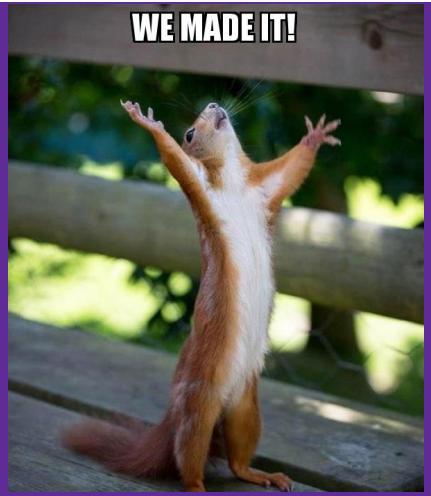
CSE 333 Section 10

You have accomplished a lot, let's take a look



UNIVERSITY of WASHINGTON

Logistics

- **Daylight Savings *starts* this weekend**
  - We lose an hour 😭
- **HW4**
  - Due **Thursday** @ 11:59 pm (tonight!!)
  - You (as in *everyone*) can use 2 late days even if you used a bunch earlier!
- **Final Exam**
  - Open **Monday – Wednesday** (3/13-15) on Gradescope

Exercise Summary

Exercise Summary (Part I: C)

Ex 1 – Introduction to C

- Wrote a C program from scratch.
- Learned how to parse command line arguments.

Ex 2 - Pointers and Arrays

- Wrote code that uses pointers and data representations.
- Examined the relationship between pointers and arrays.

Ex 3 - Custom structs and typedefs, dynamic memory

- Defined custom structs with typedef.
- Use malloc and free to manage dynamically-allocated data without memory leaks.
- Wrote a multi-file program.

Ex 4 - C stdio and POSIX I/O

- Used various POSIX I/O library functions with files and directories.
- Implemented your own buffered system.
- Implemented code that error checks I/O function calls and properly cleans up resources on every execution path.

Exercise Summary (Part 2: C++)

Ex 5 – Introduction to C++

- Used various “new” things C++ has compared to C
- Stream objects & operators, C++ headers, etc.

Ex 6 - C++ classes

- Modularizing classes in C++.
- Class constructors, data members, access specifiers.
- Operator overloading.

Ex 7 - More complex C++ classes

- Non-member vs non-member friend functions.
- Destructors, use of dynamic memory allocation in classes.

Ex 8 - Templates and Containers

- Used templates to build C++ functions/classes.
- Used containers/algorithms provided by C++’s vast STL library (*e.g.*, `map`).

Exercise Summary (Part 3: Systems)

Ex 9 – C++ Inheritance

- Learned about abstract classes and how to implement abstract classes.
- Automatic memory management via smart pointers.

Ex 10 - Client Side Networking

- Build a client-side networking app using POSIX API (`getaddrinfo`, `socket`, `connect`, `read`/`write`, `close`).
- Use DNS lookup, IP addresses, TCP stream connections.

Ex 11 - Server Side Networking

- Build a server-side networking application using POSIX API (`getaddrinfo`, `socket`, `bind`, `listen`, `accept`, `read`/`write`, `close`).
- Use DNS lookup on own dynamic IP address, binding to port numbers + IP addresses, client accept loop.

Ex 12 - Concurrency

- Convert sequential code to concurrent/multithreaded via the POSIX pthreads API.
- Use synchronization primitives to avoid concurrency issues – pthread mutexes/locks.

Homework Summary

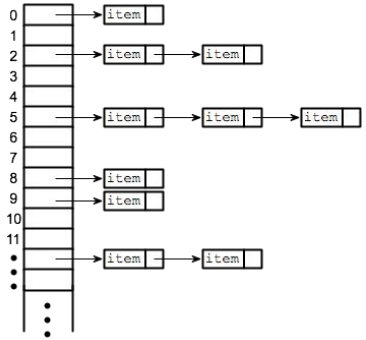
Homework Summary

Homework 1: Implemented modules for a doubly-linked list and a chained hash table

Homework 2: Created a local search engine (like `grep`)

Homework 3: Modified Homework 2 to save and read index files from disk

Homework 4: Creating an internet-accessible version of your search engine!




```
[attu1 hw2]$ ./searchshell ./test_tree
Indexing './test_tree'
enter query:
ulysses
  ./test_tree/books/ulysses.txt (10)
  ./test_tree/books/countofmontecristo.txt (2)
  ./test_tree/books/leavesofgrass.txt (1)
  ./test_tree/books/lesmiserables.txt (1)
  ./test_tree/books/paradiselost.txt (1)
enter query:
```

Reflection Activity

Take about 5 minutes with yourself and your neighbors to reflect on your experiences with homework, exercises, and lecture this quarter.

Guiding Questions:

- What were a few things that you took away from this course?
- What are a few things that you are still confused about?
- What were the most interesting topics to you this quarter?
- What advice would you give to a future CSE 333 student?

Outcomes of Doing the Homework

- Building a sample system based on the design decisions the staff has made for the homeworks
  - You can consider why these decisions were made

- Getting accustomed to larger-scale programming projects
  - Learning about the code surrounding what you are to implement

- Practicing programming idioms – error checking, style guides, etc.
- Reading documentation!

Adding it to your Resume

- Why consider adding it in?
  - You had to work with this code base for an entire quarter
  - It demonstrates your ability to work with a large code base and system
  - It's pretty cool to add "mini-Google"

- What might be on there?
  - **System Design** – Creating data structures to store information. Saving it into a file. Following a protocol to create a web application. Implementing and using module interfaces. Handling unexpected/bad inputs.
  - Programming in C and C++ (handling C/C++ idioms)
  - Add it to your projects section. You just built a mini search engine all the way from the low-level data structures up to the HTML. You even added some security features.

What's Next?

Main Course Themes

- Debugging: gdb, valgrind, and… rubber duckies
  - Solving Seg. Faults, Memory Leaks, incorrect output
  - C/C++ programming hopefully made you more cognizant of where bugs may occur at a low level
- Program Layout: Modularity, Helper functions, Inheritance
  - Improve readability and ease of maintenance
- Documentation: Reading API’s and incorporating into programs
  - C++ Reference Guide, man pages
- Google Style Guide: linter, Lots of reading…
  - cpplint.py is an automated tool to help with catching style issues
  - A lot of it was learning to adapt to a system for writing and reading code in the same format

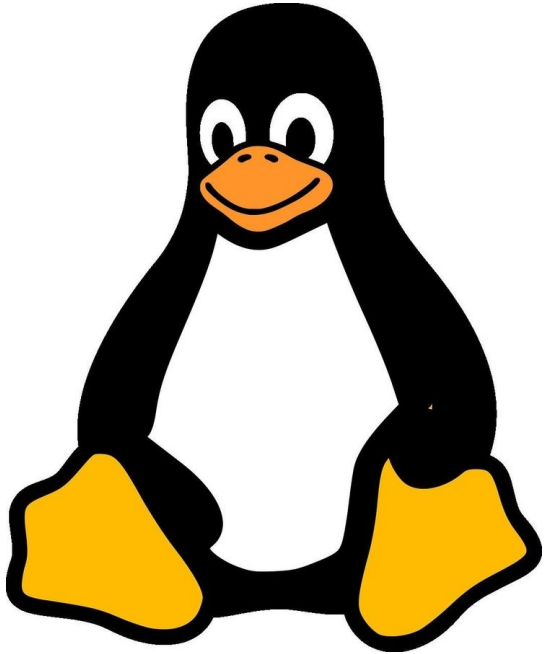
CSE 451 – Operating Systems

- CSE333 is called “Systems Programming”
  - We hope you learned to interact with the OS
  - “Magic OS stuff” kept mostly hidden in CSE 333

- If you want to understand the “magic OS” stuff, take this!
- You build your own operating system and learn how to,
  - Build a file system
  - Implement fork, wait, exit, and exec
  - Implement crash consistency

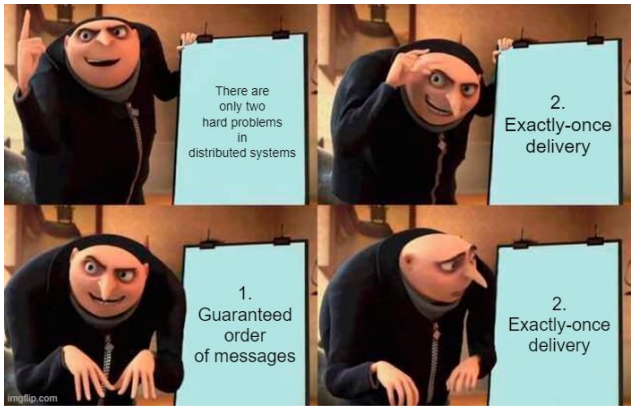
- In C (WOO HOO)
- Partner class!

- Lots of work, but very rewarding!



CSE 452 – Distributed Systems

- Understanding how to design massive, fault tolerant systems over an unreliable network.
    - Think Facebook, AWS, Google, etc.



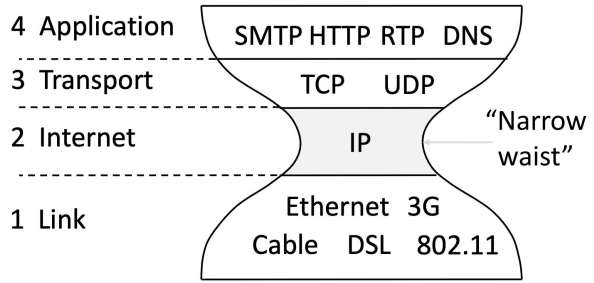
- Doesn't require OS, topics rarely if at all come up
    - But helps with writing the design docs
- More theory to it than you'd think

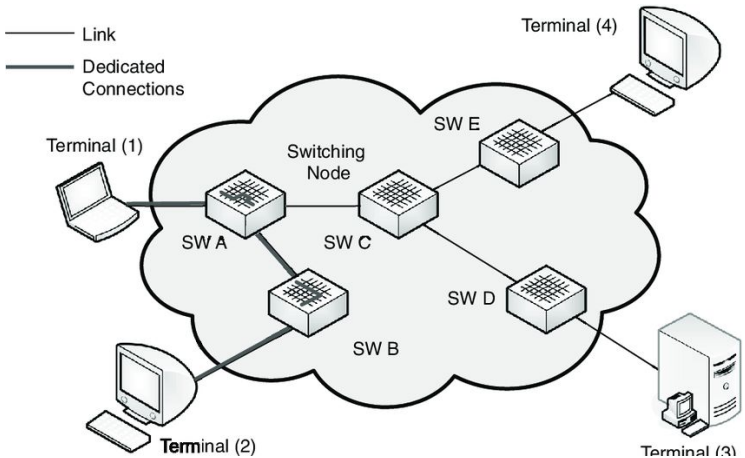
- Lots of work (project is big and confusing)
- Most thought per line of code of any class in the Allen School, also the most lines of code.

- Hard, but very very cool. Would strongly recommend. Huge resume booster.

CSE 461 - Networks

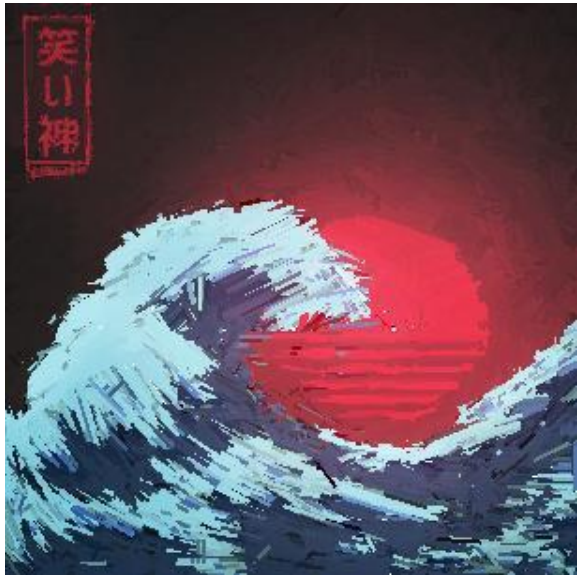
- We did a “basic” understanding of how networks work
- We mainly just understood how it works in C and with POSIX
- Mostly Python (POX and Mininet)
    - First lab is “language of choice”, build a client that solves a puzzle hosted on attu.
    - Learn how to emulate a network on a virtual machine
    - Build a switched network and a firewall
- Partner class!





CSE 457 - Graphics

- Cool, but hard
- Lots of C/C++, some linear algebra & physics
- If you want to do graphics, you have to do it in C++
- You will make cool artifacts in this course!
- Skeleton code provided is not documented well and a little bit “spaghetti” ☹



CSE 455 – Computer Vision

- Deals with low-level, mid-level, and high-level computer vision
  - Get to implement a Computer Vision C library from scratch!
  - Some topics covered include detecting features in images, creating panoramas, identifying motion in images, etc.
- Last few weeks deals with machine learning
  - “New ways” to do computer vision via ML
  - This part is in python, using mostly pytorch library
- Also an open ended final project where you can implement anything you want related to computer vision!
- Sometimes, these are the people making attu slow ☹

Shoutout: Other Classes

● How about that “mysterious” compiler?	401 Compilers
● Liked html (for some reason)?	154 Web Dev
● Want to do C on limited systems?	474 Embedded Sys
● Learn about more low-level stuff?	369 & 371 Digital Design
● Want more 351-esque concepts?	469 & 470 Comp Arch
● Liked patching up your 333gle? ssh-keys?	484 Security

TA-ing

- You are all well enough equipped to TA CSE333, CSE351, CSE374 and others.
- You do NOT have to 4.0 a class to TA it
- You do NOT have to be a super social person to TA
  (Some of us are very introverted)
- TAing will reinforce your understanding of any material
- **TAs are human too. It is okay to start off imperfect and make mistakes**
- **If you think you would be interested, we would highly recommend reaching out and giving it a try. Please feel free to talk to us if you are interested!**

Ask us Anything!!

Before We Go...

This quarter has been tough!

This section wouldn’t have been the same without all of you!

We were glad to have you! You earned your place here, you belong here (on this campus, in this program, in this course)

You know the content, take it slow, you can do it!

Evaluation Link

Put own section evaluation link here:

You've Learned A Lot! Good Luck!

Thank You for a Great Quarter!

Bonus Slides

Cool Fact: Segmentation Faults

- You've probably run afoul of SIGSEGV (a.k.a. "Seg fault")
  - What is it?
- UNIX processes can communicate with each other!
- signals are notifications sent between processes
  - They all have default handlers, such as "crash the program"
- You can use `signal()` or `sigaction()` to handle them yourself!

Cool Fact: Process Communication

- To send “real” messages between processes, you already have what you need in your toolkit!
  - Set up a socket connection from a process to itself
    - You’ll have to use nonblocking calls for this
  - `fork()`
  - Each process closes one end, and uses the other to communicate
- You can do this with TCP sockets, but there are better options available
- `socketpair()` does all of this for you!

Shoutout: The Rust Language

- No memory errors.
- No race conditions.
  - Whaaaaat? Yes.
- Performance close to C/C++ level
- Good abstractions like iterators & closures
  - Optimized down, so it’s as fast as if you wrote it by han