Использование отладчика GDB

Тестовая программа

Для демонстрации примеров работы с отладчиком GDB будем использовать следующую программу.

```
1  double f(int x);
2
3  double f(int x) {
4      double z;
5      z = x + 0.4;
6      return z;
7  }
8
9  int main(void) {
10     int    x    = 15;
11     double y    = 15.5;
12     int    a[] = {1, 2, 3, 4};
13     double b[] = {1.5, 2.5, 3.5, 4.5};
14
15     y    = f(x);
16     b[0] = f(a[0]);
17     b[1] = y;
18     y    = b[2];
19
20     return 0;
21 }
```

Строки 3 – 7 содержат определение функции `f`, которая преобразует значение типа `int` к значению типа `double`, прибавляя к нему число 0.4.

Строки 9 – 21 содержат определение функции `main`. Эта функция содержит четыре переменных. Заметим, что переменные `a` и `b` представляют из себя стековые массивы. В строке 15 с помощью вызова функции `f` модифицируется значение переменной `y`. В строках 16, 17 модифицируются два элемента массива `b`.

Запуск отладчика

В исполняемый файл тестируемой программы должна быть добавлена отладочная информация. Это достигается указанием ключа `-g` во время компиляции. Например, если текст программы содержится в файле `example.c`, то команда, запускающая компилятор, будет иметь следующий вид.

```
$ gcc -g example.c -o prog
```

Отладчик запускается с помощью команды `gdb`. При этом, в качестве аргумента командной строки необходимо указать имя исполняемого файла тестируемой программы.

```
$ gdb prog
```

После запуска должно появиться приглашение `(gdb)` для ввода команд отладчика. Для запуска программы в отладчике используется команда `r`. С помощью этой команды программу можно запускать многократно. Ввод команды `q` прекращает работу с отладчиком.

```
$ gdb prog
...
Reading symbols from prog...done.
(gdb) r
...
(gdb) r
...
(gdb) q
$
```

В приведенном примере был запущен отладчик `gdb` с одновременной загрузкой исполняемого файла `prog`. После запуска отладчик выведет на экран ряд диагностических сообщений, которые в примере были заменены на многоточие. Стоит обратить внимание на последнюю строчку из этих сообщений.

```
Reading symbols from prog...done.
```

Присутствие этой строчки говорит о том, что исполняемый файл был успешно загружен. Далее, два раза была запущена программа и завершена работа с отладчиком.

Точки останова

Перед запуском программы с помощью команды `b <file>:<line>` можно задать точку останова (англ. breakpoint), по достижению которой отладчик приостановит выполнение программы. Параметр `<file>` задает имя файла с текстом программы, а параметр `<line>` задает номер строки в этом файле, которая и выступает в качестве точки останова.

С помощью команды `c` можно продолжить выполнение программы. Выполнение продолжится до достижения очередной точки останова или до завершения программы. Для пошагового (построчного) выполнения программы используется команда `n`, с помощью которой выполняется очередная строка программы.

```
$ gdb prog
...
Reading symbols from prog...done.
(gdb) b example.c:15
...
(gdb) b example.c:5
...
(gdb) r
...
(gdb) c
...
(gdb) c
...
(gdb) c
...
(gdb) q
$
```

В приведенном примере были созданы две точки останова на строке 15 внутри функции `main` и на строке 5 внутри функции `f`. После своего запуска программа была приостановлена на строке 15. Далее, с помощью команды `c` программа продолжила свое выполнение до достижения строки 5. Далее, с помощью команды `c` программа опять продолжила свое выполнение и снова остановилась на строке 5. В этом случае остановка соответствовала вызову функции `f` из строки 16. Очередной ввод команды `c` продолжил выполнение программы до ее завершения. После этого с помощью команды `q` была прекращена работа с отладчиком.

С помощью команды `i b` можно вывести список существующих на текущий момент точек останова. Выполнив эту команду, можно увидеть, что каждой точке останова присвоен уникальный числовой номер. Удалить точку останова можно с помощью команды `d <num>`, где через параметр `<num>` задается номер удаляемой точки останова.

Просмотр значений переменных

С помощью команды `p <expr>` можно посмотреть значение выражения `p <expr>`. В следующем примере предполагается, что выполнение программы было приостановлено на строке 15. Далее, осуществляется просмотр значений выражений `x`, `y`, `a[0]`, `b[0]`, `a[0] + a[3]`, `b[1] + b[2]`.

```
(gdb) p x
$1 = 15
(gdb) p y
$2 = 15.5
(gdb) p a[0]
$3 = 1
(gdb) p b[0]
$4 = 1.5
(gdb) p a[0] + a[3]
$5 = 5
(gdb) p b[1] + b[2]
$6 = 6
```

Можно специфицировать формат печати значения выражения. Команда `p/x` напечатает число в шестнадцетиричной системе счисления, а команда `p/t` – в двоичной.

```
(gdb) p/x x
$7 = 0xf
(gdb) p/t x
$8 = 1111
```

Исследование областей памяти

С помощью команды `x/NFS <address>` можно исследовать содержимое массива, расположенного по адресу `<address>`. Параметр `N` задает количество элементов массива. Параметр `F` задает тип элементов массива. Значение `d` этого параметра соответствует целым числам, а значение `f` – вещественным числам. Параметр `S` задает размер элементов массива. Значение `w` (от слова word) этого параметра соответствует 4 байтам, а значение `g` (от слова giant) – 8 байтам.

В следующем примере предполагается, что выполнение программы было приостановлено на строке 15. Далее, осуществляется просмотр содержимого массивов `a` и `b`.

```
(gdb) x/4dw a
0x7fffffffdcf0: 1       2       3       4
(gdb) x/4fg b
0x7fffffffdd00: 1.5     2.5
0x7fffffffdd10: 3.5     4.5
```