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/HKNTutorials/gdb-intro.git
cd gdb-intro

Intro to Debugging

- First approach: print everything.
- Better way: gdb
 - Interactive
 - Watch program execute
 - See function's callers

Getting Started with GDB

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

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

Then run your program in gdb

gdb ./my_program

Run and Start

- Two different ways to begin execution of your program.
- run <command arguments> executes the program until you hit a breakpoint (or a segfault). It does not stop GDB at all, so you will not be able to do anything unless you set breakpoints.
- start <command arguments> stops GDB at the first line of code to be executed, so you can send GDB commands.

Exercise: printargs

- This program prints out the 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.
- 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 (1) to see current code
- print (p) can evaluate any C expression (no function calls)
- display / undisplay
- backtrace (bt) shows the sequence of function calls up to the current point

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>
 removes a breakpoint.

Exercise: exp

This program is supposed to calculate the exponential function via it's taylor series $e^x = 1 + x + (x^2)/2 + ... + (x^n)/n! + ...$

But instead all the results it gives are infinite. Try to debug it: set a few breakpoints, and see what it does.

Exercise: square_ints

This segfaults. So before you try this, we need to talk about what a segfault is.

What is a Segfault?

- A segfault is an exception that is typically caused by incorrectly accessing memory.
- Not all memory usage bugs cause page faults. Segfaults are a best effort mechanism - they aren't guaranteed to occur.

The Unix Memory Model

- In LC3, programs have unrestricted access to memory. In x86 / Linux, this is no longer true.
 - Some regions of address space are allocated for program use (can access).
 - Some regions of address space are allocated for OS use (OS code can access, your programs can't - will segfault). Typically includes the lowest and highest addresses.
 - Some regions are unallocated OS may allocate this on demand. Accessing this memory when the OS doesn't allow it gives a segfault.
- You are free to mess up your own program's memory

Common Causes of a Segfault

- Out-of-bounds access to array
- Dereferencing uninitialized, null, or already-freed pointers
- Stack overflow (functions calling themselves too many times)

More Exercises

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

Still have problems?

- help command from within GDB
- Google
- Get Prof. Lumetta's GDB reference sheet

Appendix: Extra Information

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 change it's behavior. Large amounts of I/O also can impact these types of programs.
- Amount of info available to you
- Confounding: you might have several bugs, and the combined symptoms make them harder to diagnose than if they occurred individually

Issues: Determinism

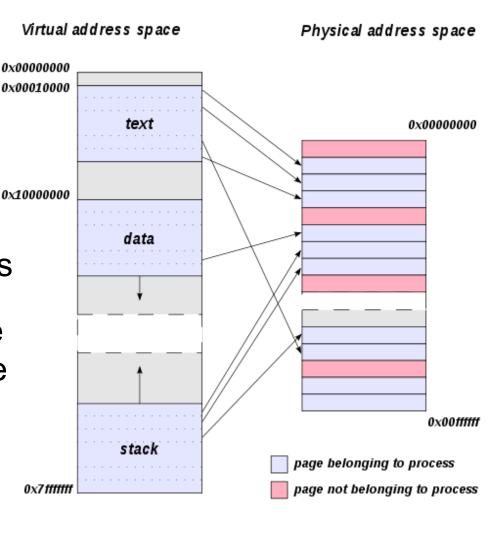
- Program execution is determined by all inputs to the program - If you can capture all inputs, you should be able to reproduce all bugs
- Obvious inputs:
 - Input files, user input (standard in), network io
- Less-obvious or problematic inputs:
 - Timing info, scheduling decisions (parallel programming) - this is especially difficult to handle
 - seeds for random number generation

Ways to use Debuggers

- Most common: run your program from within a debugger
- Postmortem debugging Examine core dumps (first run "ulimit -c unlimited")
- Remote debugging (taught in ECE 391)
- Attaching debugger to currently running process

Causes of Segfaults

- Segfaults are usually caused by page faults
- Not all incorrect memory accesses cause a page fault
- Because of the way the paging mechanism works (this is architecture dependent)
- A page fault occurs if the program accesses space that isn't allocated to it
- Programs are free to do anything with their memory
- ECE 391 covers the details



Picture Copyright © 2012 en:User:Dysprosia