Week 8 - Debugging

Goals

By the end of this lab you should:

- Have an understanding of gdb and valgrind and when to use them
- Understand how to trace a SegFault
- Never have to use a bunch of cout's in your code for debugging!

Collaboration policy:

For this lab, collaboration is allowed and encouraged. Feel free to talk together about gdb and valgrind and share helpful facts you find or help your fellow students out. However, do NOT simply give them answers, try to lead them to the solution!

Part 1 (valgrind):

Today you're going to be learning about two debugging tools, gdb and valgrind. Valgrind is by far the easier of the two, as it requires no thought on your part. However, it will only catch memory leaks and segfaults, making it less powerful than gdb.

Compiling:

Create a program that will always segfault (what is the easiest way to make this happen?). When we are using debugging tools like gdb or valgrind on our code, we always need to compile with the **-g** flag, so compile your code like:

```
g++ -g segfault.cpp -o segTest
```

Valgrind:

Valgrind is used on the *actual program*, not on the source code. To use valgrind, simply type valgrind and then run your program like you would normally:

```
valgrind ./segTest
```

NOTE: if you're using command line arguments, you'll need to pass them in here:

```
valgrind ./a.out arg1 arg2
```

You should see something similar to:

```
==1114== Memcheck, a memory error detector
==1114== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et
al.
==1114== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for
copyright info
==1114== Command: ./a.out
==1114==
==1114== Invalid write of size 4
==1114==    at 0x4004FD: main (test.cpp:3)
==1114==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==1114==
==1114==
==1114== Process terminating with default action of signal 11
(SIGSEGV)
```

```
==1114==  Access not within mapped region at address 0x0
==1114==    at 0x4004FD: main (test.cpp:3)
==1114==  If you believe this happened as a result of a stack
==1114==  overflow in your program's main thread (unlikely but
==1114==  possible), you can try to increase the size of the
==1114==  main thread stack using the --main-stacksize= flag.
==1114==  The main thread stack size used in this run was 8388608.
==1114==
==1114== HEAP SUMMARY:
==1114==     in use at exit: 0 bytes in 0 blocks
==1114==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==1114==
==1114== All heap blocks were freed -- no leaks are possible
==1114==
==1114== For counts of detected and suppressed errors, rerun with: -v
==1114== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from
0)
```

We can see that there was “Invalid write….at..test.cpp:3”. BAM! We found our segfault! For me it was on line 3 of my main when I tried to write to an invalid spot in memory. The rest of the summary details our memory usage and whether or not we have any leaks (the HEAP SUMMARY). Luckily we don’t have any leaks, “All heap blocks were freed -- no leaks are possible.”

Create a program with a memory leak and use valgrind on it!

Try running valgrind on your old assignments to see if there are any leaks Zybooks missed!

Part 2:

Valgrind is as simple as running the command real quick to see if you have any leaks or segfaults, but it can't find any other problems than those two. For ALL our other debugging, we turn to gdb.

Your program not doing what you think it should? gdb!

Some random logic error you can't track down? gdb!

You need to trace through your code but you really don't want to do it by hand because your dumb SI leader already made you do that last week? gdb!

Head over to here for some gdb practice.