Week 8

Announcements

- HW5 by 11:59 PM on Mar 2
- HW6, ADV6 due by 11:59 PM on Mar 11
- HW7, ADV7 due by 11:59 PM on Mar 18

Foreword

Today's lecture is going to be less formal/structured and may be pretty short

Lecture 8: Debugging

```
segmentation fault (core dumped)
```

Overview

- `printf` debugging
- Logging
- GDB
- Checking memory with Valgrind

Q: What is your debugging method?

`printf` debugging

- Intuitive: just print stuff out at certain points

`printf` debugging

- Intuitive: just print stuff out at certain points
  - What if you're done littering your file with debug prints?

`printf` debugging

- Intuitive: just print stuff out at certain points
  - What if you're done littering your file with debug prints?
- With the power of preprocessors, we can turn it on and off!

`printf` debugging

- Intuitive: just print stuff out at certain points
  - What if you're done littering your file with debug prints?
- With the power of preprocessors, we can turn it on and off!

```
#ifdef DEBUG
printf("This be a debug message\n");
#endif // DEBUG
```

`printf` debugging

- Intuitive: just print stuff out at certain points
  - What if you're done littering your file with debug prints?
- With the power of preprocessors, we can turn it on and off!

```
#ifdef DEBUG
printf("This be a debug message\n");
#endif // DEBUG
```

```
#ifdef DEBUG
  // this is known as a "variadic macro"
  #define dbgprintf(fmt, ...) printf(fmt, ##__VA_ARGS__)
#else
  #define dbgprintf(fmt, ...)
#endif
// ...
dbgprintf("This be a debug message\n");
```

Logging

- An extension on printing
- Provide different verbosity/logging levels
- Set your verbosity level to increase/decrease the amount of logging
  - More logging uses more resources
- Log to `stdout` or to some file

Levels

- Fatal
  - "We are going to die now"

Levels

- Fatal
  - "We are going to die now"
- Error
  - "That really hurt, but I can keep going"

Levels

- Fatal
  - "We are going to die now"
- Error
  - "That really hurt, but I can keep going"
- Warning
  - "Something *might* be feeling weird"

Levels

- Fatal
  - "We are going to die now"
- Error
  - "That really hurt, but I can keep going"
- Warning
  - "Something *might* be feeling weird"
- Info
  - "Hey a cool thing happened"

Levels

- Fatal
  - "We are going to die now"
- Error
  - "That really hurt, but I can keep going"
- Warning
  - "Something *might* be feeling weird"
- Info
  - "Hey a cool thing happened"
- Debug
  - "A thing happened, here's some details"

Levels

- Fatal
  - "We are going to die now"
- Error
  - "That really hurt, but I can keep going"
- Warning
  - "Something *might* be feeling weird"
- Info
  - "Hey a cool thing happened"
- Debug
  - "A thing happened, here's some details"
- Trace/Verbose
  - "IStartedupThenIGotAByteToEatThenISentItOverHere"

GDB (GNU Debugger)

- Debugging tool that lets you look around during execution
- Once again, this tool is pretty deep: look at the GDB manual for details
- We'll go over some big overarching concepts and features
  - Interface
  - Breakpoints and watchpoints
  - Stack frames

Interface

- Invoking: `$ gdb [options] [executable file] [core file]`
  - `$ gdb ./myapp`, `$ gdb myapp`
- Hitting return/enter without anything will repeat the previous command
- Commands
  - `run [arguments] [file redirects]`
  - `next [count]`: step *over* functions, "next line"
  - `step [count]`: step)into_ functions
  - `finish`: step *out* of current function
  - `print <expression>`: print expression (e.g. variables)
  - `list [location]`: list source code

Interface

- Invoking: `$ gdb [options] [executable file] [core file]`
  - `$ gdb ./myapp`, `$ gdb myapp`
- Hitting return/enter without anything will repeat the previous command
- Commands
  - `run [arguments] [file redirects]`
  - `next [count]`: step *over* functions, "next line"
  - `step [count]`: step)into_ functions
  - `finish`: step *out* of current function
  - `print <expression>`: print expression (e.g. variables)
  - `list [location]`: list source code
  - `break <location>`: set breakpoint
  - `watch <expression>`: set watchpoint
  - `info breakpoints`, `info watchpoints`: list break/watchpoints
  - `where`: list stack frames
  - `frame <stack frame>`: change stack frame
- Entering incomplete commands (such as a single) letter will run a command if there is no ambiguity:
  - `r` -> `run`
  - `n` -> `next`
  - `b` -> `break`

Breakpoints and watchpoints

Breakpoint: stop at a certain location in the program

- Can be conditional!
- `info breakpoints`, `info break`, `info b` will list breakpoints
- Examples:
  - `break 20`
  - `break main.cpp:21`
  - `break main.cpp:21 if argc == 4`
  - `break coolfunction`

Breakpoints and watchpoints

Breakpoint: stop at a certain location in the program

- Can be conditional!
- `info breakpoints`, `info break`, `info b` will list breakpoints
- Examples:
  - `break 20`
  - `break main.cpp:21`
  - `break main.cpp:21 if argc == 4`
  - `break coolfunction`

Watchpoint: stop when an expression changes

- `info watchpoints`, `info watch` will list watchpoints
- Examples:
  - `watch somevar`
  - `watch a + b`

Breakpoints and watchpoints

Breakpoint: stop at a certain location in the program

- Can be conditional!
- `info breakpoints`, `info break`, `info b` will list breakpoints
- Examples:
  - `break 20`
  - `break main.cpp:21`
  - `break main.cpp:21 if argc == 4`
  - `break coolfunction`

Watchpoint: stop when an expression changes

- `info watchpoints`, `info watch` will list watchpoints
- Examples:
  - `watch somevar`
  - `watch a + b`
- `disable <number>`: disable a break/watchpoint
- `delete <number>`: delete a break/watchpoint

(Catchpoint: stop when an event such as a C++ exception occurs)

Stack frames

- A *stack frame* holds all information local to a particular function call
  - Local variables
  - Arguments
  - (Return address)
- Function calls will push a frames on the *stack*
- Function returns will pop the frame off the *stack*

Stack frames

- A *stack frame* holds all information local to a particular function call
  - Local variables
  - Arguments
  - (Return address)
- Function calls will push a frames on the *stack*
- Function returns will pop the frame off the *stack*
- GDB can show us this info:
  - `where` can show us the current stack frames
  - `frame <number>` can have us switch to a stack frame so we can look at its variables

Valgrind

- General dynamic analysis tool
  - Valgrind manual
- Most known for its Memcheck tool for checking memory accesses (which we'll be focusing on)
  - Memcheck manual entry

Valgrind

- General dynamic analysis tool
  - Valgrind manual
- Most known for its Memcheck tool for checking memory accesses (which we'll be focusing on)
  - Memcheck manual entry
- Super useful at finding things like:
  - Memory leaks
  - Use-after-frees
  - Invalid reads
  - Use of uninitialized variables

Valgrind

- General dynamic analysis tool
  - Valgrind manual
- Most known for its Memcheck tool for checking memory accesses (which we'll be focusing on)
  - Memcheck manual entry
- Super useful at finding things like:
  - Memory leaks
  - Use-after-frees
  - Invalid reads
  - Use of uninitialized variables
- Easy to invoke:
  - `$ valgrind ./myapplication`
  - `$ valgrind --leak-check=full ./myapplication`

Closing thoughts

- Ultimately use the right tool for the job
- GDB doesn't work particularly well in complex systems
  - Logging can help out here, but it does incur some overhead
- Valgrind can seriously slow down your program

Demos

Questions