Debugging and gdb

UNX511 Week 3 Class 2

John Sellens

September 5, 2025

Seneca Polytechnic

Outline

Debugging and gdb

Debugging and gdb

Introduction to Debugging

- I always write perfect programs, so I don't debug. Ever.
- (pause for laughter)
- Programs don't always behave – first time through or much later
- Debugging is the process of finding and fixing problems in software
- You're likely already familiar with this processs
  - But let's be explicit and specific for just a bit
- https://en.wikipedia.org/wiki/Debugging

Programming, Being Clever, and Debugging

"Write clearly - don't be too clever."
– The Elements of Programming Style (Kernighan & Plaugher)

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
– (attributed to) Brian W. Kernighan (unconfirmed)

Why and When Do We Debug?

- When developing, we often build programs iteratively
  - A step or section at a time
- We may want to check on program state
  - As a way to confirm that we're on the right track
  - Or as a way to understand why we're on the wrong track
- Problems may be discovered after the program is "complete"
- Debugging is usually an ongoing process

How Do We Debug?

- Stare at the code and try to identify problems
- Adding printing or logging calls to provide state information
  - e.g. `printf( "at function start my var is '%s'\n", myvar );`
  - To stdout, stderr, a log file, or `syslog(3)`
  - Not very sophisticated, but I do this all the time
  - Harder to do with background services, and web pages
- Automated testing can help narrow down problems
  - For some programs, writing tests can be time consuming
- Or, use a debugger

What is a Debugger?

- A debugger is a software tool that can probe into a (running) program
- Provides information about code paths, variable values
- Typically lets you stop a running program at specified “breakpoints”
- And then poke around in the current state
- Debuggers can sometimes investigate after a failure/crash

What is a Core Dump?

- If a program crashes badly enough, it can “dump core”
  - Save program memory state to a file
  - Early computer RAM was called “core memory”
  - In the early days, core dumps were often (sometimes?) printed, on large amounts of paper (yikes!)
- A debugger, like `gdb`, can use the program source, binary, and core dump to shed light on what happened
- https://en.wikipedia.org/wiki/Core_dump

Enabling Core Dumps on Ubuntu

- In the early decades of UNIX, a crashed program would just write a `core` file on disk
- That was useful for developers, but not useful for users
- Eventually core dumps were (effectively) disabled by default
- To enable on Ubuntu Linux:
  - `sudo apt install systemd-coredump`
  - In your shell – for this session only:
    `ulimit -S -c unlimited`
  - Use the `coredumpctl` command to access core dumps

Using gdb and coredumpctl

- Compile your code with the `-ggdb` option
- Enable core dumps: `bash% ulimit -S -c unlimited`
- Run a command, see it dump core: `bash% ./Math`
  - Hint: try converting Celsius to Fahrenheit

```
% coredumpctl list
% coredumpctl info -1
% coredumpctl debug Math
% gdb Math
% coredumpctl -o core dump Math
% gdb Math core
```

gdb Basics

- You can do many things with `gdb`
- I know how to do the bare (but useful) minimum

```
% gdb Math
(gdb) break CelsiusToFahrenheit
(gdb) run
(gdb) backtrace
(gdb) print celsius
(gdb) step
(gdb) continue
```

gdb References

- https://en.wikipedia.org/wiki/GNU_Debugger
- https://www.sourceware.org/gdb/
  GDB: The GNU Project Debugger
- https://web.eecs.umich.edu/~sugih/pointers/summary.html
  GDB Tutorial
- https://www.cs.cmu.edu/~gilpin/tutorial/
  Debugging Under Unix: gdb Tutorial

Summary

- Programming is fun!
- Debugging is (might be) less fun!
- `gdb` is useful!
- Don’t be too clever!