How to debug C programs in Linux using gdb

On this page

Regardless of how experienced a coder you are, any software you develop can't be completely free of bugs. So, identifying bugs and fixing them is one of the most important tasks in the software development cycle. While there are many ways of identifying bugs (testing, self review of code, and more), there are dedicated software - dubbed debuggers - that help you understand exactly where the problem is, so that you can easily fix it.

If you are a C/C++ programmer or develop software using the Fortran and Modula-2 programming languages, you'll be glad to know there exists an excellent debugger - dubbed GDB - that lets you easily debug your code for bugs and other problems. In this article, we will discuss the basics of GDB, including some of the useful features/options it provides.

But before we move ahead, it's worth mentioning that all the instructions as well examples presented in this article have been tested on Ubuntu 14.04LTS. The example code used in the tutorial is written in C language; the comma line shell we've used is bash (versio 4.3.11); and the GDB version we've used is 7.7.1.

GDB debugger basics

In layman's terms, GDB lets you peek inside a program while the proc that lets you help identify where exactly the problem is. We'll discuss debugger through a working example in the next section, but before that, here, we'll discuss a few basic points that'll help you later on.

Firstly, in order to successfully use debuggers like GDB, you have to compile your program in such a way that the compiler also produces debugging information that's required by debuggers. For example, in case of the gcc compiler, which we'll be using to compile the example C program later on this tutorial, you need to use the -g command line option while compiling your code.

To know what the gcc compiler's manual page says about this command line option, head here.

Next step is to make sure that you have GDB installed on your system. If that's not the case, and you're on a Debian-based system like Ubuntu, you can easily install the tool using the following command:

```
sudo apt-get install gdb
```

For installation on any other distro, head here.

Now, once you've compiled your program in a way that it's debugging-ready, and GDB is there on your system, you can execute your program in debugging mode using the following command:

```
gdb [prog-executable-name]
```

While this will initiate the GDB debugger, your program executable won't be launched at this point. This is the time where you can define your debugging-related settings. For example, you can define a breakpoint that tells GDB to pause the program execution at a particular line number or function.

Moving on, to actually launch your program, you'll have to execute the following gdb command:

```
run
```

It's worth mentioning here that if your program requires some command line arguments to be passed to it, you can specify them here. For example:

```
run [arguments]
```

GDB provides many useful commands that come in handy while debugging. We'll discuss some of them in the example in next section.

GDB usage example

Now we have a basic idea about GDB as well as its usage. So let's take an example and apply the knowledge there. Here's an example code:

```
#include <stdio.h>

int main()
{
    int out = 0, tot = 0, cnt = 0;
    int val[] = {5, 54, 76, 91, 35, 27, 45, 15, 99, 0};

    while(cnt < 10)
    {
        out = val[cnt];
        tot = tot + 0xffffffff/out;
        cnt++;
    }

    printf("\n Total = [%d]\n", tot);
    return 0;
}
```

So basically, what this code does is, it picks each value contained in the 'val' array, assigns it to the 'out' integer, and then calculates 'tot' by summing up the variable's previous value and the result of '0xffffffff/out.'

The problem here is that when the code is run, it produces the following error:

```
$ ./gdb-test
Floating point exception (core dumped)
```

So, to debug the code, the first step would be to compile the program with -g. Here's the command:

```
gcc -g -Wall gdb-test.c -o gdb-test
```

Next up, let's run GDB and let it know which executable we want to debug. Here's the command for that:

```
gdb ./gdb-test
```

Now, the error I am getting is 'floating point exception,' and as most of you might already know, it's caused by n % x, when x is 0. So, with that in mind, I put a break point at line number 11, where the division is taking place. This was done in the following way:

```
(gdb) break 11
```

Note that '(gdb)' is the debugger's prompt, I just wrote the 'break' command.

Now, I asked GDB to start the execution of the program:

```
run
```

So, when the breakpoint was hit for the first time, here's what GDB showed in the output:

```
Breakpoint 1, main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
(gdb)
```

As you can see in the output above, the debugger showed the line where the breakpoint was put. Now, let's print the current value of 'out.' This can be done in the following way:

```
(gdb) print out
$1 = 5
(gdb)
```

As you can see that the value '5' was printed. So, things are fine at the moment. I asked the debugger to continue the execution of the program until the next breakpoint, something which can be done using the 'c' command.

```
c
```

I kept on doing this work, until I saw that the value of 'out' was zero.

```
...
...
...
Breakpoint 1, main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
(gdb) print out
$2 = 99
(gdb) c
Continuing.

Breakpoint 1, main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
(gdb) print out
$3 = 0
(gdb)
```

Now, to confirm that this is the exact problem, I used GDB's 's' (or 'step') command instead of 'c' this time. Reason being, I just wanted line 11, where the program execution currently stands paused, to execute, and see if crash occurs at this point.

Here's what happened:

```
(gdb) s

Program received signal SIGFPE, Arithmetic exception.
0x080484aa in main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
```

Yes, as confirmed by the highlighted output above, this is where the exception was thrown. The final confirmation came when I tried running the 's' command once again:

```
(gdb) s

Program terminated with signal SIGFPE, Arithmetic exception.
The program no longer exists.
```

So this way, you can debug your programs using GDB.

Conclusion

We've just scratched the surface here, as GDB offers a lot of features for users to explore and use. Go through the man page of GDB to know more about the tool, and try using it whenever you're debugging something in your code. The debugger has a bit of learning curve associated with it, but it's worth the hard work.

view as pdf | print

Share this page:

Recommend Tweet Follow G+

Suggested articles

0 Comment(s)

Add comment

Name *

Email *

B *I*

p

Submit comment

Tutorials > **How to debug C programs in Linux using gdb**

Sign up now!

Tutorial Info

Author:	Ansh
Published:	Jan 16, 2017
Tags:	linux, programming

Share This Page

Recommend Tweet Follow G+

40.2k Followers

Popular Tutorials

The Perfect Server - Ubuntu 16.04 (Xenial Xerus) with Apache, PHP, MySQL, PureFTPD, BIND, Postfix, Dovecot and ISPConfig 3.1

The Perfect Server - Debian 9 (Stretch) with Apache, BIND, Dovecot, PureFTPD and ISPConfig 3.1

How to Install Couch CMS on Ubuntu 16.04 LTS

Linux sdiff Command Tutorial for Beginners (6 Examples)

How to Install Ansible AWX on CentOS 7

How to use grep to search for strings in files on the shell

How to use the Linux ftp command to up- and download files on the shell

The Perfect Server - Ubuntu 16.04 (Nginx, MySQL, PHP, Postfix, BIND, Dovecot, Pure-FTPD and ISPConfig 3.1)

How to create Docker Images with a Dockerfile

Linux Basics - Set A Static IP On Ubuntu