An Intro to GDB and Debugging in C

Presented by HKN

Checking out the Code:

Log into your EWS machine and type "git clone git://github.com/mtischer/hkn-debug-tutorial-1-spring-12.git"

Methods of Debugging

- printf
- Source-level debuggers, such as gdb
- Logging
- Or combinations of the above

General Issues with Debugging

- Intrusiveness: if code depends on timing in some way (e.g. code that reads time in some way, parallel interacting threads, processes that communicate with each other, etc.), stepping through it in a debugger might not be so useful. Large amounts of I/O also can impact these types of programs.
- Amount / type of information available
- Confounding – you might have several bugs, and the combined symptoms make them harder to diagnose than if they occurred individually

Issue's cont'd: Determinism

- Program execution is determined by all inputs to the program
- Obvious inputs:
  - Input files, user input (standard in), network data
- Less-obvious / problematic inputs:
  - Timing info, scheduling decisions (parallel programming), random number generation

What is a Segfault?

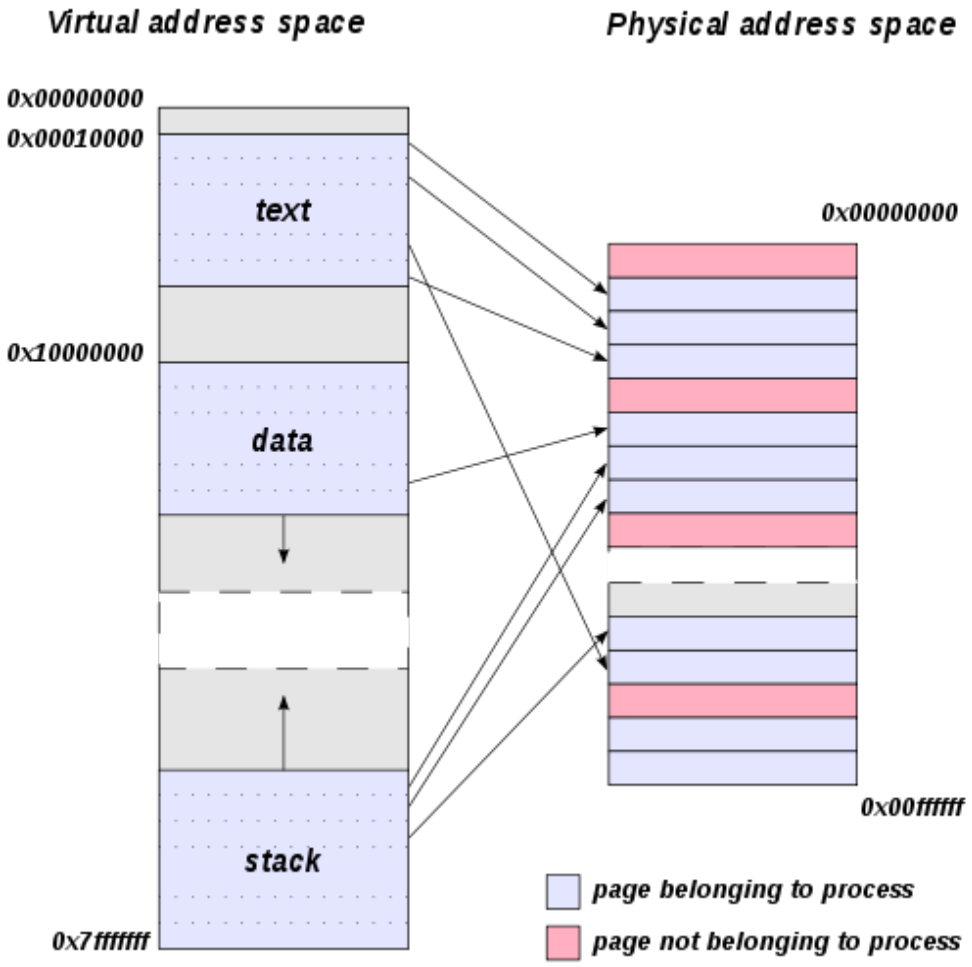
- LC-3 doesn’t have them... so why do we need them in Linux?
- A segfault is a Unix signal.
- Signals are like user-space interrupts and exceptions.

Causes of a Segfault

- Processor exceptions: page faults / segmentation faults (invalid memory accesses), privilege mode violations (e.g. illegal instructions)
- Not all page faults cause segfaults
  - e.g. swap file usage
- Not all incorrect memory accesses cause page faults. Segfaults are a best effort mechanism!

Causes of Segfaults Cont'd

- Not all incorrect memory accesses cause a page fault
- Because of the way the paging mechanism works (discussed in ECE 391)
- A page fault will only occur if the program accesses space that isn't allocated to it
- …So the program is free to corrupt its own stack, heap, etc.



Common Causes

- Buffer overflow (array misuse)
- Dereferencing uninitialized and null pointers
- Stack overflow
- Using memory after it is free'd (somewhat like using uninitialized pointers)

Ways to use Debuggers

- Run your program from within a debugger
- Examine core dumps
- Remote debugging (taught in ECE 391)
- Attaching debugger to currently running process (Visual Studio can do this)

Getting Started with GDB

First, you must compile/link your program with debugging symbols

- Use -g flag (gcc, g++ and clang)

Then run your program in gdb

- gdb ./my_program (or gdb, then file <my_program>)
- run v. start commands - give them the programs command line parameters
- -command "file" lets you send gdb commands from a file (useful when you want to always set certain breakpoints/things to display).

Exercise: printargs

- This program is designed to print out a list of arguments passed into it.
- Try to load it in GDB and use the "run <arguments>" command to correctly pass arguments to the program.

Stepping Through code

- step (s) and next (n) commands - both try to execute until the next line of code, but step will take you into a function call while next will not
- stepi and nexti - like step and next, but for a single instruction (probably won't use until 391)
- continue (c) command will run your program until it hits a breakpoint, or until it exits or receives a signal

Ending the program

- quit (q) to quit gdb. If your program is still running, it will ask you to confirm.

- kill ends the program you are debugging. You can run it again (or another program) if you wish.

Examining Variables and the Stack

- list (l) to see current code
- print (p) - can evaluate any C expression (no function calls)
- display / undisplay
- call - lets you call a function
- backtrace (bt) shows the sequence of function calls up to the current point
- frame - lets you switch to other stack frames, to examine variables of other active functions

Exercise: arewethereyet

- If you look at this program's source, you might not be able to find any problems. Yet, the program still segfaults. Why?
- Hint: the backtrace command in GDB is very useful for determining what causes a particular segfault.

Breakpoints

-Breakpoints are a mechanism which stop your program before executing a particular line of code

- break (b) <location>

  b <source_file>:<line number> or b <function_name>

- info breakpoints shows you all current breakpoints

- delete (d) <breakpoint number>

Conditional Breakpoints

- condition <breakpoint number> <condition>
- Will only break if the condition is true
- Useful if you have a piece of code that works most of the time

Watchpoints

-Instead of breaking on a particular line of code, can break when a particular piece of data changes, e.g. a global variable

- watch <variable>

Debugging with Core Dumps

- Enable core dumps by running "ulimit -c <max core file size>" (the size could be "unlimited")
- Run your program. If it crashes (e.g. segfaults), it will produce a binary file named "core.<pid>"

-gdb <core file>, then run the command "file <executable>"

More Exercises

- Try debugging fibonacci and square_ints first.
- For a challenge, try to figure out what's going wrong in string_modification and floating_point.

Still have problems?

- help command from within GDB.
- Google.
- Prof. Lumetta's GDB Tutorial / GDB reference sheet: Google "ECE 391", go to "Tools, References, and Links", then look in the section labelled "gdb".