Winter 2025

EECS 482: Introduction to Operating Systems

Lecture 1: Introduction

Prof. Ryan Huang

https://os.eecs.umich.edu

About Me

Associate Professor in EECS

Joined U-M in Winter ‘23

- Previously an Assistant Professor at Johns Hopkins
- PhD at UC San Diego

Research on computer systems

- OS, distributed systems, cloud and mobile computing
- System reliability and fault tolerance

Office: 4611 BBB

Course staff

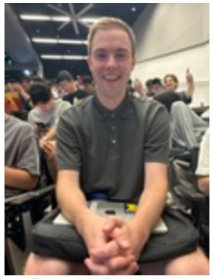

Ryan Huang

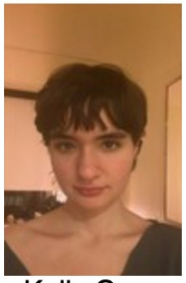

Manos Kapritsos


Anubhav Agarwal

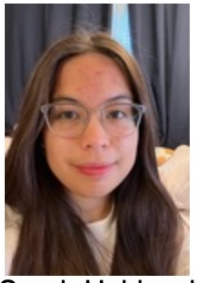

Shourya Bansal


Brady Bogda


Scott Brinley


Kelly Goss


Gary Huang

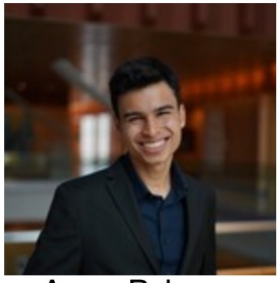

Sarah Hubbard


Kevin Jia


Matthew Jia

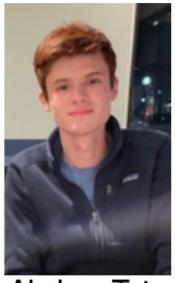

Ryan Kersten


Arnuv Peri


Aaron Rahman


Drishti Srivastava


Haytham Tang


Akshay Tate

Agenda for today

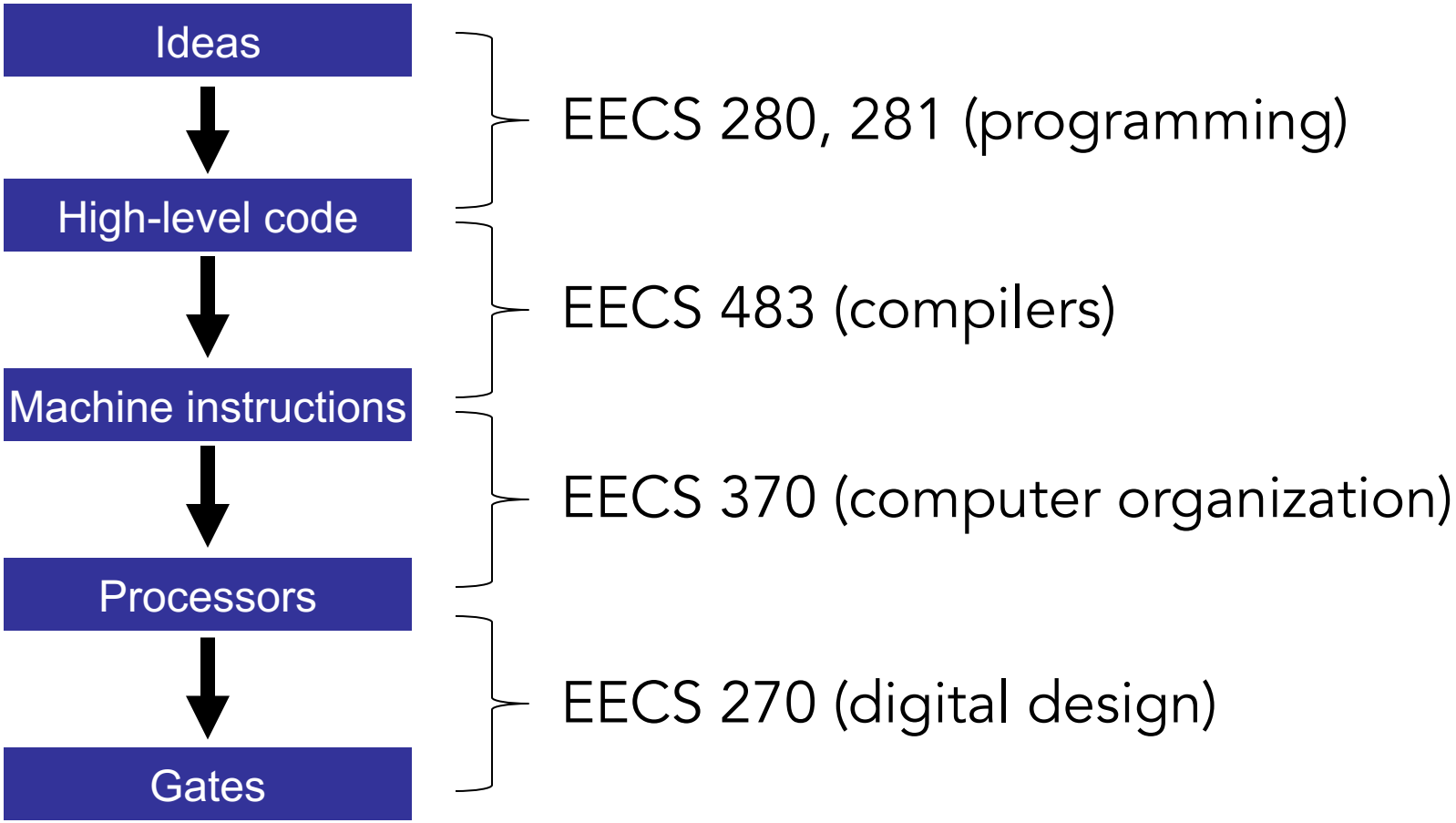
Why do we need EECS 482?

Course syllabus and logistics

Why do we need an OS and what does it do?

How did OSes evolve to what we have today?

Overview of CS/CE Curriculum



But some mysteries remain…

Bootstrapping

- How does a computer start when you turn it on?
- How to get new program into memory and start executing it?

Concurrency and I/O

- How to run multiple programs at the same time?
- How to read keyboard or mouse? Print output to screen?

Persistence and Protection

- How to organize data on disk?
- How to save your data when you turn the computer off?
- How to avoid programs breaking each other?
- How to prevent other users from accessing your data?

But some mysteries remain…

Bootstrapping

- How does a computer start when you turn it on?
- How to get new program into memory and start executing it?

The OS does all of this.

After this semester, you should be able to answer most of these questions!

- How to organize data on disk?
- How to save your data when you turn the computer off?
- How to avoid programs breaking each other?
- How to prevent other users from accessing your data?

Why study operating systems?

You may write one someday ☺

- New hardware, smart devices, self-driving cars, data centers, *etc.*

Universal abstractions, techniques, optimizations

- Concurrency, caching, indirection, naming, protection, ...

Complex software systems

- Many of you will go on to work on large software projects
- OSes are typical examples of complex systems

Understanding what you use

- Understanding the OS helps you write better apps
- Functionality, performance tuning, simplicity, etc.

Course material

Lectures are the primary references

Recommended textbook

- *Operating Systems: Principles and Practice, 2nd edition*, by Tom Anderson and Mike Dahlin

Use textbook(s) as supplementary readings

Important links

Course web page:

- https://os.eecs.umich.edu
- Syllabus, slides, projects, references will be posted here

Piazza

- https://piazza.com/umich/winter2025/eecs482
- Access code: **tpah**
- Announcements and Q&A

Lecture topics and schedule

Cover OS abstractions for H/W resources

Before Mid-term: CPU, Memory

After Mid-term: Disk, Network

End with distributed systems and case studies

Lectures

Two lecture sections (loosely synchronized)

Lectures will be interactive

Attend lecture and participate

- In person

Lectures will be recorded and available online

Labs

Attend in person

Labs will *not* be recorded

First lab section is this Friday

Pre-lab questions will be posted on course web page a week in advance

- Do them before going to your section

Projects

Four projects

- P1: Concurrent program
- P2: Thread manager
- P3: Virtual memory manager
- P4: Multi-threaded network file system

By the end of the semester, you'll have built the main parts of an operating system

Projects are HARD!

Probably one of the hardest class you will take at UM in terms of development effort

- Projects will take most of your time in this class

Why challenging?

- Not in terms of lines of code
  - My total code = 913 lines
- Instead, new concepts and lots of thinking
  - Two views: user-level software vs. hardware
  - Concurrent threads running on multiple processors
  - Multiple address spaces spread across memory and disk
  - Persistent data structures
  - Distributed computing

4-credits vs. 6-credits

What's the same?

- Lectures
- Lab sections
- Project 1
- Exams (probably)

What's different?

- 6-credit students implement advanced features in Projects 2, 3, 4
- Class time for EECS 498 structured as extra office hours by default with potential in-class instruction as needed

4-credits vs. 6-credits

	Core project	Advanced features
Project 1		
Project 2	1 CPU	> 1 CPU
Project 3	Process starts with empty arena	Process starts with copy of parent's arena
Project 4	Reader/writer locks Statically allocated locks	Upgradable reader/writer locks Dynamically allocated locks

4-credits vs. 6-credits

	Core project	Advanced features
Project 1	67 LOC	
Project 2	1 CPU 135 LOC	> 1 CPU
Project 3	Process starts with empty arena 234 LOC	Process starts with copy of parent's arena
Project 4	Reader/writer locks Statically allocated locks 402 LOC	Upgradable reader/writer locks Dynamically allocated locks
Total	838 LOC	

4-credits vs. 6-credits

	Core project	Advanced features
Project 1	67 LOC	67 (+0) LOC
Project 2	1 CPU 135 LOC	> 1 CPU 160 (+25) LOC
Project 3	Process starts with empty arena 234 LOC	Process starts with copy of parent's arena 258 (+24) LOC
Project 4	Reader/writer locks Statically allocated locks 402 LOC	Upgradable reader/writer locks Dynamically allocated locks 428 (+26) LOC
Total	838 LOC	913 (+75) LOC

Project groups

First project done individually

Remaining projects done in groups of 2 or 3

- Register your GitHub ID – we'll assign repositories
- Declare your group by January 22nd
- Post to Piazza if you don't know anyone

Project tips (1)

Choose group members carefully

- Check schedule, class goals, working style, etc.
- Mix of 4-credit and 6-credit students strongly discouraged

We'll evaluate every member's contributions

- Peer feedback
- git statistics

Group can fire one of its members (see syllabus)

Project tips (2)

Start early!

Develop project incrementally

- Build a little functionality, then test, test, test, …

Expect to spend most of your time debugging

Project tips (3)

Make good use of staff – we are here to help

- Visit office hours
- Participate in discussion on Piazza
- Look for help in lecture, lab, and textbook

Our goal is to help you solve your problems

- *Not to solve them for you*

Project policies

Development

- Github: EECS 482 will provide private repositories
- Commits should match contributions

Submission

- 1 feedback/day + 3 bonus feedbacks/project
- Due by midnight ET

Soft deadline

- You may continue to submit to AG after due date, but the extra points will be discounted (P1 will explain this)
  - Note you also have to accommodate for the next projects

Project policies cont'd

Collaboration

- Okay to clarify project handout or discuss C++ syntax
- Not okay to discuss solutions
- Not okay to benefit from past solutions

Exams (must be taken in person)

Midterm: February 26th, 6:00 pm – 8:00 pm

Final: April 28th, 7:00 pm – 9:00 pm

Administrative break

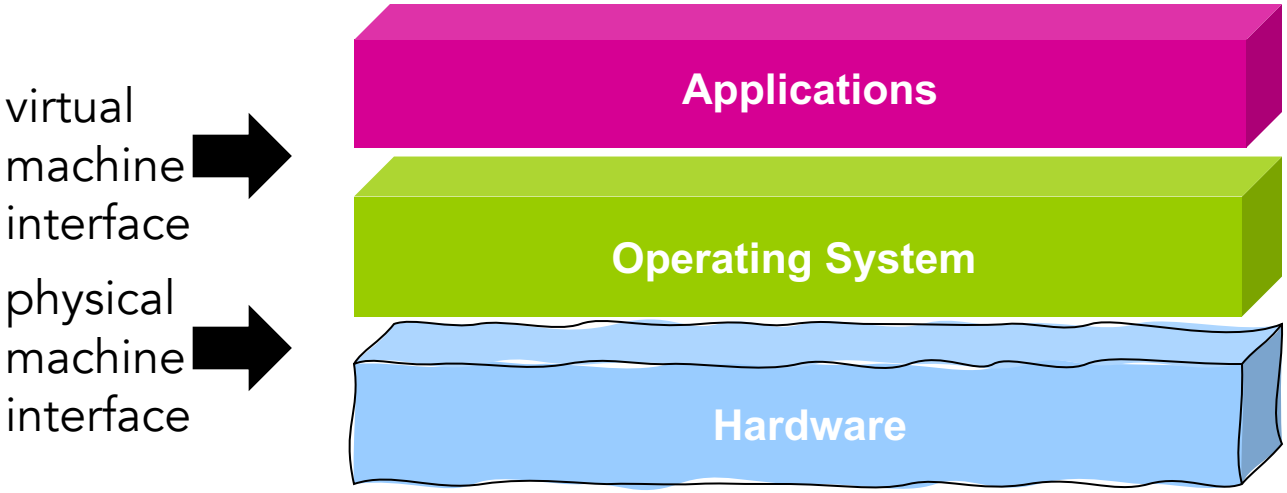
Submit your picture on the course web page

- https://os.eecs.umich.edu/
  - *"Update your student information"*
- Ask someone near you to take your picture with your phone

Look for a project group

What is an OS?

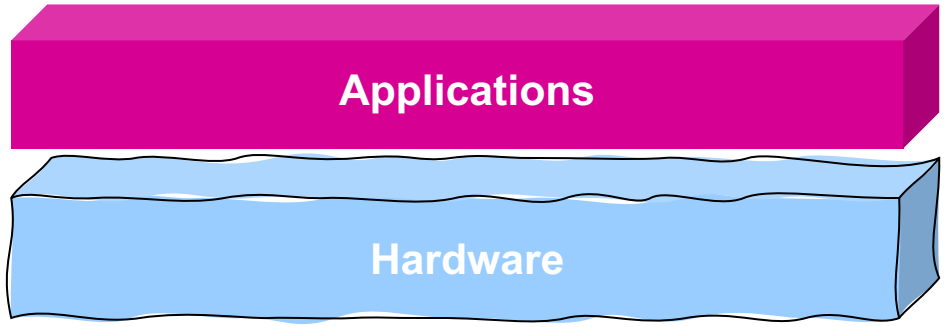
The operating system is the software layer between application programs and the hardware



Why have an OS?

What if there were no OS?

physical machine interface

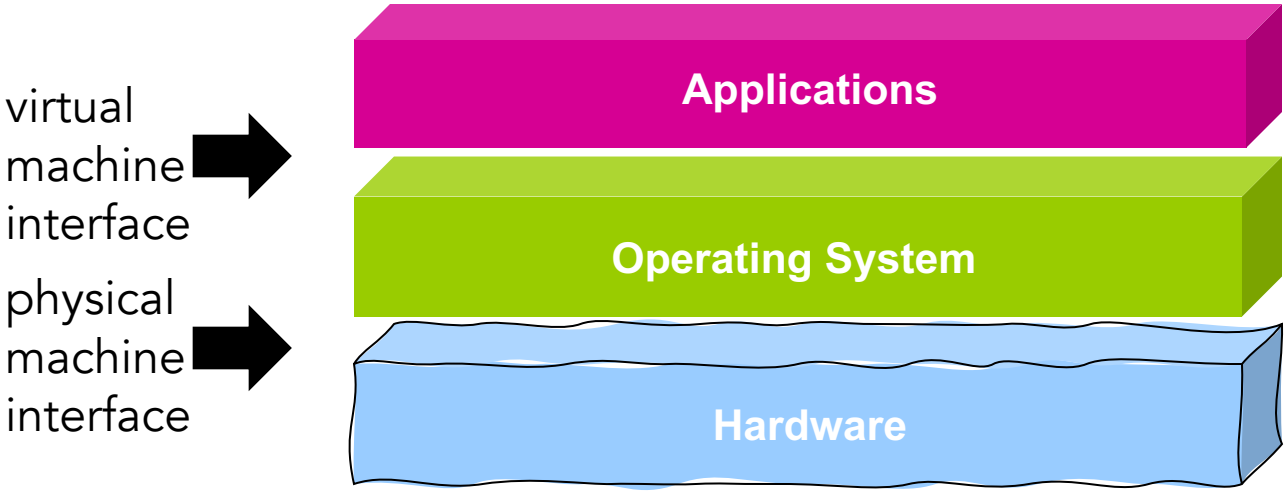


Problems

- Programming difficulty
- Portability
- Resource efficiency
- …

What is an OS?

The operating system is the software layer between application programs and the hardware



All the code that you didn't have to write to implement your app ☺

Roles of the OS: abstractions

Create abstractions to hide details of hardware

- CPU → Threads
- Memory → Address space
- Disk → Files

Benefits to applications:

- Simpler (no tweaking device registers)
- Device independent (all network cards look the same)
- Portable (across Win95/98/ME/NT/2000/XP/...)

For any area of OS, ask

- What interface does hardware present to software?
- What interface does OS present to applications?

Roles of the OS: resource management

Manage shared hardware resources

- Processor
- Memory
- Disk
- Network
- Other?

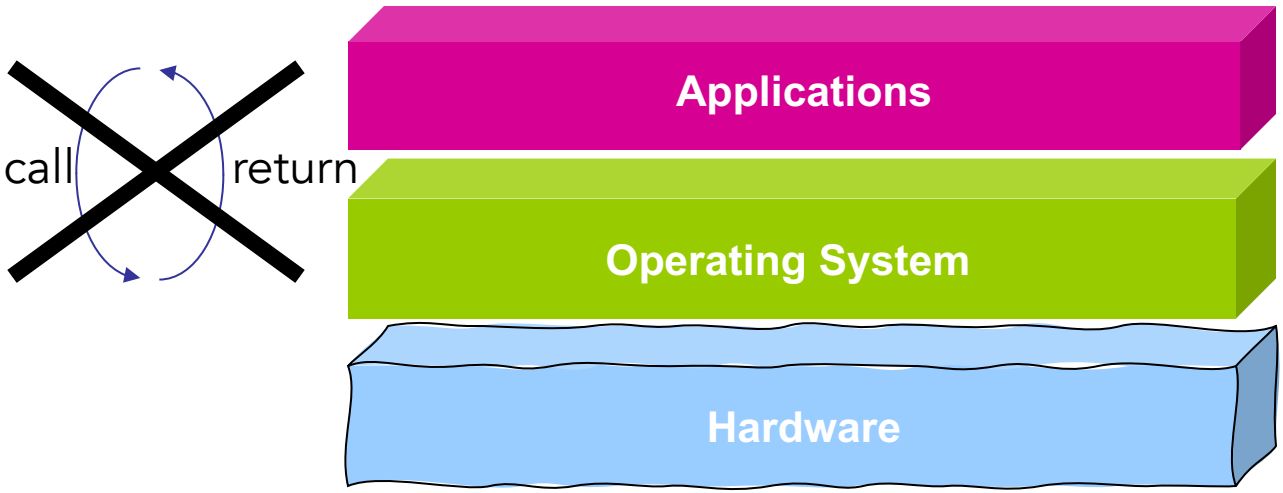
But at a cost

OS and apps: perspective 1

Application is main program

- Gets services by calling kernel (OS)
- Example: print output to the screen

This seems right, but there are some problems

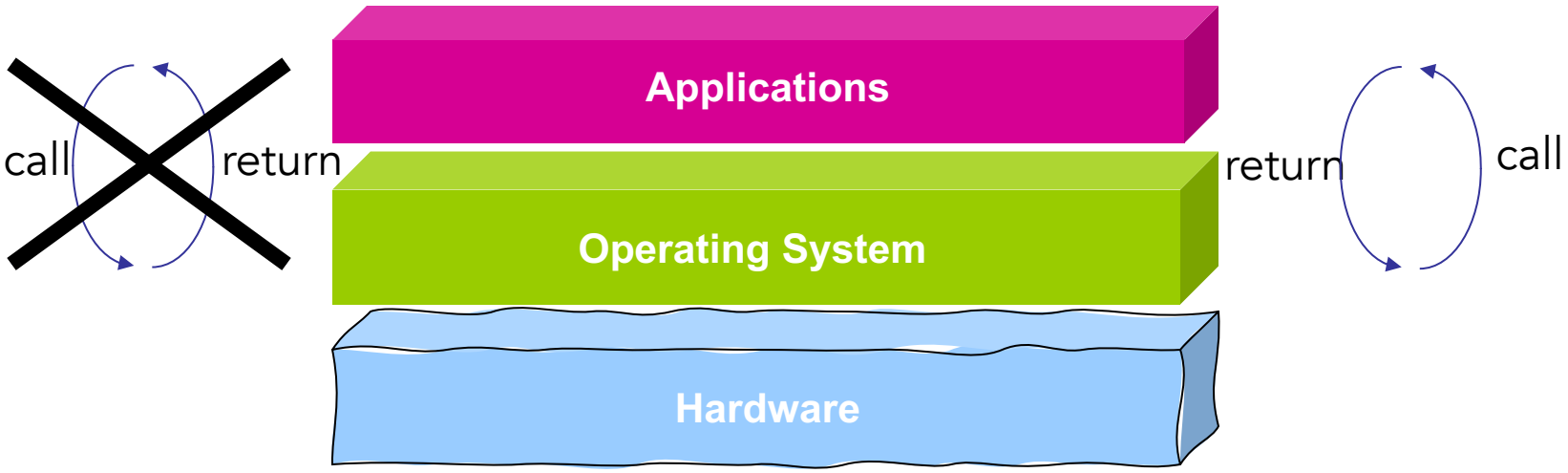


OS and apps: perspective 2

OS is main program

- Calls applications as subroutines
- Application runs until it returns control back to OS

Who is calling whom?



History of operating systems

Computing hardware was very expensive

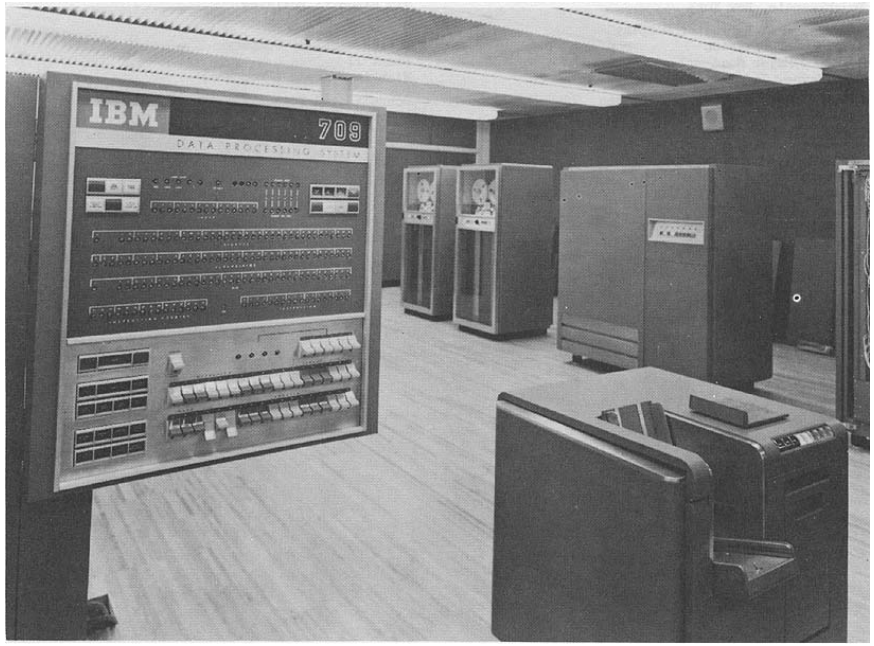
How much did an IBM 709 cost?

A: $100s

B: $1,000s

C: $10,000s

D: $1,000,000s



IBM 709 (c. late 1950~)

Single operator at console

Goal: basic functionality

OS is just a library of standard services

Pros:

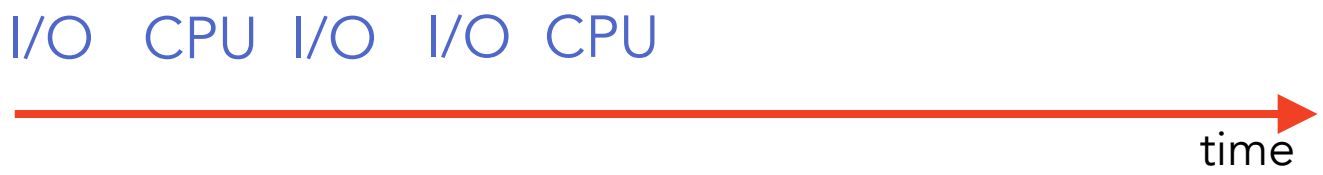
- Interactive
  - immediate response
- Simple
  - one thing happening at a time

Cons:

- Poor utilization of hardware
  - Why?
  - How to improve utilization?

Batch processing

Idea: remove/minimize user interaction



Submit a batch of jobs, wait for completion

OS is library of standard services + batch monitor

```
load the monitor;
while (batch has next job) {
   load job;
   run job to completion;
   control returns to monitor;
}
```

Batch processing (2)

Submit a batch of jobs, wait for completion

OS is library of standard services + batch monitor

Improved utilization

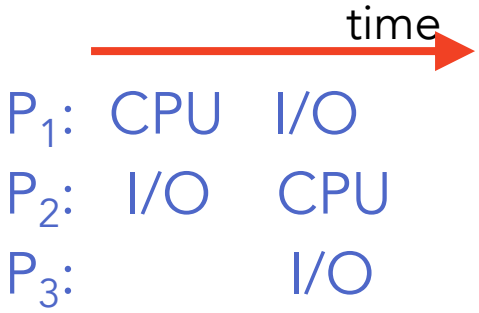
- Reduce idle time (to switch jobs)

Hardware can be still under utilized

- Each job runs to completion
  - a job waits for I/O → CPU becomes idle

Multi-programmed batch

Idea: Overlap CPU and I/O from different jobs



High utilization of hardware

OS becomes more complex

- Support concurrent execution of multiple processes
- Must ensure OS is not corrupted by program
- Must protect processes from each other

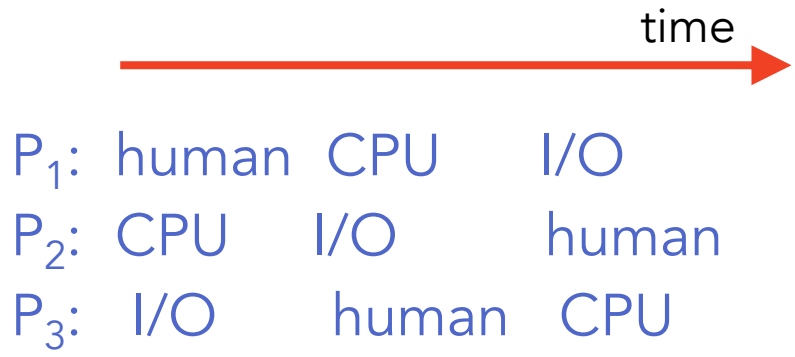
Still not interactive

Time sharing

Goal: allow humans to interact with running programs

Insight: model the user as a (very slow) I/O device

- Switch between processes while waiting for user



OS is now even more complicated

- Lots of simultaneous jobs
- Multiple sources of new jobs

History of operating systems

OS started out very simple, then became complex to use the (expensive!) hardware efficiently

But hardware is no longer expensive! Do we still need these OS advances?

- with cheap processors, do we need to run concurrent jobs?
- with single-user systems, do we still need protection?

Trends in PC operating systems

Looking ahead

Operating systems continue to evolve

- Smartphones: Android, iOS, …
- Servers: virtual machines and hypervisors
- Cloud: Amazon EC2, Microsoft Azure, Google Cloud, …

What drive those changes?

- New hardware
- New application/workloads
- New requirements

Things to do...

Browse the course web page

Subscribe to Piazza

Register GitHub ID

Start finding partners for project group

Attend lab section this Friday