**Лабораторная работа №8**

**Использование отладчика *gdb***

**Целью работы** является изучение основ работы с отладчиком *gdb* в операционной системе *Linux*.

Отладчик (*debugger*) — это инструмент, который позволяет изучать программу, выполняя ее в пошаговом режиме. Трансляция программы под отладчиком — самый быстрый и эффективный способ отыскания ошибок.

В Linux обычно устанавливают свободно распространяемый отладчик *gdb* (GNU DeBugger). Это мощный и гибкий инструмент, позволяющий, кроме прочего, связывать точки выполнения программы с ее исходным кодом. Основное достоинство *gdb* — его не нужно изучать досконально. Каждый программист выбирает для себя те возможности *gdb*, которые ему необходимы. Важно понимать, что отладка призвана упрощать создание программ, а не усложнять.

# 1 ДОБАВЛЕНИЕ ОТЛАДОЧНОЙ ИНФОРМАЦИИ

Для использования gdb программа должна быть откомпилирована с опцией -g, которая добавляет в исполняемый файл дополнительную отладочную информацию:

```
$ gcc -g -o program program.c
```

Если требуется удалить эту информацию без перекомпиляции, то вызывается программа strip:

```
$ strip program
```

# 2 ЗАПУСК ОТЛАДЧИКА

Для запуска отладчика достаточно набрать команду gdb:

```
$ gdb
GNU gdb (GDB) 7.1-1mdv2010.1 (Mandriva Linux release 2010.1)
Copyright (C) 2010 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 "i586-mandriva-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb)
```

Итак, на экране появилось следующее: версия отладчика, лицензионное соглашение и приглашение (gdb), ожидающее от пользователя ввода команд. Если запустить gdb с опцией --quiet (-q — короткий вариант), то информация о версии продукта и лицензионное соглашение выводиться не будут:

```
$ gdb -q
(gdb)
Чтобы выйти из отладчика, наберите команду quit:
(gdb) quit
```

Теперь напишем программу (листинг 1) и попробуем запустить ее под отладчиком.

Программа program1.c:

```
#include <stdio.h>

void change_a (int * mya)
{
    *mya += 10;
}

int main (void)
{
    int a;
    a = 15;

    change_a (&a);
    printf ("Number: %d\n", a);

    return 0;
}
```

Сначала программу нужно откомпилировать с включением в нее отладочной информации:

```
$ gcc -g -o program1 program1.c
```

Программу можно передать отладчику следующим образом:

```
$ gdb -q program1

Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) quit
```

А можно также при помощи команды отладчика file:

```
$ gdb -q
(gdb) file program1
Reading symbols from
/home/nn/main/work/bhv/src/ch28/program1/program1...done.
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb)
```

Для просмотра исходного кода программы используется команда list. Если вызвать эту команду без параметров, то будут выведены первые несколько строк исходного кода с нумерацией:

```
(gdb) list
1 #include <stdio.h>
2
3 void change_a (int * mya)
4 {
5 *mya += 10;
6 }
7
8 int main (void)
9 {
10 int a;
(gdb)
```

Каждый новый ввод команды list без параметров будет выводить следующий набор строк исходного кода программы:

```
(gdb) l
11 a = 15;
12
13 change_a (&a);
14
15 printf ("Number: %d\n", a);
16 return 0;
17 }
(gdb)
```

Если исходный код закончился, то будет выведено соответствующее сообщение (при очередной попытке вызвать list без параметров):

```
(gdb) list
Line number 18 out of range; program1.c has 17 lines.
(gdb)
```

Если в качестве параметра команды list указать номер строки, то будет выведен блок исходного кода, "окружающий" эту строку:

```
(gdb) list 10
5 *mya += 10;
6 }
7
8 int main (void)
9 {
10 int a;
11 a = 15;
12
13 change_a (&a);
14
(gdb)
```

Можно также указать через запятую диапазон выводимых строк:

```
(gdb) list 10,15
10 int a;
11 a = 15;
12
13 change_a (&a);
14
15 printf ("Number: %d\n", a);
(gdb)
```

А если в качестве аргумента list написать имя функции, то будет выведен блок исходного кода, "окружающий" эту функцию:

```
(gdb) list main
4 {
5 *mya += 10;
6 }
7
8 int main (void)
9 {
10 int a;
11 a = 15;
12
13 change_a (&a);
(gdb) list change_a
1 #include <stdio.h>
2
3 void change_a (int * mya)
4 {
5 *mya += 10;
6 }
7
8 int main (void)
9 {
10 int a;
```

```
(gdb) quit
```

# 3 ТРАНСЛЯЦИЯ ПРОГРАММЫ ПОД ОТЛАДЧИКОМ

По отношению к программам можно выделить два стека.

- Стек данных (data stack) — специальная область памяти процесса, в которой данные читаются и записываются в порядке обратной очереди. Аргументы и локальные переменные функций языка С хранятся в стеке данных.
- Стек вызовов (call stack) — это стек вызовов функций в программе. В самой глубине этой очереди находится функция main(). Каждый элемент стека вызовов называется фреймом вызова (call frame).

Чтобы просто выполнить программу под отладчиком, введите команду run:

```
$ gdb -q program1
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) run
Starting program:
/home/nn/main/work/bhv/src/ch28/program1/program1
Number: 25
Program exited normally.
(gdb) quit
```

Строка "Program exited normally" сообщает нам, что программа завершилась обычным образом. Изменим теперь нашу программу так, чтобы функция main() возвращала числовое значение переданного аргумента (листинг 2).

Листинг 2. Программа program2.c

```
cpp
#include <stdio.h>
void change_a (int * mya)
{
    *mya += 10;
}

int main (int argc, char ** argv)
{
    int a;
    if (argc < 2) {
        fprintf (stderr, "Too few arguments\n");
        return 1;
    }
```

```
    a = 15;
    change_a (&a);
    printf ("Number: %d\n", a);
    return atoi (argv[1]);
}
Сначала запустим программу без отладчика:
$ gcc -g -o program2 program2.c
$ ./program2 10
Number: 25
$ echo $?
10
```

Команда run отладчика gdb позволяет задать список аргументов программы так же, как если бы мы делали это в командной оболочке:

```
$ gdb -q
(gdb) file program2
Reading symbols from /home/nn/program2...done.
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) run 5
Starting program:
/home/nn/program2 5
Number: 25
Program exited with code 05.
(gdb) quit
```

Сообщение "Program exited with code 05" содержит код возврата программы.

Теперь научимся выполнять программу под отладчиком в пошаговом режиме.

Команда start запускает программу и останавливает ее на входе в функцию main(). Команде start можно передавать аргументы программы:

```
$ gdb -q program2
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) start 7
Breakpoint 1 at 0x8048456: file program2.c, line 9.
Starting program:
/home/nn/main/work/bhv/src/ch28/program2/program2 7
main (argc=Cannot access memory at address 0x0
) at program2.c:9
9 {
(gdb)
```

Итак, программа остановилась на входе в функцию main(). Обратите внимание на эту строку:

```
9 {
```

Она означает, что следующим шагом будет выполнение девятой строки исходного кода, т. е. вход в функцию main(). Для продвижения по программе в отладчике gdb имеются следующие команды:

- step (s — краткая форма) — используется для выполнения одной инструкции (строки). Вызов функции, являющейся частью программы, не будет для команды step атомарной инструкцией. Иными словами, если в программе встречается вызов внутренней функции, то step "заходит" в эту функцию. Этой команде можно передать в качестве аргумента число строк, которые требуется выполнить за один раз;
- next (n — краткая форма) — работает так же, как и step, но без вхождения в функции. Для этой команды вызов функции является просто инструкцией (строкой). Команде next можно передать число строк, которые требуется выполнить за один раз;
- stepi (si — краткая форма) — функционирует как step, но атомарной инструкцией считает не строку исходного кода, а машинную команду;
- nexti (ni — краткая форма) — работает как next, но атомарной инструкцией считает не строку исходного кода, а машинную команду;
- continue (c — краткая форма) — если нет точек останова, то эта команда выполняет оставшуюся часть программы без остановок;
- kill — завершает выполнение текущей программы;
- finish — действует наподобие continue, но не до конца программы, а до возврата из текущей функции.

На каждом этапе выполнения программы мы можем просмотреть ее текущее состояние. Далее приведен список команд gdb, позволяющих осуществлять мониторинг текущего состояния программы.

- backtrace (bt — краткая форма) — выводит на экран текущее состояние стека выполнения программы. Другое название этой команды — where.
- print (p — краткая форма) — выводит на экран значение указанной переменной. В эту команду можно также передавать выражения, составленные по правилам языка C.

- display — это очень полезная команда, которая заставляет отладчик выводить значение указанного выражения при каждой остановке программы.
- show env — выводит на экран окружение процесса. Если указать в качестве аргумента конкретную переменную, то будет выведено ее значение.
- info args — выводит значения аргументов программы.
- info locals — выводит значения локальных переменных текущей функции.

Итак, отладчик остановил нашу программу на входе в функцию main(). Перейдем на следующую строку командой step:

```
(gdb) step
main (argc=2, argv=0xbfd945b4) at program2.c:12
12 if (argc < 2) {
(gdb)
Посмотрим состояние стека вызовов:
(gdb) bt
#0 main (argc=2, argv=0xbfd945b4) at program2.c:12
(gdb)
```

Поскольку мы в данный момент находимся в функции main(), которая лежит на дне стека вызовов, то вывод команды bt содержит всего один фрейм с обозначением #0. Посмотрим теперь текущую информацию об аргументах программы:

```
(gdb) info args
argc = 2
argv = (char **) 0xbfd945b4
(gdb)
```

Адрес массива argv нам ни о чем не говорит, поэтому уточним первый аргумент программы при помощи команды print:

```
(gdb) print argv[1]
$1 = 0xbfd95244 "7"
(gdb)
Узнаем заодно значение переменной окружения USER:
(gdb) show env USER
USER = nn
(gdb)
Выполняем следующую инструкцию:
(gdb) step
17 a = 15;
(gdb)
```

Поскольку выражение в скобках оператора ветвления if является ложным, то программа сразу "перескочила" на строку 17. Эта строка еще не выполнена, и значение переменной a пока не установлено. В этом можно убедиться, если воспользоваться командой print:

```
(gdb) print a
$1 = -1208636528
(gdb)
```

Переменная a изменяется на протяжении всей программы. Полезно будет включить автоматический вывод ее значения на каждом шаге:

```
(gdb) display a
1: a = -1208636528
(gdb)
Сделаем следующий шаг:
(gdb) next
19 change_a (&a);
1: a = 15
(gdb)
```

Первая строка вывода говорит, что следующий шаг — вызов функции change_a() в строке 19. Вторая строка вывода — это отслеживаемое значение переменной a.

Чтобы "войти" в функцию change_a(), нужно выполнить команду step:

```
(gdb) step
change_a (mya=0xbf9f8170) at program2.c:5
5 *mya += 10;
(gdb)
```

Обратите внимание, что функция change_a() не видит переменную a, поэтому отслеживание ее значения временно прекратилось. Давайте посмотрим состояние стека вызовов:

```
(gdb) backtrace
#0 change_a (mya=0xbf9f8170) at program2.c:5
#1 0x080484b2 in main (argc=2, argv=0xbf9f8214)
at program2.c:19
(gdb)
```

Теперь в стеке находится два фрейма. Посмотрим значение mya:

```
(gdb) print mya
$2 = (int *) 0xbf9f8170
(gdb)
```

Это адрес, который нам ни о чем не говорит. Разыменуем указатель:

```
(gdb) print *mya
$3 = 15
(gdb)
```

Другое дело! Команда print умеет работать с выражениями. Вот как это делается:

```
(gdb) print *mya * 2 + 17
$4 = 47
(gdb)
Сделаем следующий шаг:
(gdb) step
6 }
(gdb)
```

Мы перешли к точке возврата из функции change_a(). Посмотрим значение mya:

```
(gdb) print *mya
$5 = 25
(gdb)
```

Возвращаемся в main():

```
(gdb) step
main (argc=2, argv=0xbf9f8214) at program2.c:21
21 printf ("Number: %d\n", a);
1: a = 25
(gdb)
```

Проверяем стек вызовов:

```
(gdb) where
#0 main (argc=2, argv=0xbf9f8214) at program2.c:21
(gdb)
```

Как и ожидалось, мы опять на дне стека. Выводим значение a:

```
(gdb) next
Number: 25
22 return atoi (argv[1]);
1: a = 25
(gdb)
```

И выходим из программы:

```
(gdb) continue
Continuing.
Program exited with code 07.
(gdb) quit
```

# 4. ТОЧКИ ОСТАНОВА

Наша экспериментальная программа настолько мала, что "проиграть" под отладчиком каждую ее строку нетрудно. Но в реальности программистам приходится отлаживать код, вмещающий тысячи строк. Чтобы отладка программы не превращалась в рутинную процедуру исследования каждой строки исходного кода, программисты используют точки останова (breakpoints) и точки слежения (watchpoints).

Точка останова — это условная метка (строка исходного кода или функция), по достижении которой отладчик останавливает программу и ждет дальнейших указаний. Точка слежения — это выражение, при каждом изменении которого отладчик останавливает программу. Точек останова и точек слежения можно задать сколько угодно много.

Для работы с точками останова и слежения предусмотрены следующие команды отладчика gdb:

- break (b — краткая форма) — эта команда задает точку останова. Ее аргумент — имя функции или номер строки;
- watch — задает точку слежения, принимая в качестве аргумента выражение или имя переменной;
- info break — выводит информацию о точках останова;
- info watch — выводит информацию о точках слежения;
- delete breakpoints — удаляет все точки останова.

Запустим наш пример под отладчиком еще раз:

```
$ gdb -q
(gdb) file program2
Reading symbols from
/home/nn/main/work/bhv/src/ch28/program2/program2...done.
Using host libthread_db library
"/lib/libthread_db.so.1".
(gdb)
```

Зададим точку останова на инструкции вывода значения переменной a. Для этого придется сначала просмотреть исходный код, чтобы узнать номер строки:

```
(gdb) l 15,30
15 }
16
17 a = 15;
18
19 change_a (&a);
20
21 printf ("Number: %d\n", a);
22 return atoi (argv[1]);
23 }
(gdb) break 21
Breakpoint 1 at 0x80484b2: file program2.c, line 21.
(gdb)
```

Попробуем теперь запустить программу командой run:

```
(gdb) run 4
Starting program:
/home/nn/main/work/bhv/src/ch28/program2/program2 4
Breakpoint 1, main (argc=2, argv=0xbff5c784) at program2.c:21
21 printf ("Number: %d\n", a);
(gdb)
```

Программа дошла до строки 21 и остановилась. Чтобы закончить программу, введите команду continue:

```
(gdb) continue
Continuing.
Number: 25
Program exited with code 04.
(gdb)
```

Здесь следует отметить, что завершение программы не приводит к удалению точек останова. В этом легко убедиться:

```
(gdb) info break
Num Type Disp Enb Address What
1 breakpoint keep y 0x080484b2 in main at program2.c:21
breakpoint already hit 1 time
(gdb)
```

Зададим еще одну точку останова, но уже для функции change_a():

```
(gdb) break change_a
Breakpoint 2 at 0x8048447: file program2.c, line 5.
(gdb) info break
Num Type Disp Enb Address What
1 breakpoint keep y 0x080484b2 in main at program2.c:21
breakpoint already hit 1 time
2 breakpoint keep y 0x08048447 in change_a at program2.c:5
(gdb)
```

Запустим программу еще раз:

```
(gdb) run 5
Starting program:
/home/nn/main/work/bhv/src/ch28/program2/program2 5
Breakpoint 2, change_a (mya=0xbfa98a10) at program2.c:5
5 *mya += 10;
(gdb)
```

Остановка произошла на первой инструкции функции change_a(). Если теперь дать команду continue, то программа не завершится, а остановится перед инструкцией вывода значения переменной a:

```
(gdb) continue
Continuing.
Breakpoint 1, main (argc=2, argv=0xbfa98ab4) at program2.c:21
21 printf ("Number: %d\n", a);
(gdb)
```

Повторный вызов continue завершит программу:

```
(gdb) continue
Continuing.
Number: 25
Program exited with code 05.
(gdb)
```

Добавим теперь в программу точку слежения, которая будет останавливать выполнение программы при каждом изменении переменной a. Но здесь следует учитывать, что добавление данной точки слежения возможно только при выполнении программы внутри функции main(), т. е. там, где видна отслеживаемая переменная. Чтобы нас не путали установленные ранее точки останова, удалим их:

```
(gdb) delete breakpoints
Delete all breakpoints? (y or n) y
(gdb) info break
No breakpoints or watchpoints.
(gdb)
```

Теперь запустим программу, шагнем в функцию main() и зададим точку слежения:

```
(gdb) start 18
Breakpoint 5 at 0x8048456: file program2.c, line 9.
Starting program:
/home/nn/main/work/bhv/src/ch28/program2/program2 18
main (argc=Cannot access memory at address 0x0
) at program2.c:9
9 {
```

```
(gdb) step
main (argc=2, argv=0xbf9e0a04) at program2.c:12
12 if (argc < 2) {
(gdb) watch a
Hardware watchpoint 6: a
(gdb)
Проверяем:
(gdb) info watch
Num Type Disp Enb Address What
6 hw watchpoint keep y a
(gdb)
```

Запускаем выполнение:

```
(gdb) continue
Continuing.
Hardware watchpoint 6: a
Old value = -1208603760
New value = 15
main (argc=2, argv=0xbf9e0a04) at program2.c:19
19 change_a (&a);
(gdb)
```

Отладчик остановил выполнение программы перед строкой 19 и заботливо написал старое и новое значение переменной a. Продолжаем:

```
(gdb) c
Continuing.
Hardware watchpoint 6: a
Old value = 15
New value = 25
change_a (mya=0xbf9e0960) at program2.c:6
6 }
(gdb)
```

Следующая инструкция continue также останавливает программу, потому что функция main(), содержащая переменную a, завершается:

```
(gdb) c
Continuing.
Number: 25
Watchpoint 6 deleted because the program has left the block in
which its expression is valid.
0xb7e3c26b in exit () from /lib/libc.so.6
(gdb)
```

Теперь мы можем завершить программу и выйти из отладчика:

```
(gdb) c
Continuing.
Program exited with code 022.
(gdb) quit
```

Обратите внимание, что отладчик выводит код возврата в восьмеричной форме, о чем свидетельствует нулевой префикс числа.

## ПОЛУЧЕНИЕ ДОПОЛНИТЕЛЬНОЙ ИНФОРМАЦИИ

Дополнительную информацию по использованию отладчика gdb можно получить на соответствующих страницах руководства:

- man 1 gdb;
- info gdb.

Кроме того, сам отладчик содержит встроенную интерактивную справочную систему. Просто запустите отладчик и наберите команду help:

```
$ gdb -q
(gdb) help
List of classes of commands:
aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution
without stopping the program
user-defined -- User-defined commands
Type "help" followed by a class name
for a list of commands in that class.
Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb)
```

Система помощи предложила целый ряд справочных разделов. Например, если вы хотите получить информацию о командах исследования стека, просто введите следующее "заклинание": (gdb) help stack.

## ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

Изучить работу отладчика *gdb*, выполнив отладку любой собственной лабораторной работы, содержащего обращения к функциям.

- добавить в откомпилированный файл отладочную информацию;

- запустить программу под отладчиком;
- изучить команды просмотра исходного кода программы;
- изучить задание списка аргументов программе в отладчике;
- изучить команды пошагового выполнения кода;
- изучить команды вывода информации в процессе отладки;
- изучить команды работы с точками останова.