Poll Everywhere

pollev.com/cse333j

What was your favorite 333 topic group?

A. Memory Management: pointers, references, malloc/free, new/delete, memory bugs, smart pointers

B. Data Structures: arrays, structs, containers

C. Object-Oriented Programming: classes, inheritance

D. Modularization: compilation, interfaces, templates

E. I/O: files, buffering, network programming

F. Concurrency

G. I prefer not to say

Systems Programming
Course Wrap-Up

Instructors:

Justin Hsia Amber Hu

Teaching Assistants:

Ally Tribble	Blake Diaz	Connor Olson
Grace Zhou	Jackson Kent	Janani Raghavan
Jen Xu	Jessie Sun	Jonathan Nister
Mendel Carroll	Rose Maresh	Violet Monserate

Relevant Course Information

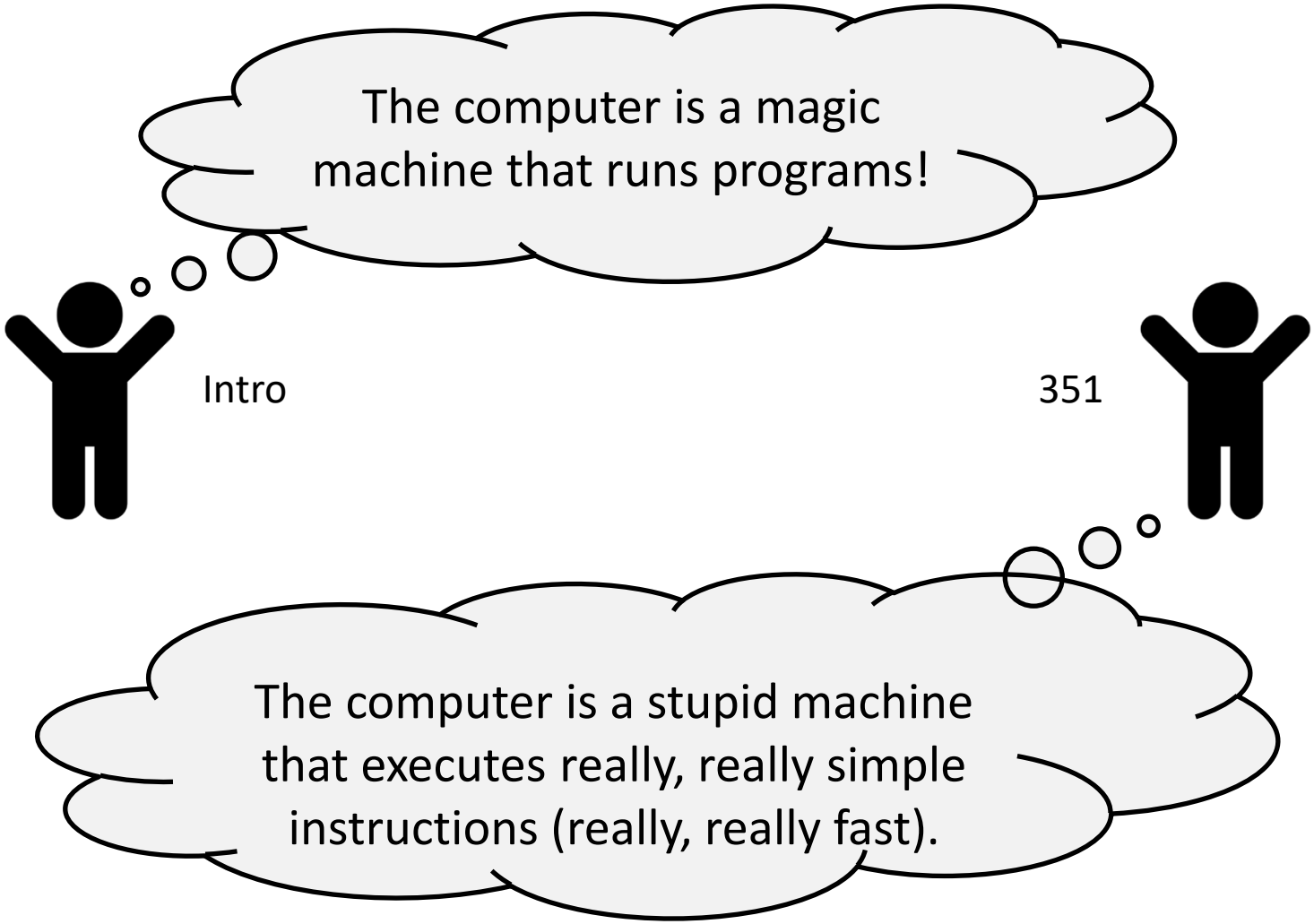
- Homework 4 due yesterday (3/12)
  - Submissions accepted until Sunday (3/15)
- Course evaluations (Ed #661) due Sunday night
- Final Exam is Wednesday, 3/18 @ 12:30–2:20 PM
  - Kane Hall 110 and 120
  - See Ed post #630
  - Final review session tonight @ 4:30 PM, CSE2 G10 + Zoom
- Check grades as Canvas assignments are released

What have we been up to for the last 10 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…?

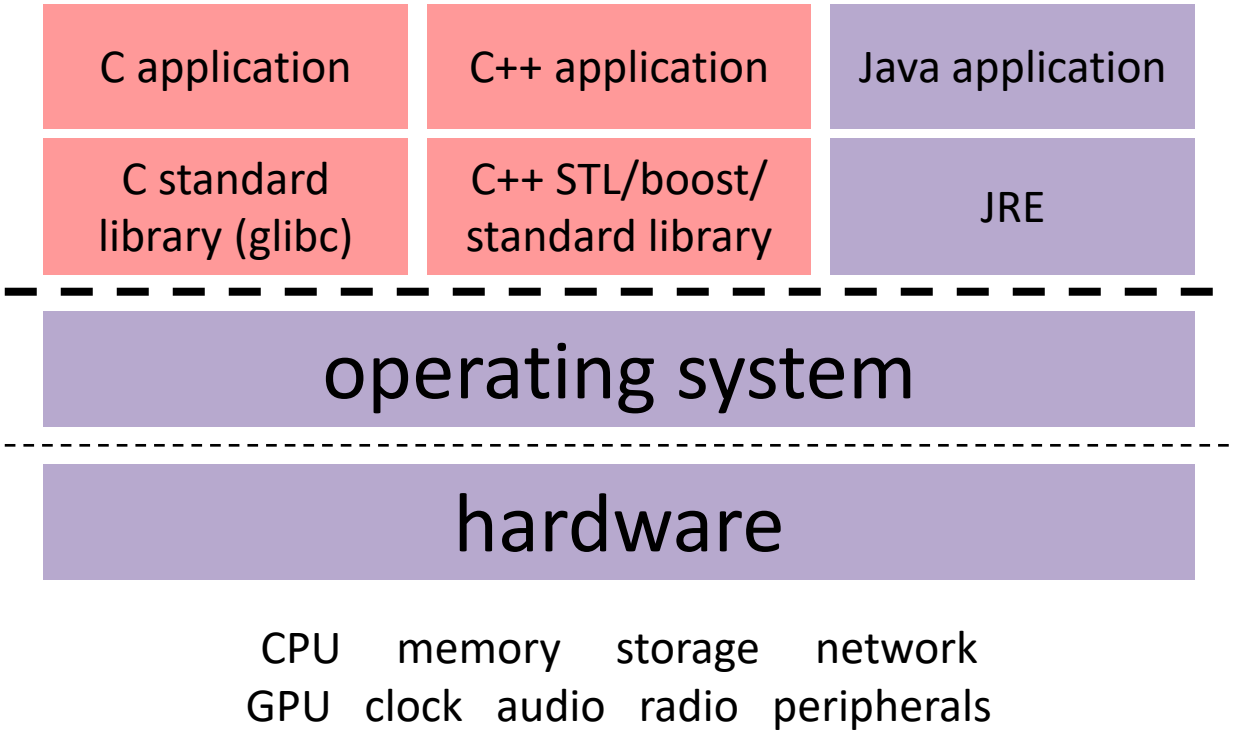
Software System

- 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, concurrency/parallelism, distributed systems
  - Buffered vs. unbuffered I/O, blocking calls vs. polling, 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

Topic Theme: Abstraction

- C: `void*` as a generic data type
- C: abstracted data types to hide system-specific details
  - *e.g.*, `size_t`, `int32_t`, `sa_family_t`, `pthread_mutex_t`
- 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 tools: thinking!

Topic Theme: Data Passing

- C: output parameters
- Processes: status codes (*e.g.*, `EXIT_SUCCESS`)
- 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 and metacognition (gdb)
  - Reading documentation (man pages, cppreference)
  - Tools (`git`, `valgrind`, makefiles)
  - C and C++ familiarity, including multithreaded and networked code

- Go forth and build cool systems!
  - But carefully consider who can/should use it as well as what values are embedded in it

Future Courses (1/2)

- CSE 451: Introduction to Operating Systems
  - How to manage all of the computer's resources
- CSE 452: Introduction to Distributed Systems
  - How to get large collections of computers to collaborate (correctly!)
- CSE 453: Data Center Systems
  - The technologies that underly next generation data centers used by large scale applications
- CSE 461: Introduction to Networks
  - How to design a network to transmit data
- CSE 401: Introduction to Compiler Construction
  - How a compiler works (theory + programming + systems!)

Future Courses (2/2)

- CSE 334: Concurrency, Parallelism, and Rust
  - Techniques to write better, faster code for modern, scalable systems infrastructure
- EE/CSE 474: Intro to Embedded Systems
  - How to interact with computers with limited resources (*e.g.*, RAM) and "real time" requirements
- CSE 444: Database Systems Internals
  - How to build a database management system

Thanks for a great quarter!

- Special thanks to the course content creators!!!


Steve Gribble


Hal Perkins


John Zahorjan


Hannah Tang


Travis McGaha

- Huge thanks to your awesome TAs!

Ask Us Anything


That's all Folks!