CS221
C and Systems Programming

Types of errors

Code does not compile or link

Code runs but crashes

Code runs but produces incorrect output

Problems may be repeatable or sporadic

Debugging strategies

Debugging closely tied to testing

- Feed inputs to program – see if program gives correct expected outputs
- Have some structure that makes it easy to test repeatedly
  - And easy to compare actual outputs to expected outputs
- Shell script, more advanced tools…

There is lots of professional automated test management and execution SW

Debugging strategies

`printf()` debugging

- Everybody uses it
- Not so fast or efficient

An improvement: "verbose mode"

- **-v** option on command line or control terminal during run-time
- `if (verbose) { printf("Debug info\n"); }`

Write pgm with verbose mode

Log files

What if your program runs for hours or days?

Want a permanent file with debug/status information

Can set "severity levels" for each printout, for filtering

- INFO, WARNING, ERROR, CRITICAL

Review logger.cpp

Debugging with a debugger

breakpoints

memory inspection

See "fastalloc.c" IN ECLIPSE!

vlab debug tools

Command line debugger - gdb

gdb executableFilename

Useful gdb commands:

- run
- break <linenumber>
- continue
- next
- step
- list
- print
- quit

Compile with "-g" option: saves variable names as human-readable text
Use GDB to walk thru 'fastalloc'
Why use gdb? Like command line, some profs insist, good support for assembly language (CS315)

Memory checker - valgrind

valgrind is an easy-to-use memory checker

- Does your program free all memory it mallocs?
- Does your program use not-allocated memory?

Usage:

- Compile your program with **gcc –g –O0**
- Run **valgrind --tool=memcheck --demangle=yes --leak-check=full –s ProgName**

Put command-line arg into Makefile! Show that

run buggy.c from stargate

valgrind code-along

Please find your **LinkedList** code from Day15

- Copy it to stargate using **scp**

On a vlab machine

- Rebuild it with **-g -O0**
- Run valgrind on it
- You should find memory leaks—let's work to fix them!

What's a "memory leak"?

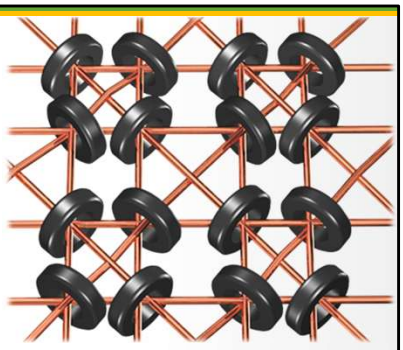
Try running your FIFO in gdb also! Day11

Corefiles

Corefiles

A "corefile" stores an image of a program's memory at the time of a crash, along with some detail from CPU about cause of the crash

Debuggers such as Eclipse and GDB let programmer browse the corefile, to understand why the program crashed.



Why "core"? Memory used to be build of iron cores
Windows: no application corefiles, only OS dumps (!)
Linux: some pain to enable. Show on Linux computer. Mac??
Eclipse: Search for "corefile" and select "Debug corefile". Best to compile with –g
Run 'makeCore' and look at corefile. Look at ADDRESSES of index and buf – adjacent!
(ANY STUDENT EXAMPLES FROM EARLIER LABS?)

corefiles

Modern operating systems tend to store corefiles in difficult-to-find places

- On vlab machines, it is **/var/lib/systemd/coredump/**
- Don't forget: **ulimit –c 8192**

Corefiles are compressed with a tool called **zstd**

- Uncompress with **zstd –d CompressedCorefilename**

Let's do a codealong on vlabs: make a corefile
CAN YOUR FIND YOURS? Or lots of other people's?

gdb and corefiles

gdb has a great feature: debugging of corefiles

- **gdb MyProgram.exe CoreFilename**

What does this do?

Tells you WHERE the program crashed, WHY, stack trace, and values of all variables.
Try with buggy2.c