Лабораторная работа №10

Итак, начнём. В данной лабораторной работе я рассмотрю базовые приёмы работы с отладчиком gdb. Перед запуском отладчика нужно сначала скомпилировать нужный файл с помощью флага –g и записать результат данной компиляции. Затем нужно воспользоваться командой gdb, после которой указать файл с результатом прошлой компиляции:

root@DESKTOP-5HM2HTK:~# gcc -g test.c -o Test.c

root@DESKTOP-5HM2HTK:~# gdb Test.c

GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1

…

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from Test.c...

(gdb)

Test.c:

1 #include <stdio.h>

2 #include <stdbool.h>

3

4 int fib(int n){

5 if(n==0){

6 return 0;

7 }

8 if(n==1){

9 return 1;

10 }

11 return fib(n-1)+fib(n-2);

12 }

13 int fact(int n){

14 int a=fib(n);

15 if(a==0){

16 return 0;

17 }

18 if(a==1){

19 return 1;

20 }

21 return a\*fact(n-1);

22 }

23

24 int main(){

25 int i=3;

26 int j=5;

27 int l=i+j;

28 int a=fib(l);

29 printf("%s\n", "Hello, GLaDOS");

30 int b=fact(l);

31 int d=fib(l)+l+fact(l);

32 int r=d+1+1+fib(1);

33 printf("%d\n%d\n%d\n%d\n%d\n",l,a,b,d,r);

34 }

Пройдёмся по действиям, которые мы можем осуществить в отладчике. С помощью команды list мы можем просматривать код нашей программы (строчки будут пронумерованы). Выведется до 10 строчек кода (5 вверх относительно указанного значения, указанная строчка и 4 вниз):

(gdb) list 16

11 return fib(n-1)+fib(n-2);

12 }

13 int fact(int n){

14 int a=fib(n);

15 if(a==0){

16 return 0;

17 }

18 if(a==1){

19 return 1;

20 }

Команда break позволит нам ставить точки на указанных строчках, при достижении которых работа программы остановится:

(gdb) break 30

Breakpoint 1 at 0x1240: file test.c, line 30.

Удаление “брейкпоинтов” производится при помощи команды delete, после которой идёт номер самого “брейкпоинта”.

Благодаря команде run мы можем запускать программу, которая дойдёт до первого “брейкпоинта”(либо до конца кода, если их нет):

(gdb) run

Starting program: /root/Test.c

Hello, GLaDOS

Breakpoint 1, main () at test.c:30

30 int b=fact(l);

Команда print выводит значения переменных или функций, которые находятся до “брейкпоинта”:

(gdb) print fib(13)

$7 = 233

(gdb) print a+i

$9 = 24

При помощи команды set var мы можем менять значения переменных (которые до “брейкпоинта”)(значения других переменных не изменятся):

(gdb) set var i=7

(gdb) print i

$10 = 7

(gdb) print l

$11 = 8

(gdb) print j+i

$12 = 12

(gdb) set var b

warning: Expression is not an assignment (and might have no effect)

help является аналогом man. Данная команда служит справочником, благодаря которому мы можем узнать что делает та или иная команда и как её нужно использовать.

Команда continue продолжает работу программы до следующего “брейкпоинта” либо до конца кода:

(gdb) continue

Continuing.

8

21

65520

65549

65552

[Inferior 1 (process 494) exited normally]

(gdb)

После завершения программы все изменённые значения с помощью set var сбрасываются.

Разберёмся с командами next и step. При использовании next выполнение программы продолжается и тут же останавливается на следующей строчке:

(gdb) r

Starting program: /root/Test.c

Breakpoint 1, main () at test.c:27

27 int l=i+j;

(gdb) n

28 int a=fib(l);

(gdb) c

Continuing.

Hello, GLaDOS

Breakpoint 2, main () at test.c:31

31 int d=fib(l)+l+fact(l);

(gdb) n

32 int r=d+1+1+fib(1);

Команда step работает как next, если в программе нет вызовов функций. Если же они есть, то step также идёт по каждой строчке функции:

(gdb) r

Starting program: /root/Test.c

Hello, GLaDOS

Breakpoint 1, main () at test.c:32

32 int r=d+1+1+fib(1);

(gdb) s

fib (n=8) at test.c:4

4 int fib(int n){

(gdb) s

5 if(n==0){

(gdb) s

8 if(n==1){

(gdb) s

9 return 1;

(gdb) s

12 }

(gdb) s

main () at test.c:32

32 int r=d+1+1+fib(1);

(gdb) s

33 printf("%d\n%d\n%d\n%d\n%d\n",l,a,b,d,r);

Последняя команда: backtrace. Она возвращает список запущенных функций вызвавшей программы. Список вызванных функций — это последовательность в данный момент активных вызовов функций программы. После возврата какого-либо значения в функции, “стек” поднимается по вызовам этой функции и очищает их. Так выглядит стек на некотором этапе при выполнении двух вложенных рекурсивных функций:

main () at test.c:31

31 int d=fib(l)+l+fact(l);

…

(gdb) s

fib (n=0) at test.c:4

4 int fib(int n){

(gdb) bt

#0 fib (n=0) at test.c:4

#1 0x00000000080011a0 in fib (n=6) at test.c:11

#2 0x00000000080011a0 in fib (n=7) at test.c:11

#3 0x00000000080011a0 in fib (n=8) at test.c:11

#4 0x00000000080011d1 in fact (n=8) at test.c:14

#5 0x0000000008001267 in main () at test.c:31

…

(gdb) s

5 if(n==0){

(gdb) bt

#0 fib (n=7) at test.c:5

#1 0x00000000080011d1 in fact (n=7) at test.c:14

#2 0x00000000080011fb in fact (n=8) at test.c:21

#3 0x0000000008001267 in main () at test.c:31

…

(gdb) s

fib (n=4) at test.c:4

4 int fib(int n){

(gdb) bt

#0 fib (n=4) at test.c:4

#1 0x00000000080011a0 in fib (n=7) at test.c:11

#2 0x00000000080011d1 in fact (n=7) at test.c:14

#3 0x00000000080011fb in fact (n=8) at test.c:21

#4 0x0000000008001267 in main () at test.c:31