An Intro to Debugging, Part 2

Presented by Eta Kappa Nu
University of Illinois at Urbana-Champaign
April 28, 2012

Setup

- In your EWS home directory, run the command:

  git clone git://github.com/dgoldstein0/hkn-debug-tutorial-2-spring-12.git <new name>
- This will create a folder called <new name> that holds all of the files and information in this tutorial (and a copy of the presentation).
- Move into that directory once you have checked it out.

A Small Amount of Review

- Debuggers (such as GDB) help simplify and speed up the debugging process.
  - We can view the values of variables from within code and watch what functions execute.
  - They give us more data!

Commonly Used GDB Commands

- (b)reak <location>: Sets a breakpoint (point where you want GDB to stop) on either a given line of code or a function name.
  - Example 1: *break test.c:123*
  - Example 2: *break add_a_ball*
- info breakpoints: Gives information about all the breakpoints.
- (d)elete <breakpoint number>: Deletes the given breakpoint.

Commonly Used GDB Commands

- (r)un: Starts running the given executable.
- Note that you usually want to set a breakpoint before you run, or the code will just execute to completion (you won't really be able to learn much).
  - One exception is when you want to determine why a segfault happens: gdb will stop automatically when you make the code segfault.
- bt (backtrace): Gives information about “where you are” in the code (what functions were called).
  - Really useful for determining what functions caused a segfault.

Commonly Used GDB Commands

- (n)ext: Moves to the next line of code. If the current line of code is a function call
- (s)tep: Moves to the next line of code. If the current line of code is a function call, steps into that function call.
- (c)ontinue: Runs the code until the next breakpoint is reached, or the code finishes executing (or crashes!).
- (f)inish: Runs the program to the end of the current function it's in.

Commonly Used GDB Commands

- display <variable>: Shows the value of a given variable every time the debugger stops.
- print <variable>: Shows the value of a given variable once.
- (q)uit: Quits gdb.

A New Tool: Valgrind

- Allows you to examine memory more closely.
- Most useful for discovering/fixing memory leaks (GDB won't help you there!).
- Usage: “valgrind --leak-check=full <program>”
- Your ultimate goal is to get “0 errors from 0 contexts” when you run valgrind on your program (ignore suppressed ones).
- Note: valgrind won't always display line numbers with clang-compiled executables! (we're using gcc here).

What is a Memory Leak?

- When a program allocates (dynamic) memory with malloc() without freeing it with free().
- The system can't reclaim that memory while the program is running, because it sees it as “in use”.
- Net result: that memory is effectively “lost” to the system.
- The real problem: keep leaking memory (bit-by-bit) until you run out!

Example 1: This Program Works... Right?

- Make and run the program “nofrees”. The program runs completely fine in the terminal (and in GDB), but it fails Valgrind horribly... why?
- Usage: “valgrind --leak-check=full <program>”
- How would we fix this problem?

Example 2: declarearray

- Look at the source code for declarearray – it seems perfectly fine!
- However, the code segfaults. Why?
- (Either valgrind or GDB will probably work here.)

An Advanced GDB Trick

- Command files: for when you want to run a pre-defined set of commands when you start GDB.
  - Usage: “gdb -command "file” <program>”
  - Useful for when you know that an error is in a certain section of code (saves time typing).
  - Try it with the file “init.gdb” (adds a breakpoint).

Advanced Valgrind Tricks

- There are lots more options!
  - --show-reachable=yes
  - --track-fds=yes
  - --track-origins=yes

Common Mistakes

- Memory leaks: more malloc()'s than free()'s.
- Double frees: Calling free() on something that's already been freed (sometimes leads to a segfault).
- Array index errors: Dereferencing the wrong index (typically accessing array[x] on an array of size x—arrays begin at index 0!).
- Pointer errors: Dereferencing an object that has already been freed (be especially careful when you point two pointers to the same object).

Exercises

- We have provided two additional exercises: reverselist and arraymanipulation.
- Try them out, and let us know if you have questions!

Thank You!