Course Wrap-up

Computer Systems Programming, Spring 2023

Instructor: Travis McGaha

TAs:

Kevin Bernat
Mati Davis
Chandravaran Kunjeti
Shufan Liu
Jialin Cai
Donglun He
Heyi Liu
Eddy Yang

Poll Everywhere

pollev.com/tqm

- How “close” are you to finishing the semester?

Logistics

- HW4 Posted — Due Thursday 4/20 @ 11:59
  Extended to 4/26 @11:59 pm
- Project Released! — Due Wednesday 4/26 @ 11:59
- HW2 grades & Midterm grades posted
  - Can fix HW2 submissions
  - Midterm has regrades & the clobber policy
- Final Exam Review in lecture on Wednesday

Logistics

- Late Policy:
  - You can still use the same late policy for HW5 and the final project
  - I can grant extensions into reading days
  - I REALLY don't want to grant extensions into finals week
  - Email me (Travis) at least a day in advance of the deadline so that I have time to process the extension

- Final Exam: May 2nd @noon to May 6th @noon
  - Cumulative & Midterm Clobber policy

- Office hours next week posted on the course website
  - Travis may have SOME OH over finals, tbd

Lecture Outline

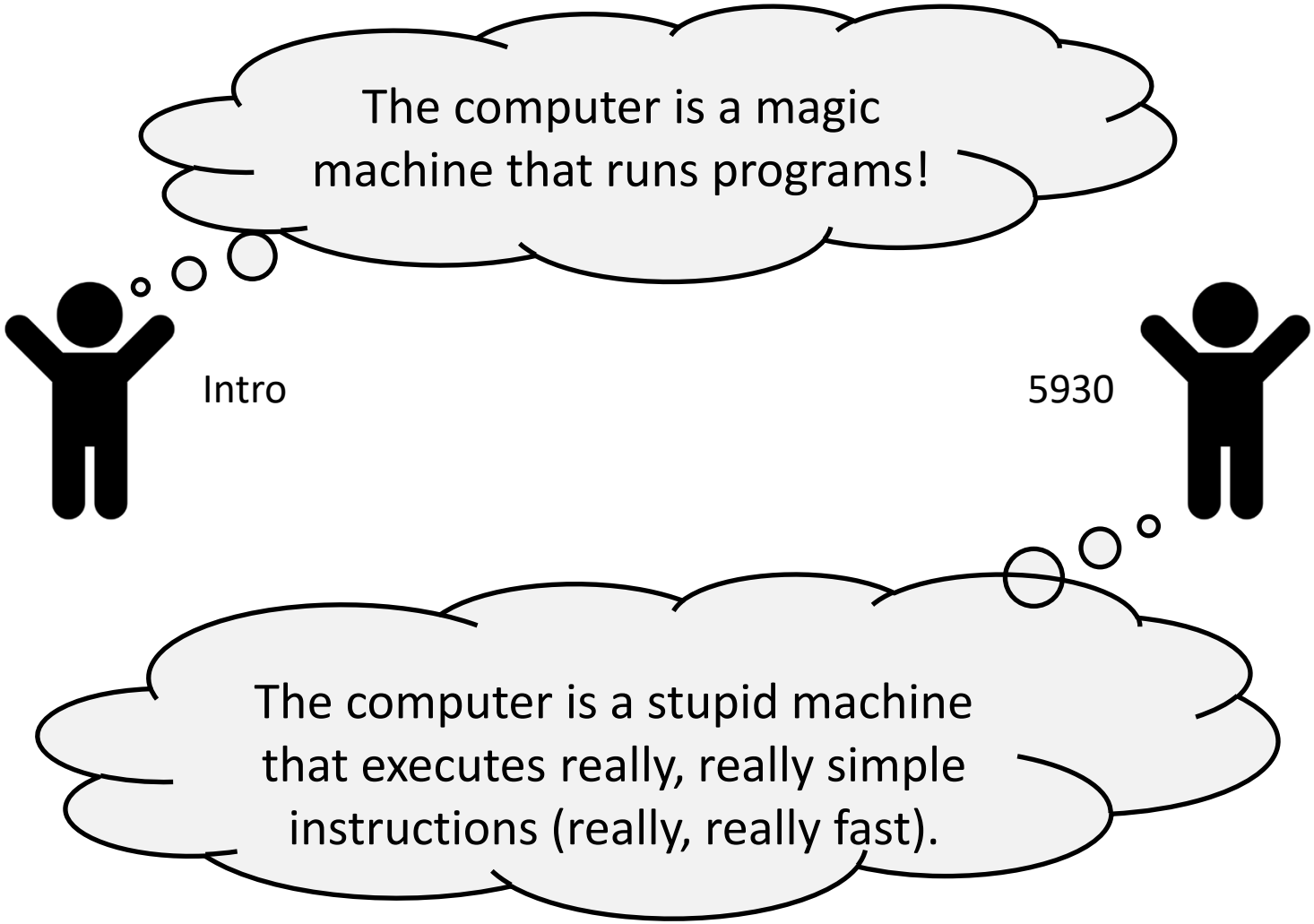
- **Course Wrap-up**

What have we been up to for the last 14 weeks?

- Ideally, you would have "learned" everything in this course, but we'll use red stars today to highlight the ideas that we hope stick with you beyond this course

Course Goals

- Explore the gap between:



Systems Programming: The Why

- The programming skills, engineering discipline, and knowledge you need to build a system
  1) Understanding the "layer below" makes you a better programmer at the layer above
  2) Gain experience with working with and designing more complex "systems"
  3) Learning how to handle the unique challenges of low-level programming allows you to work directly with the countless "systems" that take advantage of it

So What is a System?

- "A **system** is a group of interacting or interrelated entities that form a unified whole. A system is delineated by its spatial and temporal boundaries, surrounded and influenced by its environment, described by its structure and purpose and expressed in its functioning."
  - https://en.wikipedia.org/wiki/System
  - Still vague, maybe still confusing

- But hopefully you have a better idea of what a system in CS is now
  - What kinds of systems have we seen…?

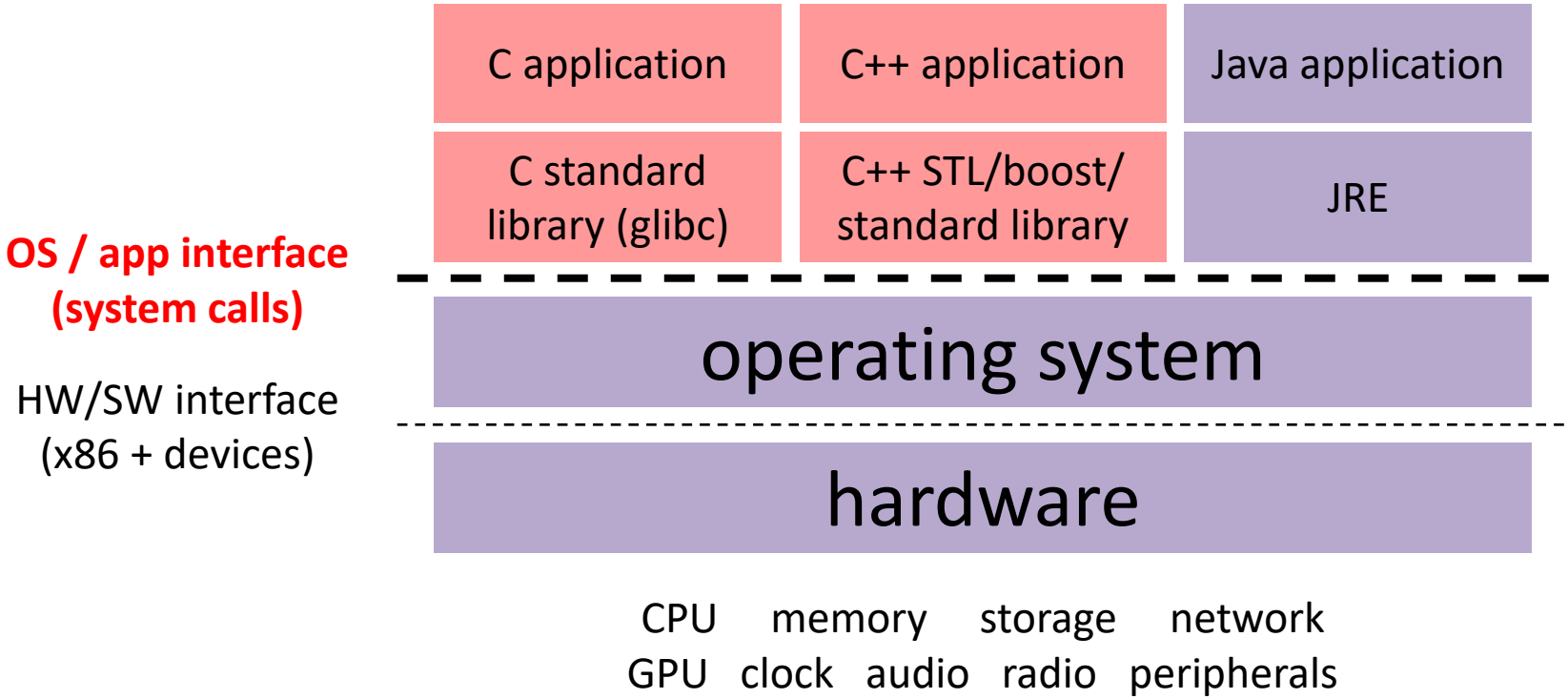
Software System

- Writing complex software systems is *difficult*!
  - Modularization and encapsulation of code
  - Resource management
  - Documentation and specification are critical
  - Robustness and error handling
  - Must be user-friendly and maintained (not write-once, read-never)

- **Discipline:** cultivate good habits, encourage clean code
  - Coding style conventions
  - Unit testing, code coverage testing, regression testing
  - Documentation (code comments, design docs)

The Computer as a System

- Modern computer systems are increasingly complex!
  - Networking, threads, processes, pipes, files
  - Buffered vs. unbuffered I/O, blocking calls, latency



A Network as a System

- A networked system relies heavily on its connectivity
  - Depends on materials, physical distance, network topology, protocols
- Conceptual abstraction layers
  - Physical, data link, network, transport, session, presentation, application
  - Layered *protocol* model
    - We focused on IP (network), TCP (transport), and HTTP (application)
- Network addressing
  - MAC addresses, IP addresses (IPv4/IPv6), DNS (name servers)
- Routing
  - Layered packet payloads, security, and reliability

Systems Programming: The What

- The programming skills, engineering discipline, and knowledge you need to build a system
  - **Programming:** C & C++
  - **Discipline:** design, testing, debugging, performance analysis
  - **Knowledge:** long list of interesting topics
    - Concurrency, OS interfaces and semantics, techniques for consistent data management, distributed systems algorithms, …
    - Most important: a deep understanding of the "layer below"

Main Topics

- C
  - Low-level programming language
- C++
  - The 800-lb gorilla of programming languages
  - "better C" + classes + STL + smart pointers + …
- Memory management
- System interfaces and services
- Networking basics – TCP/IP, sockets, …
- Concurrency basics – POSIX threads, synchronization
- Multi-processing Basics – Fork, Pipe, Exec

Topic Theme: Abstraction

- C: `void*` as a generic data type
- C++: hide execution complexity in simple-looking code
  - *e.g.*, operator overloading, dispatch, containers & algorithms
- C++: templates to generalize code
- OS: abstract away details of interacting with system resources via system call interface
- Networking: 7-layer OSI model hides details of lower layers
  - *e.g.*, DNS abtracts away IP addresses, IP addresses abstract away MAC addresses

Topic Theme: Using Memory

- Variables, scope, and lifetime
  - *Static*, *automatic*, and *dynamic* allocation / lifetime
  - C++ objects and destructors; C++ containers and copying
- Pointers and associated operators (`&`, `*`, `->`, `[]`)
  - Can be used to link data or fake "call-by-reference"
- Dynamic memory allocation
  - **`malloc`**/**`free`** (C), **`new`**/**`delete`** (C++), smart pointers (C++)
  - Who is responsible? Who owns the data? What happens when (not if) you mess this up? (dangling pointers, memory leaks, …)
- Tools
  - Debuggers (`gdb`), monitors (`valgrind`)
  - Most important tool: thinking!

Topic Theme: Data Passing

- C: output parameters
- C++: Copy constructors, and copy vs move semantics
- Threads: return values or shared memory/resources
  - Leads to synchronization concerns
- I/O to send and receive data from outside of your program (*e.g.*, disk/files, network, streams)
  - Linux/POSIX treats all I/O similarly
  - Takes a LONG time relative to other operations
  - Blocking vs. polling
- Buffers can be used to temporarily hold passed data
  - Buffering can be used to reduce costly I/O accesses, depending on access pattern

Topic Theme: Optimize for your User

- Readability:
  - Properly **modularize** your code using functions, classes, namespaces, and header files
    - Takes advantage of the preprocessor and linker
  - **Documentation** should be thorough, up-to-date, and easy to find (*e.g.*, public interface)
  - Error reporting behaviors should be documented properly
- Usability:
  - Use proper linkage and encapsulation to avoid namespace collisions
  - Make building easy and efficient via build tools (*e.g.*, Makefile)
  - Your programs should be **robust** – no unexpected or unexplained crashes

Congratulations!

- Look how much we learned!
- Lots of effort and work, but lots of useful takeaways:
  - Debugging practice
  - Reading documentation
  - Tools (gdb, `valgrind`, `helgrind`)
  - C and C++ familiarity, including multithreaded and networked code
- Go forth and build cool systems!

Future Courses

- Systems Courses
  - CIS 5050: Software Systems
  - CIS 5480: Operating Systems Design and Implementation
  - CIS 5530: Networked Systems
  - CIS 5550 Internet and Web Systems
  - CIS 5500: Database and Information Systems
  - CIS 5410 Embedded Software for Life-Critical Applications
- Otherwise related courses
  - CIS 5600 Interactive Computer Graphics
  - CIS 5610 Advanced Computer Graphics
  - CIS 5650 GPU Programming and Architecture
  - CIS 5570 Programming for the Web

Thanks for a great semester!

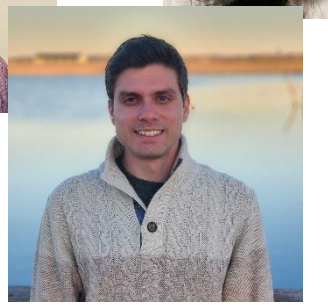
- Special thanks to all the instructors before me (Both at UPenn and UW) who have influenced me to make the course what it is

- Huge thanks to the course TA's for helping with the course!





No image



Thanks for a great semester!

- Thanks to you!
  - It has been another tough semester. Still not completely out of the pandemic, Zoom fatigue, faltering motivation, etc
  - Relatively "new" version of the course. Many of the assignments and infrastructure are recently developed.
  - You've made it through so far, be proud that you've made it and what you've accomplished!

- **Please take care of yourselves, your friends, and your community**

Ask Me Anything