Poll Everywhere

pollev.com/cse333

About how long did Exercise 12 take you?

A. [0, 2) hours
B. [2, 4) hours
C. [4, 6) hours
D. [6, 8) hours
E. 8+ Hours
F. I didn’t submit / I prefer not to say

Course Wrap-Up

CSE 333 Winter 2023

Instructor: Justin Hsia

Teaching Assistants:

Adina Tung
Danny Agustinus
Edward Zhang
James Froelich
Lahari Nidadavolu
Mitchell Levy
Noa Ferman
Patrick Ho
Paul Han
Saket Gollapudi
Sara Deutscher
Tim Mandzyuk
Timmy Yang
Wei Wu
Yiqing Wang
Zhuochun Liu

Relevant Course Information

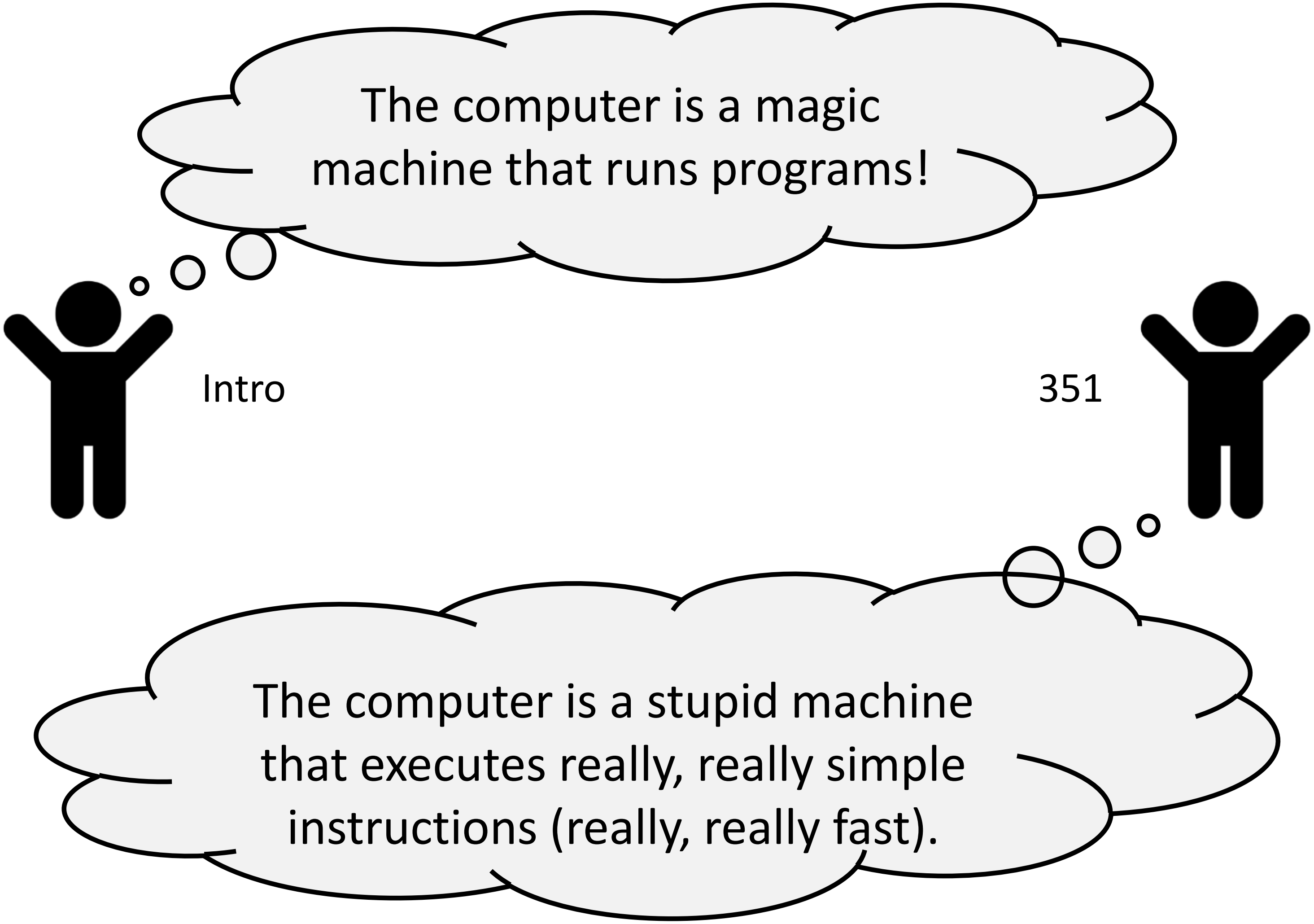
- Homework 4 due tomorrow (3/9)
  - Submissions accepted until Sunday (3/12)
- Course evaluations (Ed #1126) due Sunday night
- Final starts Monday (3/13) and runs until the end of Wednesday (3/15)
- Check grades as Canvas assignments are released

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:


The computer is a magic machine that runs programs!
Intro
351
The computer is a stupid machine that executes really, really simple instructions (really, really fast).

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…?

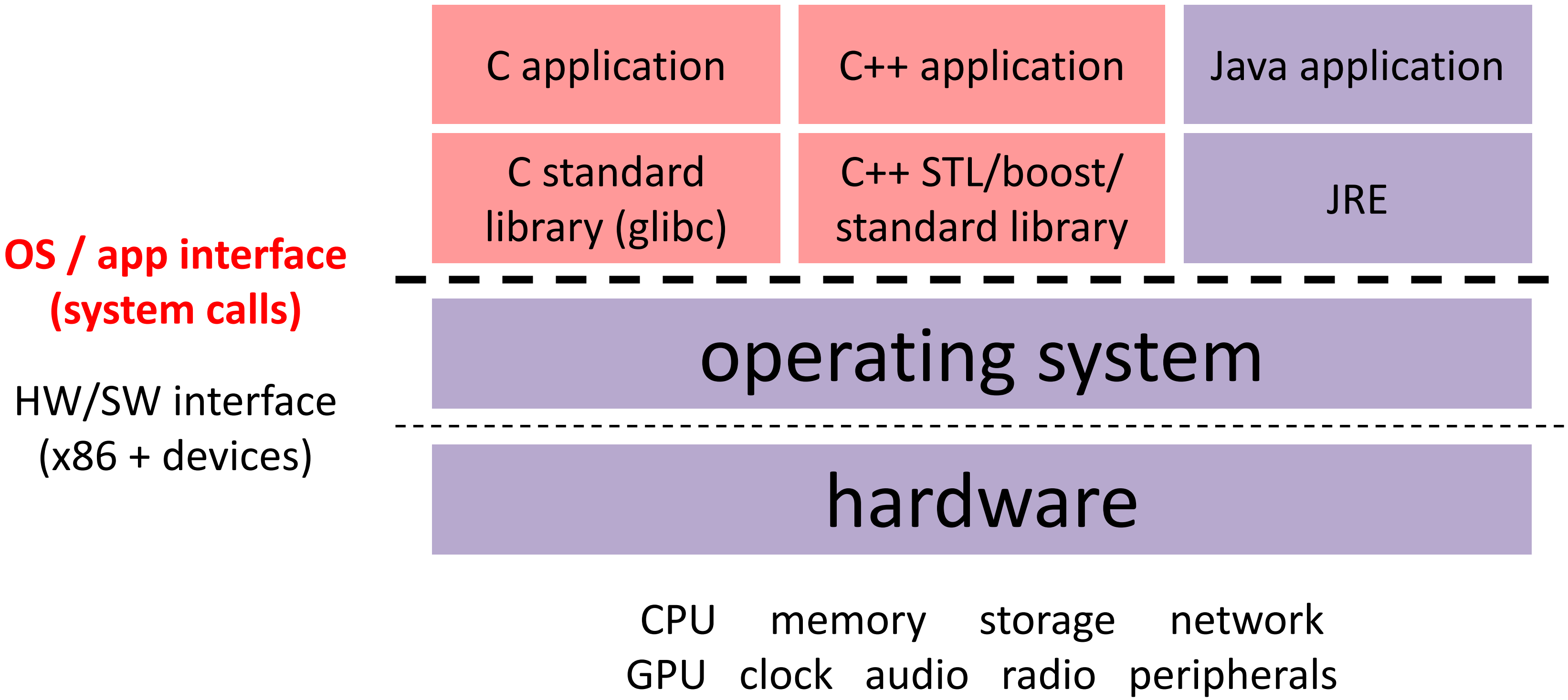
Software System

- 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



C application

C++ application

Java application

C standard library (glibc)

C++ STL/boost/ standard library

JRE

OS / app interface (system calls)

operating system

HW/SW interface (x86 + devices)

hardware

CPU memory storage network

GPU clock audio radio peripherals

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 tool: 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`, bug journals)
  - Reading documentation
  - 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 – Systems Courses

- CSE 451: Introduction to Operating Systems
  - How do you manage all of the computer's resources?
- CSE 452: Introduction to Distributed Systems
  - How do you get large collections of computers to collaborate (correctly!)?
- CSE 461: Introduction to Computer Communication Networks
  - How to design a network to transmit data?
- CSE 401: Introduction to Compiler Construction
  - How does a compiler work? (theory + programming + systems!)
- CSE 444: Database Systems Internals
  - How to build a database management system?

Future Courses – Courses in C/C++

- EE/CSE 474: Intro to Embedded Systems
  - How to interact with computers with limited resources (*e.g.*, RAM) and "real time" requirements
- CSE 455: Computer Vision
  - Theory-heavy course on the representation and analysis of images (*e.g.*, colors, object recognition)
  - C first half, Python in second half
- CSE 457: Computer Graphics
  - Theory- and coding-heavy course on creating digital art
  - Graphics almost always use C++ or C#

Future Courses – Otherwise Related

- CSE 331: Software Design and Implementation
  - Dedicated to good software practices, design, modularity, and more – “core” knowledge for being a good software dev

- CSE 332: Data Structure and Parallelism
  - Use parallelism (a form of concurrency) in Java at the end

- Various Web Programming Courses:
  - CSE 154: Web Programming
  - INFO 340: Client-Side Development
  - INFO 441: Server-Side Development
    - Website design and building

Thanks for a great quarter!

- Special thanks to the course content creators!!!

Steve Gribble

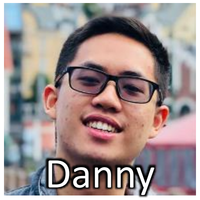
Hal Perkins

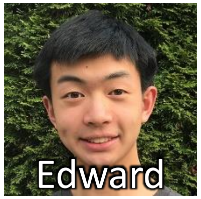
John Zahorjan

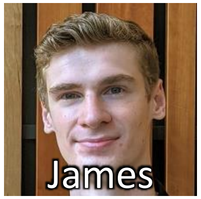
Hannah Tang

Travis McGaha

- Huge thanks to your awesome TAs!







Ask Me Anything

That's all Folks!