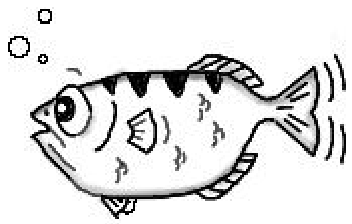
THE GEEK STUFF
Linux | DB | Open Source | Web

How to Debug C Program using gdb in 6 Simple Steps

by SathiyaMoorthy on March 15, 2010



Earlier we discussed the basics of how to write and compile a C program with C Hello World Program.

In this article, let us discuss how to debug a c program using gdb debugger in 6 simple steps.

Write a sample C program with errors for debugging purpose

To learn C program debugging, let us create the following C program that calculates and prints the factorial of a number. However this C program contains some errors in it for our debugging purpose.

```
$ vim factorial.c
# include <stdio.h>

int main()
{
        int i, num, j;
        printf ("Enter the number: ");
        scanf ("%d", &num );

        for (i=1; i<num; i++)
                j=j*i;

        printf("The factorial of %d is %d\n",num,j);
}

$ cc factorial.c

$ ./a.out
Enter the number: 3
The factorial of 3 is 12548672
```

Let us debug it while reviewing the most useful commands in gdb.

Step 1. Compile the C program with debugging option -g

Compile your C program with -g option. This allows the compiler to collect the debugging information.

```
$ cc -g factorial.c
```

Note: The above command creates a.out file which will be used for debugging as shown below.

Step 2. Launch gdb

Launch the C debugger (gdb) as shown below.

```
$ gdb a.out
```

Step 3. Set up a break point inside C program

```
Syntax:

break line_number
```

Other formats:

- break [file_name]:line_number
- break [file_name]:func_name

Places break point in the C program, where you suspect errors. While executing the program, the debugger will stop at the break point, and gives you the prompt to debug.

So before starting up the program, let us place the following break point in our program.

```
break 10
Breakpoint 1 at 0x804846f: file factorial.c, line 10.
```

Step 4. Execute the C program in gdb debugger

```
run [args]
```

You can start running the program using the run command in the gdb debugger. You can also give command line arguments to the program via run args. The example program we used here does not requires any command line arguments so let us give run, and start the program execution.

```
run
Starting program: /home/sathiyamoorthy/Debugging/c/a.out
```

Once you executed the C program, it would execute until the first break point, and give you the prompt for debugging.

```
Breakpoint 1, main () at factorial.c:10
10                      j=j*i;
```

You can use various gdb commands to debug the C program as explained in the sections below.

Step 5. Printing the variable values inside gdb debugger

```
Syntax: print {variable}

Examples:
print i
print j
print num

(gdb) p i
$1 = 1
(gdb) p j
$2 = 3042592
(gdb) p num
$3 = 3
(gdb)
```

As you see above, in the factorial.c, we have not initialized the variable j. So, it gets garbage value resulting in a big numbers as factorial values.

Fix this issue by initializing variable j with 1, compile the C program and execute it again.

Even after this fix there seems to be some problem in the factorial.c program, as it still gives wrong factorial value.

So, place the break point in 10th line, and continue as explained in the next section.

Step 6. Continue, stepping over and in – gdb commands

There are three kind of gdb operations you can choose when the program stops at a break point. They are continuing until the next break point, stepping in, or stepping over the next program lines.

- c or continue: Debugger will continue executing until the next break point.
- n or next: Debugger will execute the next line as single instruction.

- s or step: Same as next, but does not treats function as a single instruction, instead goes into the function and executes it line by line.

By continuing or stepping through you could have found that the issue is because we have not used the <= in the 'for loop' condition checking. So changing that from < to <= will solve the issue.

gdb command shortcuts

Use following shortcuts for most of the frequent gdb operations.

- l – list
- p – print
- c – continue
- s – step
- ENTER: pressing enter key would execute the previously executed command again.

Miscellaneous gdb commands

- **l command:** Use gdb command l or list to print the source code in the debug mode. Use l line-number to view a specific line number (or) l function to view a specific function.
- **bt: backtrack** – Print backtrace of all stack frames, or innermost COUNT frames.
- **help** – View help for a particular gdb topic — help TOPICNAME.
- **quit** – Exit from the gdb debugger.

If you enjoyed this article, you might also like..