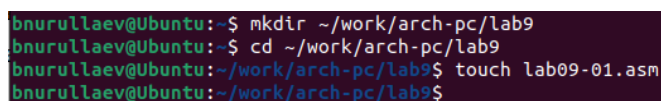
этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` 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 `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис. 4.11)



```
bnurullaev@Ubuntu:~$ mkdir ~/work/arch-pc/lab9
bnurullaev@Ubuntu:~$ cd ~/work/arch-pc/lab9
bnurullaev@Ubuntu:~/work/arch-pc/lab9$ touch lab09-01.asm
bnurullaev@Ubuntu:~/work/arch-pc/lab9$
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 4.11)

```

GNU nano 6.2 /home/bnurullaev/work/arch-pc/lab09/lab09-1.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 iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

```

Рис. 4.2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. 4.11)

```

bnurullaev@Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
bnurullaev@Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
bnurullaev@Ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17

```

Рис. 4.3: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. 4.11)

```

GNU nano 6.2 /home/bnurullaev/work/arch-pc/lab09/lab09-1.2
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 iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _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
ret

```

Рис. 4.4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу. (рис. 4.11)

```

bnurullaev@Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
bnurullaev@Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
bnurullaev@Ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=35

```

Рис. 4.5: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 4.11)

```
GNU nano 6.2 /home/bnurullaev/work/arch-pc/lab09/lab09-2.s
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

^G Справка      ^O Записать     ^W Поиск        ^K Вырезать     ^T Выполнить    ^C Позиция
^X Выход        ^R ЧитФайл     ^\ Замена      ^U Вставить     ^D Выровнять    ^_ К строке
```

Рис. 4.6: Ввод текста программы из листинга 9.2

Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. (рис. 4.11)

```
[ 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 5507 In: _start
(gdb) layout regs
(gdb) info
```

Рис. 4.7: Включение режима псевдографики

4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова. (рис. 4.11)

```
[ Register Values Unavailable ]

0x8049066      add     BYTE PTR [eax],al
0x8049068      add     BYTE PTR [eax],al
0x804906a      add     BYTE PTR [eax],al
0x804906c      add     BYTE PTR [eax],al
0x804906e      add     BYTE PTR [eax],al
0x8049070      add     BYTE PTR [eax],al
0x8049072      add     BYTE PTR [eax],al
0x8049074      add     BYTE PTR [eax],al
0x8049076      add     BYTE PTR [eax],al
0x8049078      add     BYTE PTR [eax],al
0x804907a      add     BYTE PTR [eax],al
0x804907c      add     BYTE PTR [eax],al

native process 5507 In:  start
A syntax error in expression, near '<0x8049031>'.
(gdb) i b
Num      Type      Disp Enb Address      What
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. 4.8: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. 4.11)

```

[ Register Values Unavailable ]

0x8049066      add     BYTE PTR [eax],al
0x8049068      add     BYTE PTR [eax],al
0x804906a      add     BYTE PTR [eax],al
0x804906c      add     BYTE PTR [eax],al
0x804906e      add     BYTE PTR [eax],al
0x8049070      add     BYTE PTR [eax],al
0x8049072      add     BYTE PTR [eax],al
0x8049074      add     BYTE PTR [eax],al
0x8049076      add     BYTE PTR [eax],al
0x8049078      add     BYTE PTR [eax],al
0x804907a      add     BYTE PTR [eax],al
0x804907c      add     BYTE PTR [eax],al

Терминал
native process 5507 In:  start
esp      0xffffd0a0      0xffffd0a0
ebp      0x0             0x0
esi      0x0             0
edi      0x0             0
eip      0x8049000      0x8049000 <_start>
eflags   0x202          [ IF ]
cs       0x23           35
ss       0x2b           43
--Type <RET> for more, q to quit, c to continue without paging--cs      0x23      35ds
es       0x2b           43
fs       0x0            0
gs       0x0            0
(gdb) █

```

Рис. 4.9: До использования команды stepi

(рис. 4.11)

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. (рис. 4.11)


```
[ Register Values Unavailable ]

0x8049216      add     BYTE PTR [eax],al
0x8049218      add     BYTE PTR [eax],al
0x804921a      add     BYTE PTR [eax],al
0x804921c      add     BYTE PTR [eax],al
0x804921e      add     BYTE PTR [eax],al
0x8049220      add     BYTE PTR [eax],al
0x8049222      add     BYTE PTR [eax],al
0x8049224      add     BYTE PTR [eax],al
0x8049226      add     BYTE PTR [eax],al
0x8049228      add     BYTE PTR [eax],al
0x804922a      add     BYTE PTR [eax],al
0x804922c      add     BYTE PTR [eax],al

native process 5507 In:  _start
eip      0x8049000      0x8049000 <_start>
eflags   0x202          [ IF ]
cs       0x23           35
ss       0x2b           43
--Type <RET> for more, q to quit, c to continue without paging--cs      0x23      35ds
es       0x2b           43
fs       0x0            0
gs       0x0            0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 4.10: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. 4.11)

```
[ Register Values Unavailable ]

0x8049216      add     BYTE PTR [eax],al
0x8049218      add     BYTE PTR [eax],al
0x804921a      add     BYTE PTR [eax],al
0x804921c      add     BYTE PTR [eax],al
0x804921e      add     BYTE PTR [eax],al
0x8049220      add     BYTE PTR [eax],al
0x8049222      add     BYTE PTR [eax],al
0x8049224      add     BYTE PTR [eax],al
0x8049226      add     BYTE PTR [eax],al
0x8049228      add     BYTE PTR [eax],al
0x804922a      add     BYTE PTR [eax],al
0x804922c      add     BYTE PTR [eax],al

native process 5507 In: _start
(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) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>:  "Hello, "
(gdb) set {char}0x804a000='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:  "hello, "
(gdb) █
```

Рис. 4.11: Использование команды set

5 Выводы

Во время выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм и ознакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. Архитектура ЭВМ