Embedded Des Enabling Robotics
Lab 2 report

Shiyu Wang
Hao Jiang
Fall 2016
23th September

Introduction

This lab further familiar us with the use of Linux system as well as to learn how to use a program debugger. In pre-lab, we did research on the concept of debugging. We also did some modify on the programs given. In lab, we used GDB debugger to see what is going on inside a program. We used GDB debugger to compile three programs and made revisions to them. We used debugging languages given in the sheet. Finally, we got the revised program and got desirable results for the inputs entered.

Lab Preparation

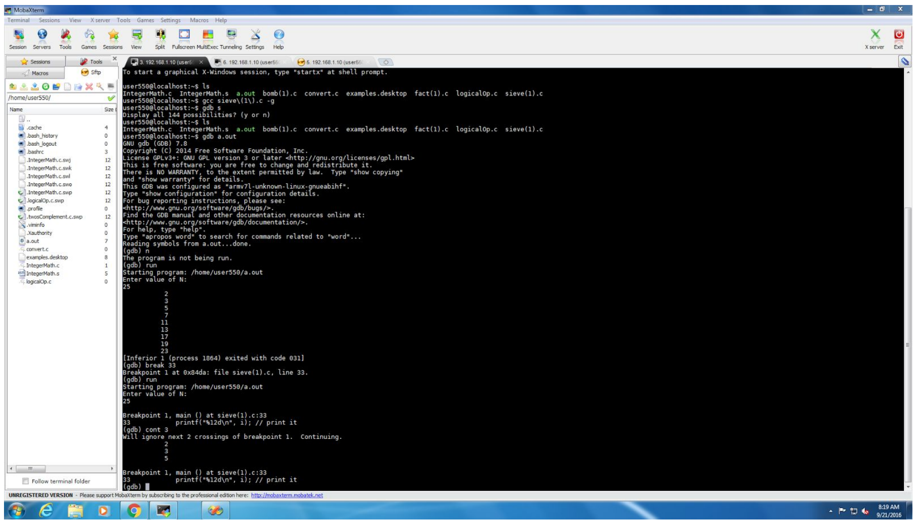
We first logged onto the computer using our credentials. We then connected the Zedboard to the PC host via Ethernet to the RJ45 connector. The ZedBoard was boosted after connected to the computer. We made an SSN connection to the ZedBoard where the IP is 192.168.1.10, and port 22.

We then downloaded the three programs posted on BlackBoard and uploaded them to the host PC using SFTP which transferred them onto the ZedBoard.

Lab 2.1 Tracing sieve.c

We observed this program for a while and learnt that this program tried to find all the prime numbers which are less or equal to the number we entered. We first compile sieve.c with <gcc sieve.c>, and then we used <gdb a.out> to enable debugging. Under gdb, we first an this program using command <run>. We entered N = 25, and expected to get 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25. However, the program only gave us only part the results we desired to have - 9, 21, and 25 itself were left out from the final output. We then used the gdb function to breakpoint to see where the program was running wrongly.

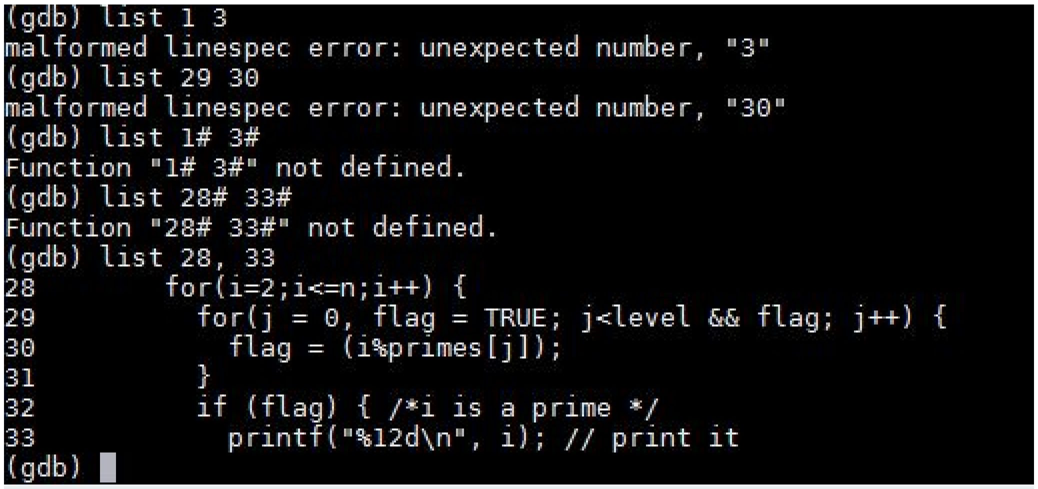
We broke at line 33 using the command <break 33>, and then again entered <run>. The program told us to enter the value for N. Again, we entered 25 for value of N. We continued the program for 3 more times using the command <cont 3> and got our very first three results: 2, 3, 5 which are correct for our purpose. According to lab manual, we then typed <print *prime@3>, and got the result: <$ 1={2, 3, 5}>. They are the first three results we expected. We think <print *prime@number> expression will give us an array of first three prime numbers.



```
To start a graphical X-Windows session, type "startx" at shell prompt.

user550@localhost:~$ ls
IntegerMath.c  IntegerMath.s  a.out  bomb(1).c  convert.c  examples.desktop  fact(1).c  logicalOp.c  sieve(1).c
user550@localhost:~$ gcc sieve\(1\).c -g
user550@localhost:~$ gdb s
Display all 144 possibilities? (y or n)
user550@localhost:~$ ls
IntegerMath.c  IntegerMath.s  a.out  bomb(1).c  convert.c  examples.desktop  fact(1).c  logicalOp.c  sieve(1).c
user550@localhost:~$ gdb a.out
GNU gdb (GDB) 7.8
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "armv7l-unknown-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
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 a.out...done.
(gdb) n
The program is not being run.
(gdb) run
Starting program: /home/user550/a.out
Enter value of N:
25
           2
           3
           5
           7
          11
          13
          17
          19
          23
[Inferior 1 (process 1864) exited with code 031]
(gdb) break 33
Breakpoint 1 at 0x84da: file sieve(1).c, line 33.
(gdb) run
Starting program: /home/user550/a.out
Enter value of N:
25

Breakpoint 1, main () at sieve(1).c:33
33            printf("%12d\n", i); // print it
(gdb) cont 3
Will ignore next 2 crossings of breakpoint 1.  Continuing.
           2
           3
           5

Breakpoint 1, main () at sieve(1).c:33
33            printf("%12d\n", i); // print it
(gdb) 
```

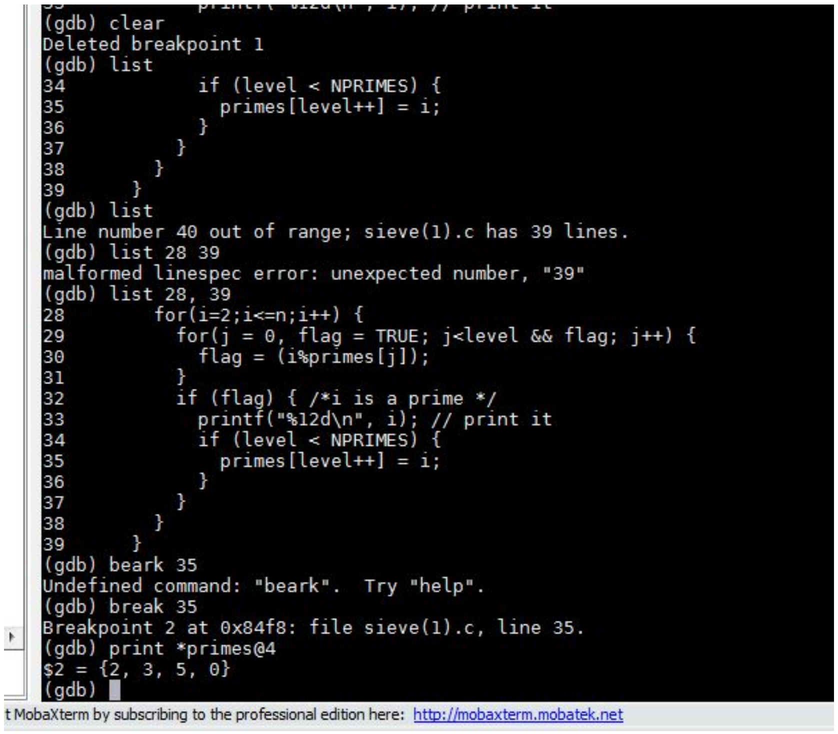
These values are all prime numbers, but are not the adjacent prime numbers. They seem to be outputs of the program. The expression <print *prime@3> seems to continue the program for 3 more times starting from the break point and print the results. The other <print *prime@100> seems to run it 100 more times and print the results. Since there are not so many results, the command simply run the program for another 91 times without outputting the results. The command <print *prime@number> seems to be the another alternative in addition to <cont number>.

We then listed the source code using the command: <list 1 3>. This gave us error, so this meant we have entered the wrong command. After trying for multiple times, we figured out we missed a comma. We then listed line 28 to line 30 with<list 28, 30>. The output was given as below.



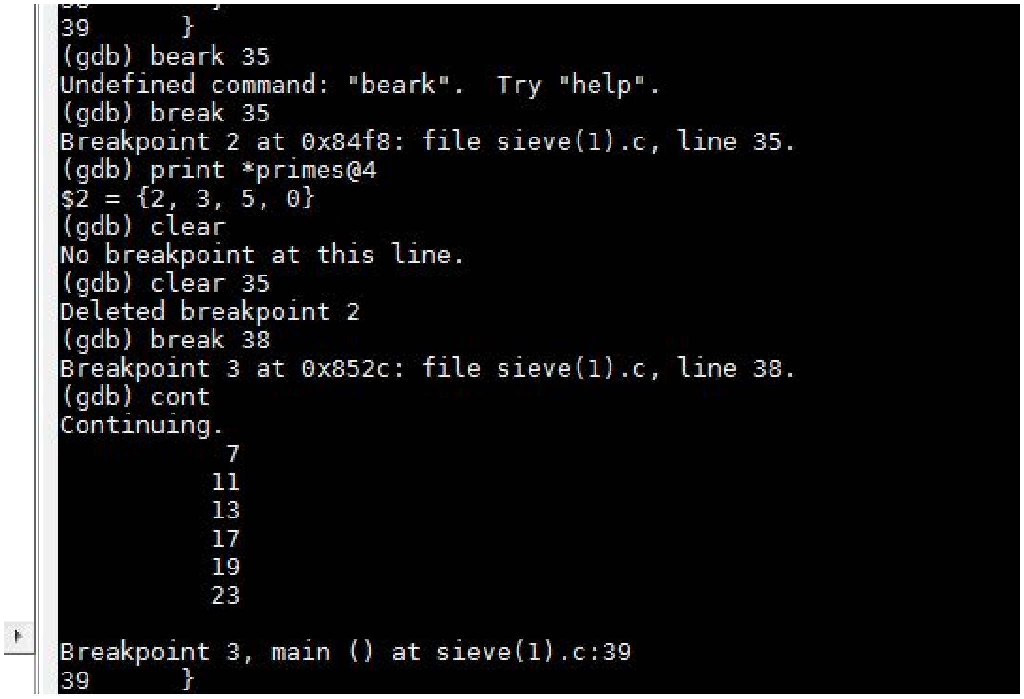
```
(gdb) list 1 3
malformed linespec error: unexpected number, "3"
(gdb) list 29 30
malformed linespec error: unexpected number, "30"
(gdb) list 1# 3#
Function "1# 3#" not defined.
(gdb) list 28# 33#
Function "28# 33#" not defined.
(gdb) list 28, 33
28        for(i=2;i<=n;i++) {
29          for(j = 0, flag = TRUE; j<level && flag; j++) {
30            flag = (i%primes[j]);
31          }
32          if (flag) { /*i is a prime */
33            printf("%12d\n", i); // print it
(gdb) 
```

We cleared the break point with command: <clear>. It deletes all the breakpoints. We then broke at line 35 where we thought the prime array was updated. Then we used another <print *prime@4>. We got: < $2= {2, 3, 5, 0}>. Again, we cleared breakpoints and set another breakpoint at line 38.



```
(gdb) clear
Deleted breakpoint 1
(gdb) list
34            if (level < NPRIMES) {
35              primes[level++] = i;
36            }
37          }
38        }
39      }
(gdb) list
Line number 40 out of range; sieve(1).c has 39 lines.
(gdb) list 28 39
malformed linespec error: unexpected number, "39"
(gdb) list 28, 39
28        for(i=2;i<=n;i++) {
29          for(j = 0, flag = TRUE; j<level && flag; j++) {
30            flag = (i%primes[j]);
31          }
32          if (flag) { /*i is a prime */
33            printf("%12d\n", i); // print it
34            if (level < NPRIMES) {
35              primes[level++] = i;
36            }
37          }
38        }
39      }
(gdb) beark 35
Undefined command: "beark".  Try "help".
(gdb) break 35
Breakpoint 2 at 0x84f8: file sieve(1).c, line 35.
(gdb) print *primes@4
$2 = {2, 3, 5, 0}
(gdb) 
```

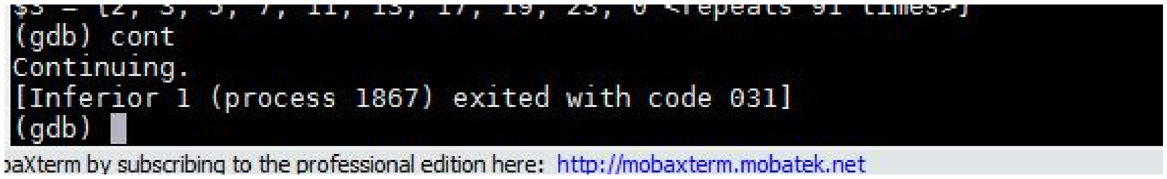
t MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net



```
39      }
(gdb) beark 35
Undefined command: "beark".  Try "help".
(gdb) break 35
Breakpoint 2 at 0x84f8: file sieve(1).c, line 35.
(gdb) print *primes@4
$2 = {2, 3, 5, 0}
(gdb) clear
No breakpoint at this line.
(gdb) clear 35
Deleted breakpoint 2
(gdb) break 38
Breakpoint 3 at 0x852c: file sieve(1).c, line 38.
(gdb) cont
Continuing.
          7
         11
         13
         17
         19
         23

Breakpoint 3, main () at sieve(1).c:39
39      }
```

We continued at line 38, and output: 7 11 13 17 19 23 was given. We <print *prime@100>, what we got was *$3 = { 2, 3, 5, 7, 11, 13, 15, 17, 19, 23, 0 <repeats 91 times> }.*

After continuing run the whole program, the system gave us the output below:

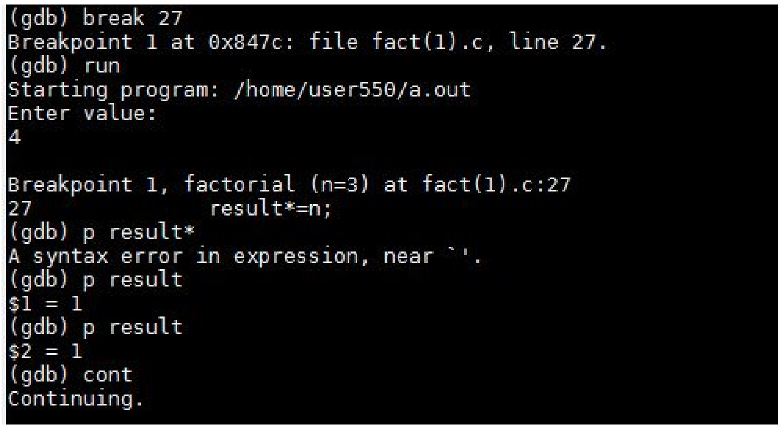


```
(gdb) cont
Continuing.
[Inferior 1 (process 1867) exited with code 031]
(gdb) 
```

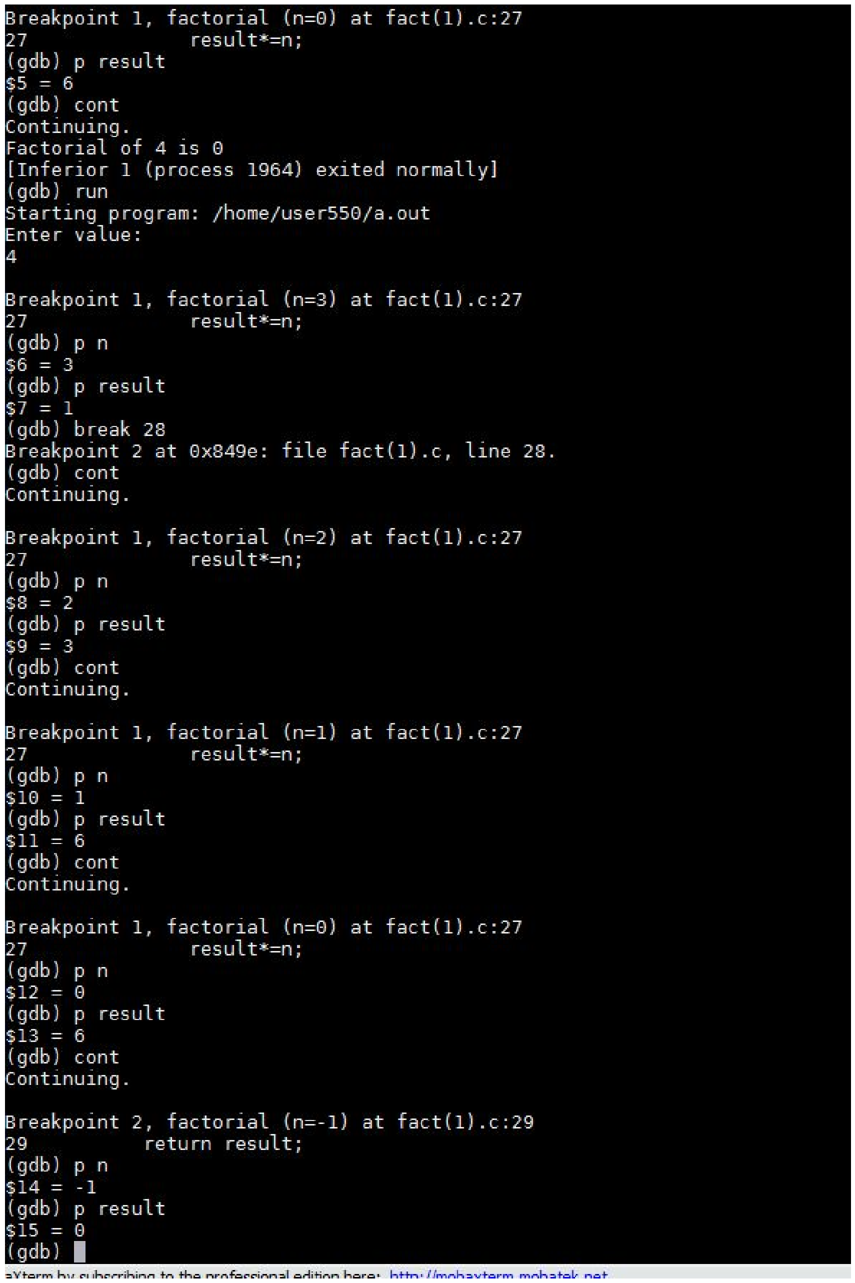
aXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

Lab 2.2 Finding logical bugs in fact.c

This program calculates n factorial. We first ran it, and found out the result was 0. We compiled it with gcc, and enabled debugging using gdb. We then set up a breakpoint at line 27 and ran it. We entered a value of 4 which is value for n. We printed the result using the expression: < p result >. We got 1 and we <cont>. We got our output n= 4, result=1. We then set another breakpoint at line 28 and continued. Following sets of outputs were n = 3, result = 4; n = 2, result = 12; n = 1, result = 24; n = 0, result = 24; n = -1, result = 0. This is where the program went wrong. We can fix the problem by writing the while loop as while (n-- && n>0).



```
(gdb) break 27
Breakpoint 1 at 0x847c: file fact(1).c, line 27.
(gdb) run
Starting program: /home/user550/a.out
Enter value:
4

Breakpoint 1, factorial (n=3) at fact(1).c:27
27                result*=n;
(gdb) p result*
A syntax error in expression, near `'.
(gdb) p result
$1 = 1
(gdb) p result
$2 = 1
(gdb) cont
Continuing.
```

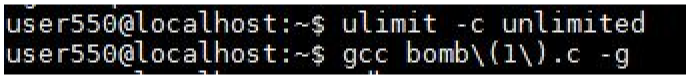


```
Breakpoint 1, factorial (n=0) at fact(1).c:27
27                result*=n;
(gdb) p result
$5 = 6
(gdb) cont
Continuing.
Factorial of 4 is 0
[Inferior 1 (process 1964) exited normally]
(gdb) run
Starting program: /home/user550/a.out
Enter value:
4

Breakpoint 1, factorial (n=3) at fact(1).c:27
27                result*=n;
(gdb) p n
$6 = 3
(gdb) p result
$7 = 1
(gdb) break 28
Breakpoint 2 at 0x849e: file fact(1).c, line 28.
(gdb) cont
Continuing.

Breakpoint 1, factorial (n=2) at fact(1).c:27
27                result*=n;
(gdb) p n
$8 = 2
(gdb) p result
$9 = 3
(gdb) cont
Continuing.

Breakpoint 1, factorial (n=1) at fact(1).c:27
27                result*=n;
(gdb) p n
$10 = 1
(gdb) p result
$11 = 6
(gdb) cont
Continuing.

Breakpoint 1, factorial (n=0) at fact(1).c:27
27                result*=n;
(gdb) p n
$12 = 0
(gdb) p result
$13 = 6
(gdb) cont
Continuing.

Breakpoint 2, factorial (n=-1) at fact(1).c:29
29            return result;
(gdb) p n
$14 = -1
(gdb) p result
$15 = 0
(gdb) 
```

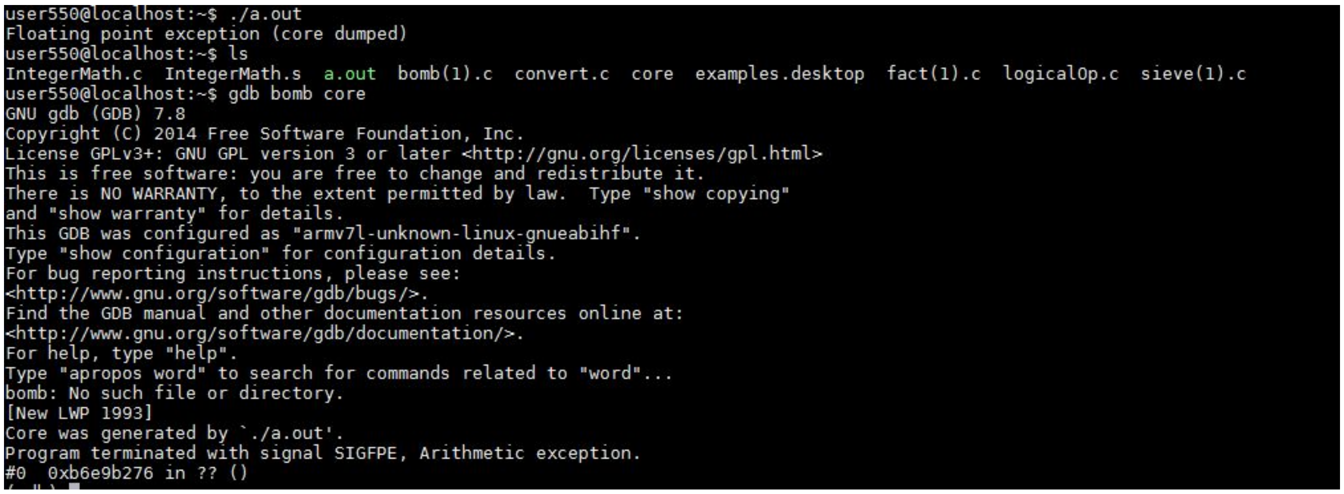
aXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

Lab 2.3 Finding the source of a crash in bomb.c

In this part, we utilized a dump from bomb.c program. We first issued <ulimit -c unlimited> on the command line. This, according to lab manual, generates a core dump. We then compiled it using gcc, and ran it using command <./a.out>.

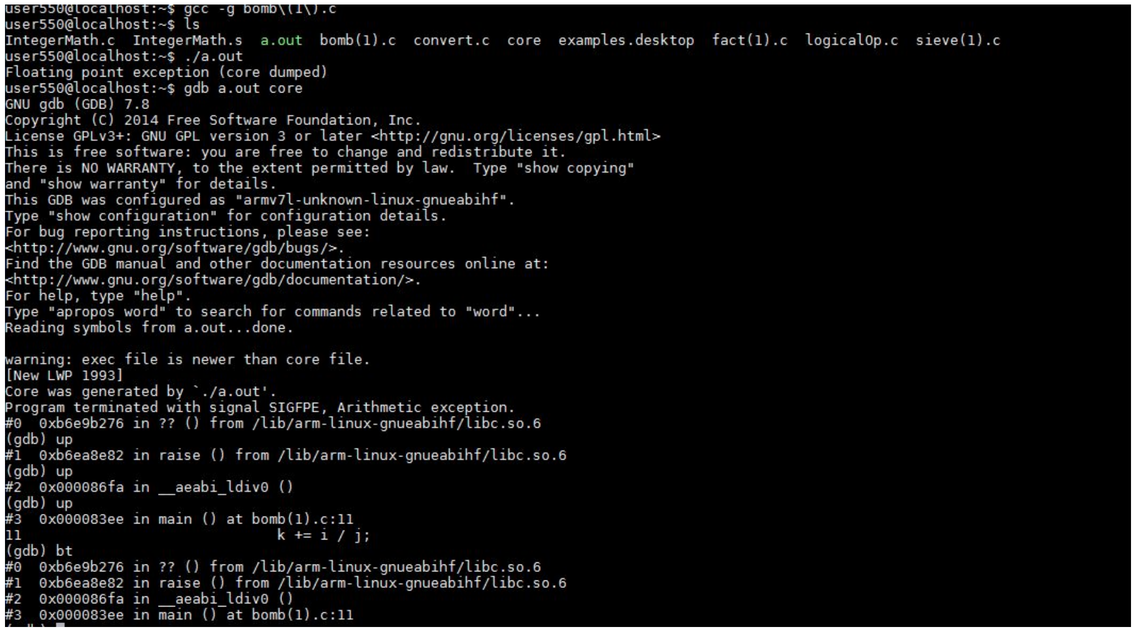


```
user550@localhost:~$ ulimit -c unlimited
user550@localhost:~$ gcc bomb\(1\).c -g
```



```
user550@localhost:~$ ./a.out
Floating point exception (core dumped)
user550@localhost:~$ ls
IntegerMath.c  IntegerMath.s  a.out  bomb(1).c  convert.c  core  examples.desktop  fact(1).c  logicalOp.c  sieve(1).c
user550@localhost:~$ gdb bomb core
GNU gdb (GDB) 7.8
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "armv7l-unknown-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
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"...
bomb: No such file or directory.
[New LWP 1993]
Core was generated by `./a.out'.
Program terminated with signal SIGFPE, Arithmetic exception.
#0  0xb6e9b276 in ?? ()
```

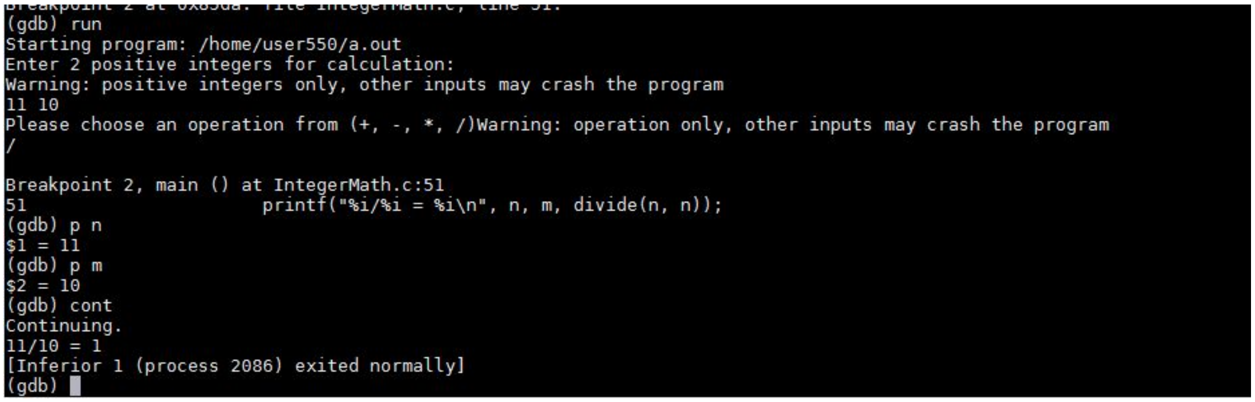
As can be shown, a core dump was generated. We then <gdb bomb core>. The last line we got was as described as in the manual, and it means we succeeded this core dump generation. We then used <up> and <down> commands to get to the line in bomb that actually caused the abnormal program exit. We <backtrace> at the position where we couldn't go up. As is shown, #0 to #3 stake frames are involved. We can see it was line 11 which caused the program to exit abnormally.



```
user550@localhost:~$ gcc -g bomb\(1\).c
user550@localhost:~$ ls
IntegerMath.c  IntegerMath.s  a.out  bomb(1).c  convert.c  core  examples.desktop  fact(1).c  logicalOp.c  sieve(1).c
user550@localhost:~$ ./a.out
Floating point exception (core dumped)
user550@localhost:~$ gdb a.out core
GNU gdb (GDB) 7.8
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "armv7l-unknown-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
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 a.out...done.

warning: exec file is newer than core file.
[New LWP 1993]
Core was generated by `./a.out'.
Program terminated with signal SIGFPE, Arithmetic exception.
#0  0xb6e9b276 in ?? () from /lib/arm-linux-gnueabihf/libc.so.6
(gdb) up
#1  0xb6ea8e82 in raise () from /lib/arm-linux-gnueabihf/libc.so.6
(gdb) up
#2  0x000086fa in __aeabi_ldiv0 ()
(gdb) up
#3  0x000083ee in main () at bomb(1).c:11
11                              k += i / j;
(gdb) bt
#0  0xb6e9b276 in ?? () from /lib/arm-linux-gnueabihf/libc.so.6
#1  0xb6ea8e82 in raise () from /lib/arm-linux-gnueabihf/libc.so.6
#2  0x000086fa in __aeabi_ldiv0 ()
#3  0x000083ee in main () at bomb(1).c:11
```

Lab 2.4 Debugging IntegerMath.c

This section is a practice and needs collaboration between lab partners. One of us introduced the error, and the other one tried to find it. One minor error was introduced. It could easily be corrected, but it could also be very hard and tricky to find. First, Shiyu put a bug on "divide operation". First, Hao ran the program few times, and found the problem is happening only on division operation. A breakpoint was introduced by Hao at the critical position before and after the division function. Then a <run> command was introduced to find out what went wrong.



```
Breakpoint 2 at 0x85da: file IntegerMath.c, line 51.
(gdb) run
Starting program: /home/user550/a.out
Enter 2 positive integers for calculation:
Warning: positive integers only, other inputs may crash the program
11 10
Please choose an operation from (+, -, *, /)Warning: operation only, other inputs may crash the program
/

Breakpoint 2, main () at IntegerMath.c:51
51                      printf("%i/%i = %i\n", n, m, divide(n, n));
(gdb) p n
$1 = 11
(gdb) p m
$2 = 10
(gdb) cont
Continuing.
11/10 = 1
[Inferior 1 (process 2086) exited normally]
(gdb) 
```

After using gdb to run few steps, Hao found the problem is the variables. It is supposed to using n / m, but Shiyu made it as n / n.

Then we swapped our roles, Hao changed the print call in main function, which is hidden well from the code, because everything else looks correctly. So, after putting many print calls to make sure every variable is working correctly, Shiyu found the problem.

Lab 2.5 Extra credit

There are many tools to Trace/Debug on Linux. According to valgrind.org, Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. (Valgrind.org). Ftrace is the Linux kernel internal tracer; Affinic Technology's Affinic 0.5.3 (recently updated to version 1.0) is a commercial GUI wrapper for GDB. It can be downloaded and tried out for free. Also, DDD 3.3.12 is the GNU project's standard GUI front end to gdb and its other language debuggers.