

Introduction to the gdb Debugger

When a bug in your program is hard to find, use the Linux Gnu debugger gdb.

First start an interactive session in a terminal window with the gdb command, then run your program executable file within gdb. The debugger will trap segmentation faults, and allows you to set breakpoints in your code and probe values of variables.

To make use of the debugger you must add the '-g' argument to your gcc command line to have the compiler include symbolic debugging information, such as source code line numbers and variable names, in the output executable file.

Example Segmentation Fault Program 1

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
double *x;
```

```
int main(int argc, char *argv[]) {
    x[0] = 42.0;
    exit(0);
}
```

```
$ gcc -Wall -Wshadow -o segfault1 segfault1.c -lm
$ ./segfault1
Segmentation fault (core dumped)
$
```

Symbol x has been declared as pointer to double, but has not been initialized to any allocated memory before being used as an array pointer. The default value will be the NULL pointer, which points to nonexistent memory. Using this as an array pointer will result in a memory segmentation fault and abort the program.

Example Debugging Session

```
$ gcc -Wall -Wshadow -g -o segfault1 segfault1.c -lm
$ gdb segfault1
GNU gdb (GDB) Fedora 8.2-7.fc29
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./segfault1...done.
(gdb) run
Starting program: segfault1
Program received signal SIGSEGV, Segmentation fault.
0x0000000000401144 in main (argc=1, argv=0x7fffffffdb78) at
segfault1.c:8
8      x[0] = 42.0;
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 30970) killed]
(gdb) quit
$
```

Recompile the program with the '-g' option to gcc

Run the debugger with the executable as an argument

Run the program within gdb

gdb will trap the segfault signal and report the line in the source code file responsible

Example Segmentation Fault Program 2

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
double *x;
```

```
int main(int argc, char *argv[]) {
    x = (double *) malloc(10 * sizeof(double));
    x[100000] = 42.0;
    free(x);
    exit(0);
}
```

Symbol x has been declared as pointer to double, and has now has been initialized to allocated memory for an array of length 10. Trying to write into an array element way out of bounds will result in a memory segmentation fault and abort the program.

```
$ gcc -Wall -Wshadow -o segfault2 segfault2.c -lm
$ ./segfault2
Segmentation fault (core dumped)
$
```

Example Debugging Session

```
$ gcc -Wall -Wshadow -g -o segfault2 segfault2.c -lm
$ gdb segfault2
GNU gdb (GDB) Fedora 8.2-7.fc29
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./segfault2...done.
(gdb) run
Starting program: segfault2
Program received signal SIGSEGV, Segmentation fault.
0x000000000040117b in main (argc=1, argv=0x7fffffffdb78) at
segfault2.c:9
9      x[100000] = 42.0;
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 31034) killed]
(gdb) quit
$
```

Recompile the program with the '-g' option to gcc

Run the debugger with the executable as an argument

Run the program within gdb

gdb will trap the segfault signal and report the line in the source code file responsible

Example Program For A Breakpoint

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double *x;

int main(int argc, char *argv[]) {
    double a,b,c;

    a = atof(argv[1]);
    b = atof(argv[2]);
    c = atof(argv[3]);
    printf("sum = %f\n", a+b+c);
    exit(0);
}
```

This simple program takes three command line arguments with numerical values and displays their sum to standard output.

Line 13 of this source code file breakpoint1.c is this printf() statement

```
$ gcc -Wall -Wshadow -o breakpoint1 breakpoint1.c -lm
$ ./breakpoint1 1 2 3
sum = 6.000000
$
```

Example Debugging Session

```
$ gcc -Wall -Wshadow -g -o breakpoint1 breakpoint1.c -lm
```

```
$ gdb ./breakpoint1
```

```
GNU gdb (GDB) Fedora 8.2-7.fc29
```

```
Find the GDB manual and other documentation resources online at:
```

```
<http://www.gnu.org/software/gdb/documentation/>.
```

```
For help, type "help".
```

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

```
Reading symbols from ./breakpoint1...done.
```

```
(gdb) set args 4 5 6
```

```
(gdb) break breakpoint1.c:13
```

```
Breakpoint 1 at 0x4011a9: file breakpoint1.c, line 13.
```

```
(gdb) run
```

```
Starting program: breakpoint1 4 5 6
```

```
Breakpoint 1, main (argc=4, argv=0x7fffffffdb48) at breakpoint1.c:13
```

```
13    printf("sum = %f\n",a+b+c);
```

```
(gdb) print a
```

```
$1 = 4
```

```
(gdb) print b
```

```
$2 = 5
```

```
(gdb) print c
```

```
$3 = 6
```

```
(gdb) print x
```

```
$4 = (double *) 0x0
```

```
(gdb) continue
```

```
Continuing.
```

```
sum = 15.000000
```

```
[Inferior 1 (process 31305) exited normally]
```

```
(gdb) quit
```

```
$
```

Recompile the program with the '-g' option to gcc

Run the debugger with the executable as an argument

Set the command line arguments to be passed to the program within gdb

Set a breakpoint at the source code line number of the printf() statement and start the program

gdb will pause before the breakpoint statement and allow you to print values of variables, including pointers (a pointer value of hexadecimal 0 is the NULL pointer value)

After probing variable values, the program can be continued until exit