CSSE2310

...

Debugging and GDB

Week 5, Sem 1, 2020

What is gdb

- gdb (the GNU debugger) is a useful tool for debugging programs
- It allows you to perform helpful debugging steps such as:
  - stepping through the execution of your programs
  - printing variables at particular points in code
  - finding the point of origin of segfaults
  - and more....

How to run a program in gdb

- In order to be able to run a program in gdb, the program needs to have debugging information added into it.
- Debugging information can be added by adding the `-g` flag into your `gcc` command
  - e.g. `gcc -Wall -pedantic -std=c99 -g push2310.c`
- To run your program in gdb, you need to start the debugger with your program and use the `run (r)` command to execute an instance of it

```
$ gdb push2310

(gdb) run 1 H tests/board1
```

Finding segfaults… the easy way

- You can use gdb's `backtrace (bt)` command to see the call stack at a particular location in execution.
- This allows you to easily locate the source of a segfault in your program,

```
$ gdb orig
Reading symbols from /home/students/s4436755/tutoring/csse2310/2019/pracs/week05/orig...done.
(gdb) run
Starting program: /home/students/s4436755/tutoring/csse2310/2019/pracs/week05/orig
Program received signal SIGSEGV, Segmentation fault.
0x00000000004005dd in crash () at orig.c:8
8               *p=0;
(gdb) backtrace
#0  0x00000000004005dd in crash () at orig.c:8
#1  0x00000000004005fa in boom (i=0) at orig.c:14
#2  0x0000000000400613 in C () at orig.c:20
#3  0x0000000000400635 in B () at orig.c:25
#4  0x0000000000400640 in A () at orig.c:30
#5  0x00000000004006b9 in main (argc=1, argv=0x7fffffffddd8) at orig.c:56
```

Read from bottom to top
(e.g. main called A, A called B, etc.)

Moving through the call stack

- You can jump to a particular point in the call stack using the up and down commands
  - E.g. in the below example we start at #0, up 3 will move us to B( ) (#3), down 2 will then move us to boom( ) (#1)

```
$ gdb orig
Reading symbols from /home/students/s4436755/tutoring/csse2310/2019/pracs/week05/orig...done.
(gdb) run
Starting program: /home/students/s4436755/tutoring/csse2310/2019/pracs/week05/orig
Program received signal SIGSEGV, Segmentation fault.
0x00000000004005dd in crash () at orig.c:8
8               *p=0;
(gdb) backtrace
#0  0x00000000004005dd in crash () at orig.c:8
#1  0x00000000004005fa in boom (i=0) at orig.c:14
#2  0x0000000000400613 in C () at orig.c:20
#3  0x0000000000400635 in B () at orig.c:25
#4  0x0000000000400640 in A () at orig.c:30
#5  0x00000000004006b9 in main (argc=1, argv=0x7fffffffddd8) at orig.c:56
```

Moving through your program

- You can add a *breakpoint* into your program to cause gdb to pause execution at that point so that you can manually step through from there
- To add a breakpoint, you can use the `break` command
  - You can break on a function (e.g. `break main`)
  - You can break on a line number (e.g. `break main.c:19`)
  - You can add conditions to breakpoints as well (e.g. `break main.c:44 if (i == 10)`)
- Once you reach a breakpoint, you can manually move through your code
  - Use `next (n)` to execute the current line being displayed
  - Use `step (s)` to step into a function
    - Note that step is the same as next if there is no function call on that line)
- You can continue normal execution using the `continue (c)` command

Viewing and removing breakpoints

- You see a list of breakpoints using `info breakpoints`

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004005dd in crash at orig.c:8
        breakpoint already hit 1 time
```

- You can remove a breakpoint using `delete breakpoints`
  - e.g. `delete breakpoints 1` OR `delete b 1`

Viewing code and printing values

- Variables and function results can be displayed using the `print` command
  - `print argc`
  - `print atoi(“13spider”);`
- Blocks of code can be displayed using the `list` command
  - You can list code at the current position (e.g. `list`)
  - You can list code at a given function (e.g. `list main`)
  - You can list code at a given line number (e.g. `list main.c:40`)

That’s great and all but… how do I exit gdb?

- Ctrl+C will not exit gdb. Instead it will pause your program’s execution.
- To exit gdb, you can use the `quit (q)` command

Practice Exercises

- Copy the following file into your directory
  - `cp ~uqjfenw1/public/debug/pasc .`

1. What line causes the program to segfault when it is run with no arguments?
2. What is the value of v(35, 19)?