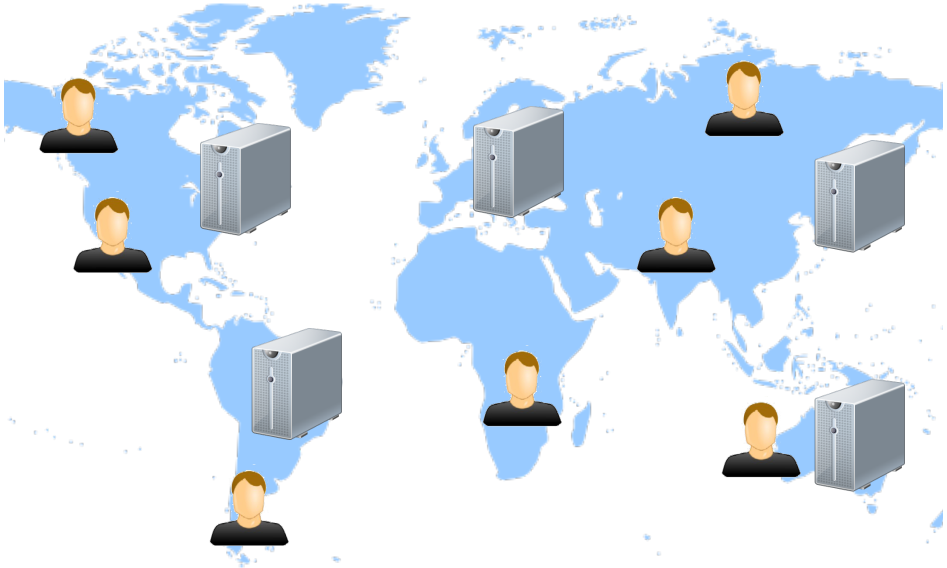
EECS 482 Introduction to Operating Systems

Winter 2018

Harsha V. Madhyastha

About Me

- PhD, University of Washington
- Area of Research: Distributed Systems



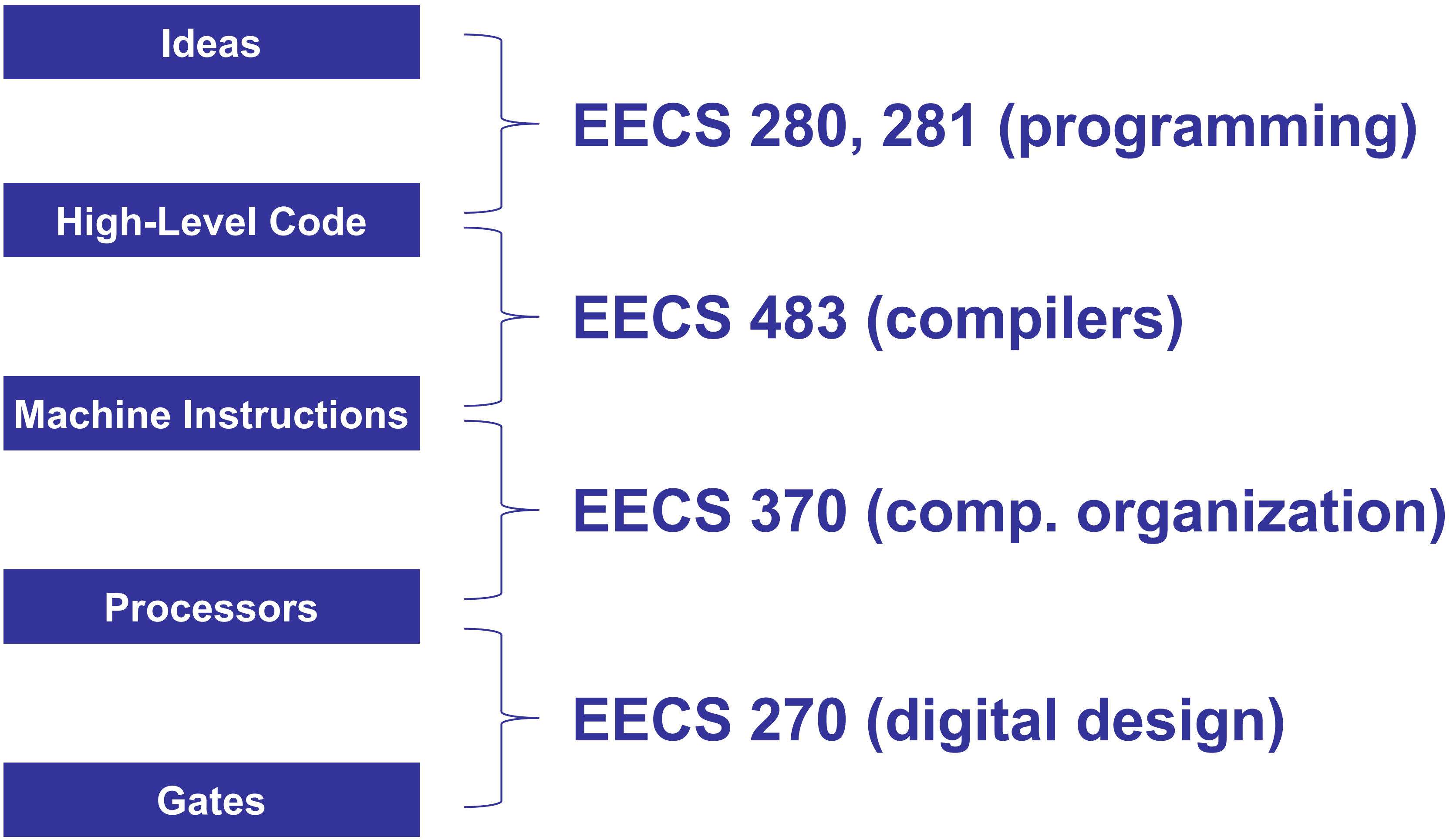
Acknowledgments

- Peter Chen
- Jason Flinn
- Manos Kapritsos

Agenda for Today

- Why do we need 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?

482 in EECS Curriculum



Ideas

High-Level Code

Machine Instructions

Processors

Gates

EECS 280, 281 (programming)

EECS 483 (compilers)

EECS 370 (comp. organization)

EECS 270 (digital design)

What is missing?

- Bootstrap:
  - How does a computer start when you turn it on?
  - How to get a program into memory and have the CPU start executing it?
- Concurrent execution with I/O:
  - How to read keyboard or mouse? Print output to screen?
  - How to run multiple programs at the same time, without one breaking the other?
- Persistence and security:
  - How to save your data when you turn the computer off? How to prevent other users from accessing your data?

What is missing?

- Bootstrap:
  - How does a computer start when you turn it on?
  - CPU

The OS does all of this.

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

- C
  - screen?
  - vithout
- Persistence and security:
  - How to save your data when you turn the computer off? How to prevent other users from accessing your data?

Why take an OS class?

- Understanding what you use
  - Understanding the OS helps you write better apps
  - Functionality, performance tuning, simplicity, etc.
- Universal abstractions and optimizations
  - Caching, indirection, naming, atomicity, protection,…
  - Examples: Cloud computing, Web services, mobile
- Mastering concurrency
  - Performance today achieved through parallelism
  - Mastery required to be a top-notch developer

Objectives of this class

- We will understand principles of concurrency
  - One paradigm: multi-threaded program
  - Principles apply to other forms (e.g., event-based)

- We will study design principles of an OS
  - This course is not about specifics of any particular OS

- We will develop an understanding of OS impact on application performance and reliability

Class Material

- Class webpage
  - http://web.eecs.umich.edu/~harshavm/eecs482/
  - Also linked from Canvas

- Syllabus, course calendar, slides, homeworks, and projects will be posted on class webpage

- Subscribe to Piazza
  - Announcements and class discussion

Lecture Schedule

- Cover how OS abstracts H/W resources
- Before mid-term: CPU, memory
- After mid-term: Network, storage
- End with distributed systems and case studies

Lectures

- Lecture videos will be posted online
- Lecture slides on course web page
  - Bring print outs to class
- Textbook (highly recommended):
  - Anderson and Dahlin, "*Operating Systems: Principles and Practice*"

Discussion Sections

Philip Dakin

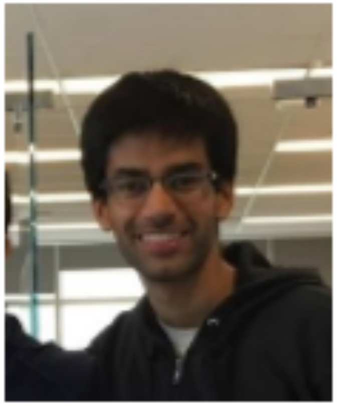
Manav Gabhawala

Zhongren Gao

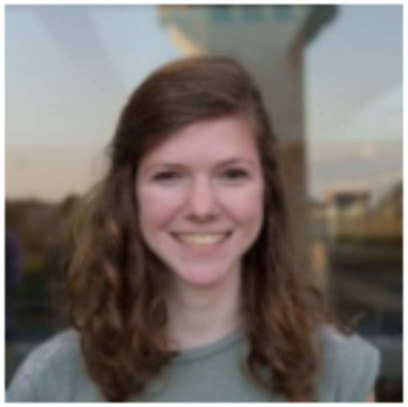
Thomas Oliver

Benjamin Reeves

Steven Schulte



Aman Sharma



Alyssa Wagenmaker

Discussion Sections

- Questions to be discussed will be posted on course web page a week in advance
  - Do them **before** going to your section
  - Prepares you for exams

- No discussion sections this Friday

Projects

- 4 projects
  - Writing a concurrent program
  - Thread manager
  - Virtual memory pager
  - Multi-threaded secure network file system

- First one individually, others in groups of 2 or 3
  - Register your GitHub ID – we'll assign repositories
  - Declare your group (by 1/22)
  - Post to Piazza if you don't know anyone

Projects are HARD!

- Probably the hardest class you will take at UM in terms of development effort
  - Projects will take 95% of your time in this class

- Reason for being hard:
  - Not number of lines of code
  - Instead, new concepts!

Project recommendations

- Choose group members carefully
    - Check schedule, class goals, style, etc.
- We'll evaluate every member's contributions
    - Peer feedback
    - git log and github statistics
- Group can fire one of its members (see syllabus)

Project recommendations

- Do not start working on projects at last minute!
  - Projects are autograded
  - No. of hours you put in or lines of code don’t count
  - Testing is integral process of development

- Make good use of help available
  - 20 office hours per week (3x when projects are due)
  - Monitor and participate in discussion on Piazza
  - Hints during lectures, discussions, and textbook

Policies

- Submission
  - 1 submission per day to autograder + 3 bonus
  - Due at midnight (hard deadline!)
  - 3 late days across all projects

- Collaboration
  - Okay to clarify problem or discuss C++ syntax
  - Not okay to discuss solutions
  - Not okay to borrow from past solutions

Exams

- Midterm: February 21st (6:30 – 8:30pm)
- Final: April 23rd (7 – 9pm)
- No makeup exams
  - Unless dire circumstances
  - Make sure you schedule interviews appropriately

Grading breakdown

- Projects:
  - Project 1: 3%
  - Projects 2, 3, and 4: 15% each
- Mid-term and Final: 26% each

Enrollments

- Attend section you are enrolled in
  - Exams may have lecture-specific questions

- Overrides
  - Currently near cap for course staffing

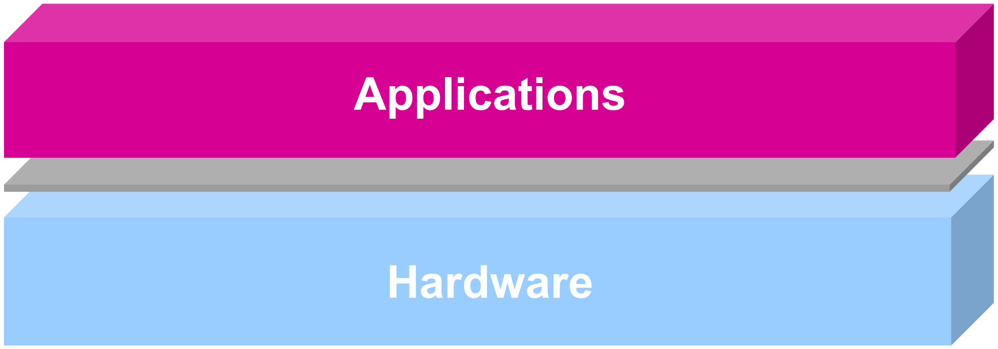
- Talk to me if you are retaking this class

Pro tips for success in 482

1. Start early on projects and pick group wisely
2. Leverage github and communicate with team
3. Take advantage of available help
   - Go to office hours, post/monitor questions on Piazza
4. Attend lectures and discussions
   - Read textbook, solve questions before discussion
5. Ask questions when something is unclear

Why have an OS?

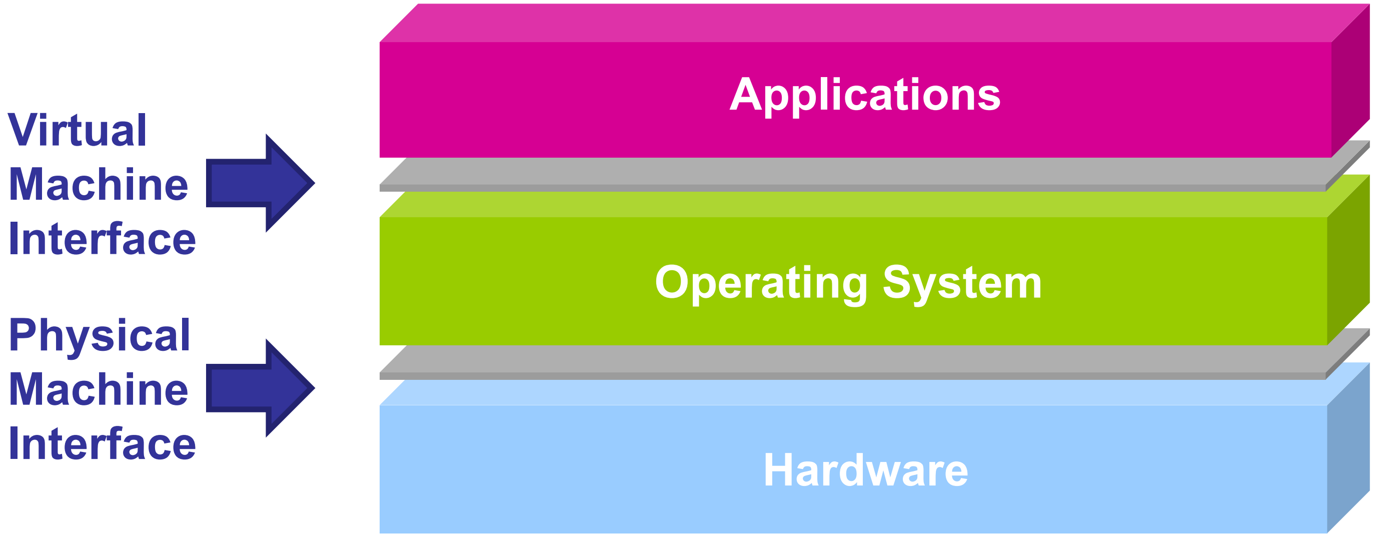
- What if applications ran directly on hardware?



- Problems:
  - Portability
  - Resource sharing

What is an OS?

- The operating system is the software layer between user applications and the hardware



- OS is "*all the code that you don't have to write*" to implement your application

Roles of the OS

- Illusionist: Create abstractions to ease use of hardware
  - CPU → Threads
  - Memory → Address space
- For any area of OS, ask
  - What interface does hardware present?
  - What interface does OS present to applications?
- Government: Manage shared hardware resources
  - But at a cost (taxes)

OS and Apps: 2 Perspectives

- Perspective 1: application is main program
  - Gets services by calling kernel (OS)
  - Example: Print output to the screen

- Problems with this view:
  - how does application start?
  - how do tasks occurring outside any program (e.g. receiving network packets) get done?
  - how do multiple programs run simultaneously without messing each other up?

OS and Apps: 2 Perspectives

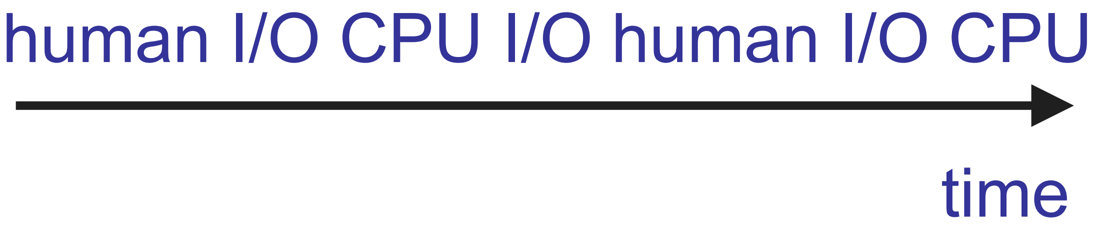
- Perspective 2: OS is main program
  - Calls applications as subroutines
  - Illusion: every app runs on its own computer

- Lower layer (OS) invokes higher layer (apps)!
- App or processor returns control to OS

- Correct perspective, but what is it that makes the OS the "main" program?

History of operating systems

- Single operator at console



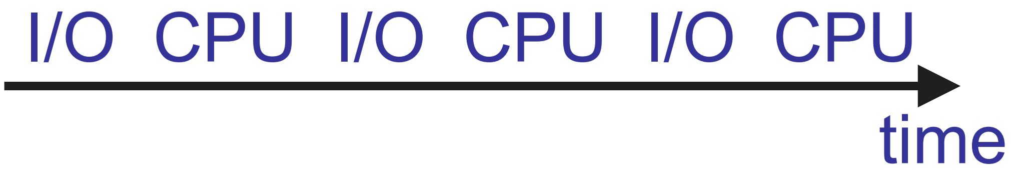
human I/O CPU I/O human I/O CPU

time

- Positives:
  - Interactive
  - Very simple
- Downside:
  - Poor utilization of hardware

History of operating systems

- Batch processing
  - Goal: Improve CPU and I/O utilization by removing user interaction



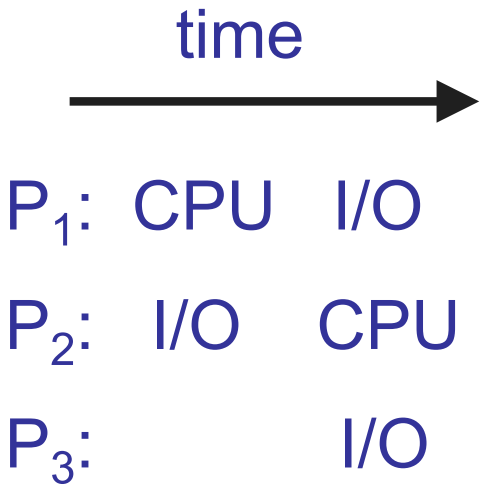
I/O CPU I/O CPU I/O CPU

time

- OS is batch monitor + library of standard services
- Protection becomes an issue
  - Why wasn’t this an issue for single operator at console?

History of operating systems

- Multi-programmed batch
  - Improve utilization further by overlapping CPU and I/O



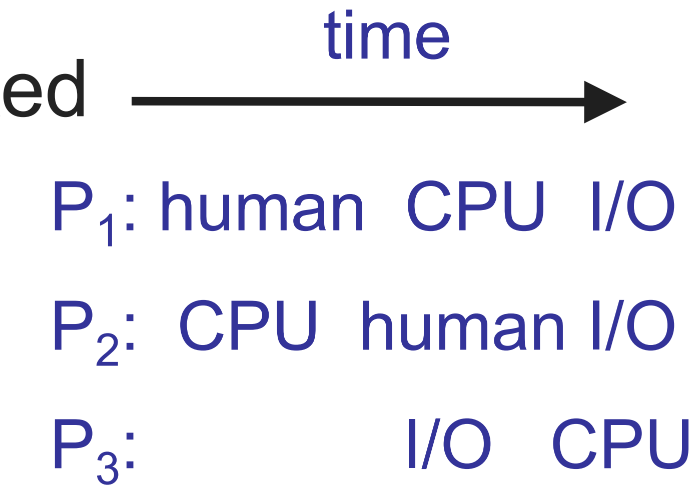
P_1:	CPU	I/O
P_2:	I/O	CPU
P_3:		I/O

- OS becomes more complex
  - Runs multiple processes concurrently, allowing simultaneous CPU and I/O
  - Multiple I/Os can take place simultaneously
  - Protects processes from each other
  - Still not interactive

History of operating systems

- Time sharing
  - Goal: Allow people to interact with programs as they run
  - Insight: User can be modeled 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



P_1:	human	CPU	I/O
P_2:	CPU	human	I/O
P_3:		I/O	CPU

History of operating systems

- OS started out very simple
- Became complex to use hardware efficiently
- Today: Personal computers
  - Is the main assumption (hardware is expensive) still true?
- How does this affect OS design?
  - PCs don’t need to time share between multiple jobs?
  - PCs don’t need protection between multiple jobs?
- PCs gradually added back time-sharing features

Looking ahead …

- OSes continue to evolve
  - Cloud: Amazon EC2, Microsoft Azure, …
  - Smartphones: Android, iOS, …

- What are the drivers of OS change?
  - New app requirements
  - New objectives

Thing to do …

- Browse the course web page

- Subscribe to Piazza
- Register GitHub ID
- Start finding partners for project group

- No discussion section on Friday