EECS 482 Introduction to Operating Systems

Winter 2018

Baris Kasikci
barisk@umich.edu
(Thanks, Harsha Madhyastha for the slides!)

About Me

- Prof. Kasikci (Prof. K.), Prof. Baris (Prof. Barish)
- Assistant Professor
  - *Joined Michigan in Fall'17*
  - *PhD from EPFL*
  - *Previously, researcher at Microsoft Research*
  - *Previously, an embedded systems developer*
- Interests: system reliability, security, performance
  - *Employ a mix of methods from Operating Systems, Programming Languages, Software Engineering, Computer Architecture*

About You

- Please take a selfie an send me now to barisk@umich.edu
- Please contact eecs482@umich.edu if you need special arrangement for any disabilities
- Come talk to me
  - *BBB 4816, my door is always open*
  - *Anytime about career, life, any difficulties you are facing, hard decisions, etc.*
  - *During office hours about 482*

EECS 482 Staff

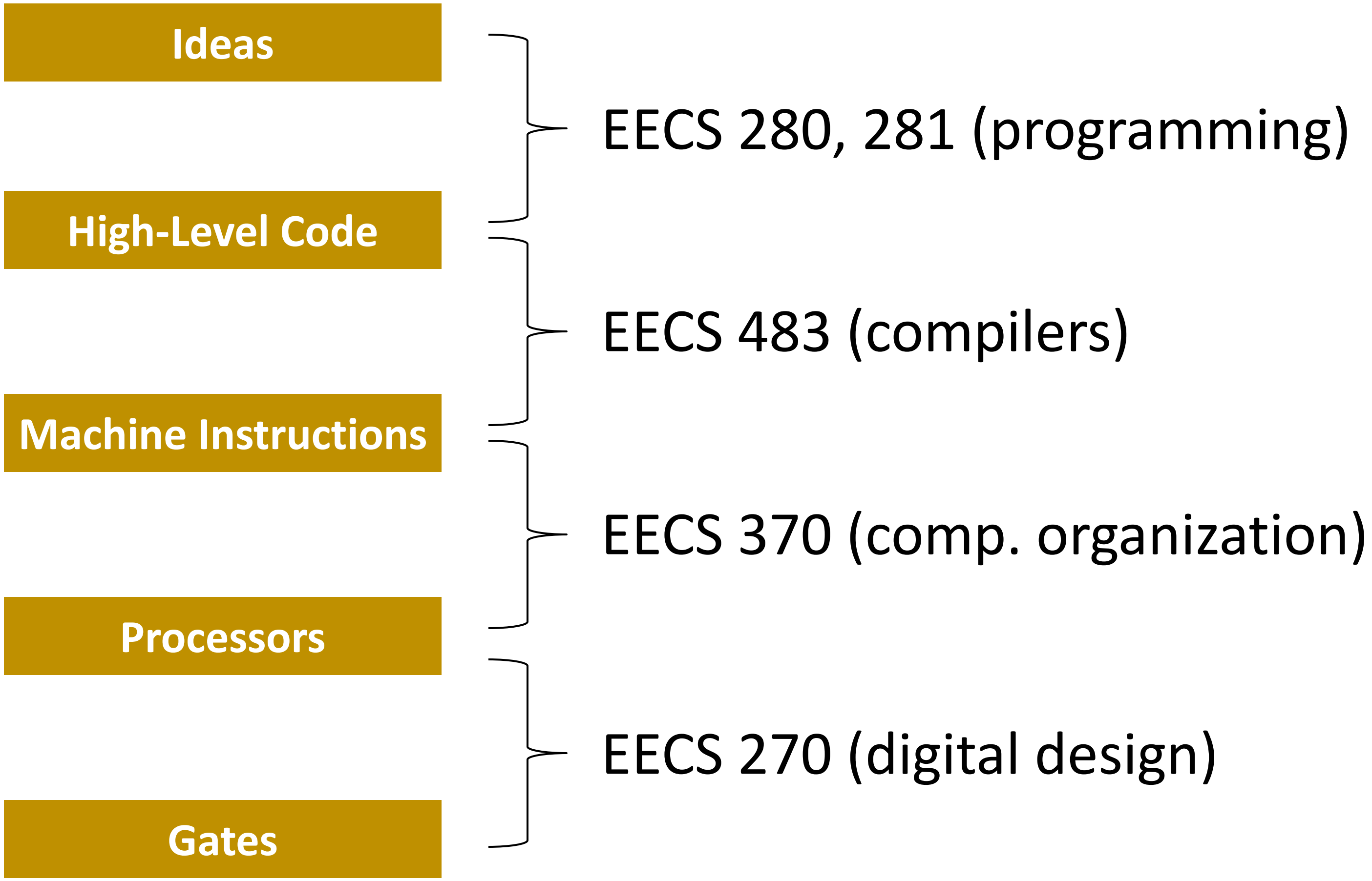
- Instructors
- GSIs & IAs

Philip Dakin 1695 Beyster	Manav Gabhawala 1695 Beyster	Zhongren Gao 1695 Beyster	Thomas Oliver 1695 Beyster	Benjamin Reeves 1695 Beyster	Steven Schulte 1695 Beyster	Aman Sharma 1695 Beyster	Alyssa Wagenmaker 1695 Beyster

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?

Neurons to silicon?



What is missing?

- Bootstrapping:
  - *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?*
  - *How can multiple users use the same computer securely?*

What is missing?

- Bootstrapping:
  -
  - ecuting it?
- Co
  -
  - breaking
- Pe
  -
  -
  -

You will be able to answer all these questions by the end of EECS 482

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
  - *What causes your program to crash when you dereference NULL?*
  - *How can multithreaded code be slower than single-threaded code?*

Prerequisites

- EECS 281
- EECS 370
- Extensive C/C++ programming experience (STL)
- Familiarity with UNIX
- Understanding of computer architecture
    - Stack pointer
    - Program counter
    - Low-level execution of a program
    - Etc.
- Some understanding of paging, TLB, caching

Class Homepage and Tools

- Class webpage
  - *http://web.eecs.umich.edu/~harshavm/eecs482/*
  - *Syllabus, slides, homework, etc. posted on class webpage*
  - *Subscribe to Piazza!*
  - Announcements and class discussion

Lecture Schedule

- CPU (threads and concurrency)
- Memory (address spaces)
- Midterm
- Network (sockets)
- Storage (file systems)
- Aggregation: distributed systems and case studies

Lectures

- 2 sections
  - *Mostly synchronized, exams will have a few different questions*
- Lecture captured (videos online)
- Slides and lecture notes will be posted on the webpage
- Textbook (highly recommended):
  - Anderson and Dahlin, “Operating Systems: Principles and Practice”
  - Additional readings posted on the webpage

Lab/Discussion Sections

- OK to attend any discussion
  - *As long as there are seats*

Questions posted several days in advance
  - *Do them **before** going to your section*
  - *This prepares you well for exams*
  - *Covers some background knowledge*

- **No Discussion Sessions This Friday!**

Projects

- 4 projects
  - *Writing a concurrent program*
  - *Thread manager*
  - *Virtual memory pager*
  - *Multi-threaded secure network file system*
- First is individual, do others in groups of 2-3
  - *Register your GitHub id – we'll assign repositories*
  - *Declare your group (by 1/22) via course web page*
  - *Mail eecs482@umich.edu if taken 482 before*
    - Can't reuse any code except for project 1.

Project recommendations

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

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: threads, interrupts, address spaces, name spaces etc.

Project recommendations

- Do not start working on projects at last minute!
  - Projects are autograded (must be mostly correct)
  - 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 (extra hours when projects are due)
  - There will be long queues
  - Monitor and participate in discussion on Piazza
  - Hints during lectures, discussions (also in textbook!)

Policies

- Submission
  - *1 submission per day to autograder + 3 bonus*
  - *Due at midnight (hard deadline!)*
  - *3 late days budget across all projects (if you hand in your project two days late, you will have one late day left)*
- Collaboration
  - *Okay to clarify problem or discuss C++ syntax*
  - *Not okay to discuss solutions*
  - *Past solutions a real problem (several HC cases)*

Exams (Tentative!)

- Midterm: February 21st (6:30-8:30pm)
- Final: April 23th (7-9pm)
- No makeup exams
  - Unless dire circumstances
  - Make sure you schedule interviews appropriately
  - E-mail me (eecs482@umich.edu) with exceptions/conflicts

Grading breakdown

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

Enrollment

Must have prerequisites (281 & 370 or equivalent)

Overrides

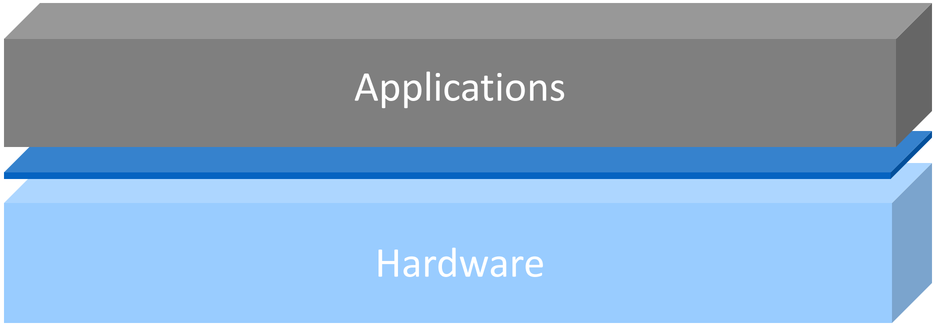
- *Currently near cap for course staffing*
- *Hope many can enroll due to normal churn*

Pro tips for success in 482

- **Start early** on projects
- Leverage GitHub and communicate with team
- Take advantage of available help
  - Go to office hours, post/monitor questions on Piazza
- Attend lectures and discussions
  - Read textbook, solve questions before discussion
- Ask questions when something is unclear

Why have an OS?

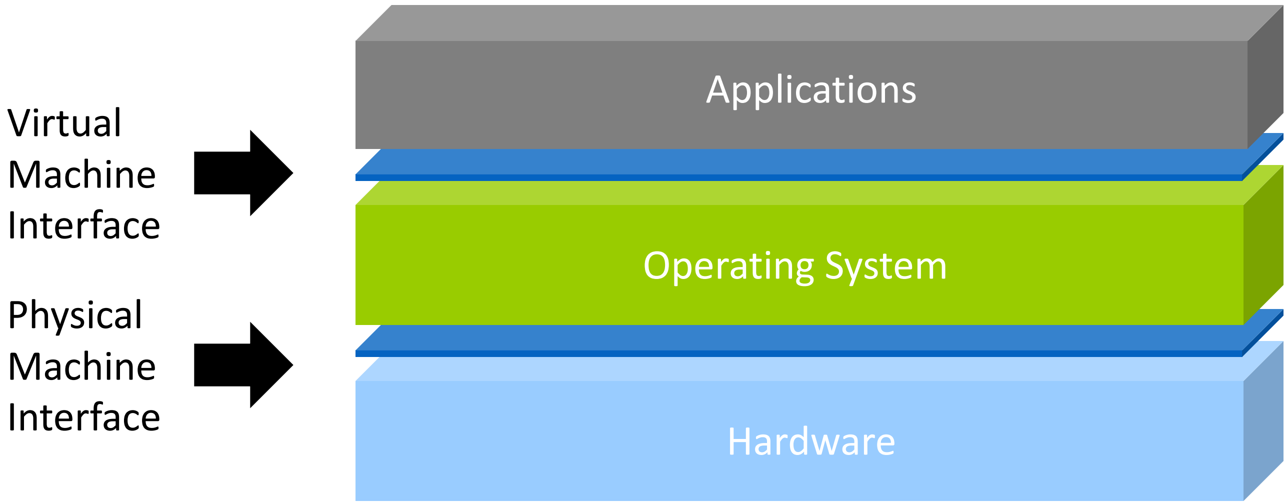
- What if applications ran directly on hardware?



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

OS and Apps: 2 Perspectives

- Perspective 1: application is main program
  - *Gets services by calling kernel (OS)*
  - *Example: print this 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 Applications

- 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?

Why take an OS class? - 1

- Mastering concurrency
  - Performance today achieved through parallelism
  - Mastery required to be a top-notch developer
- 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

Why take an OS class? - 2

- Build an OS
- Concepts reused in many applications
  - *Google's web server farm*
  - *Amazon Web Services (time-shared)*
  - *Hypervisors (VMWare ESX server)*
  - *NVDIA device driver*
- Software development
  - *Design an abstraction*
  - *Make it efficiently usable by others*
- Design-related interview questions

History of operating systems

- Single operator at console

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

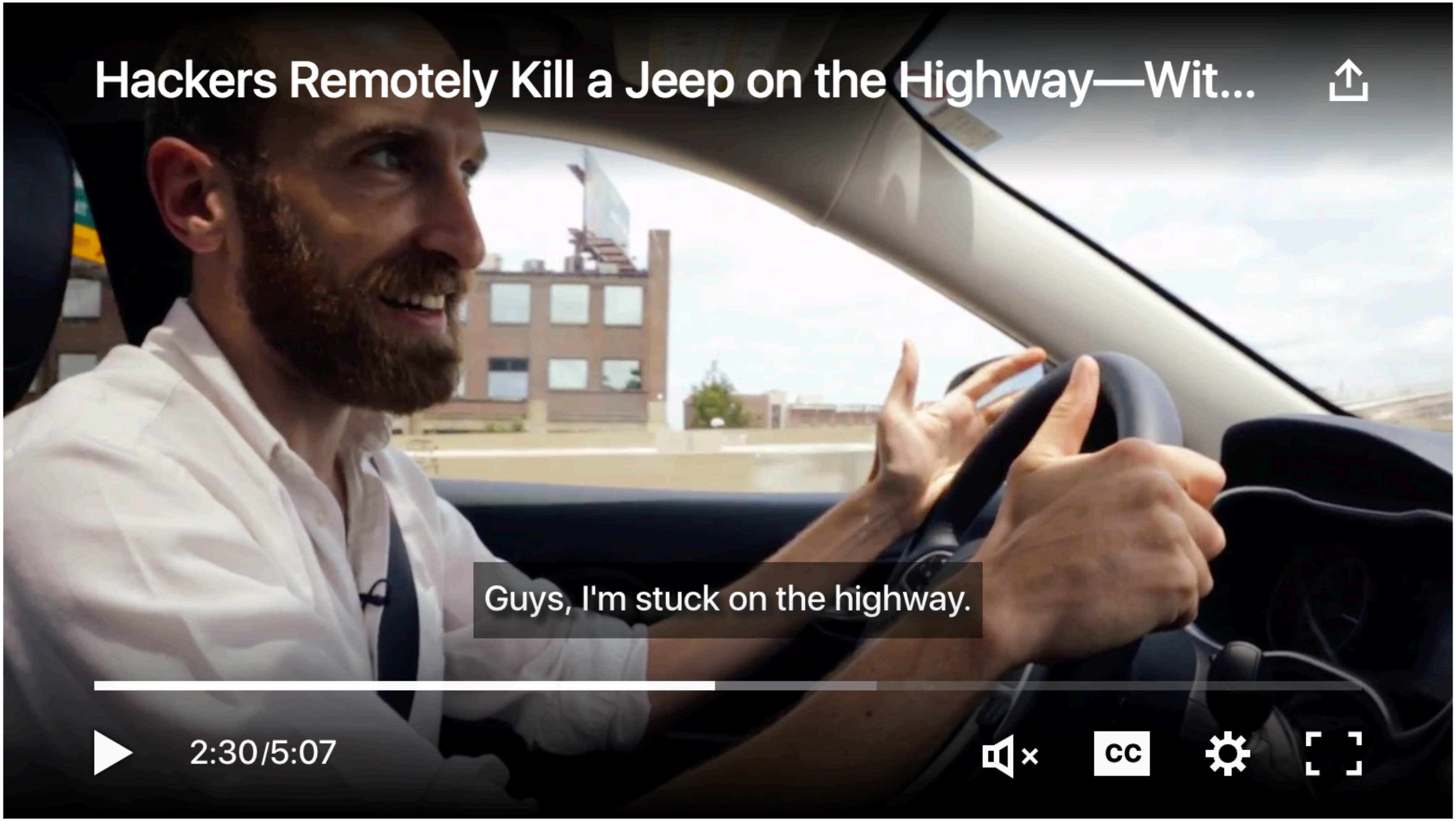
time

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

History of operating systems

- Batch processing (using punchcards)
  - *Goal: Improve CPU and I/O utilization by removing user interaction*

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



https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

History of operating systems

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

P_1: CPU Disk CPU Print

P_2: Disk CPU Print CPU Print

P_3: Disk CPU

time

History of operating systems

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

OS becomes more complex!

- Runs multiple processes concurrently
- Enables simultaneous CPU and I/O
- Multiple I/Os take place simultaneously
- Protects processes from each other
- But, 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*

P_1: CPU Disk CPU Print

P_2: User CPU User CPU User

P_3: **User** Disk CPU

time

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 complex

Lots of simultaneous jobs

Multiple sources of new jobs (people can start new jobs)

Interactivity is restored

History of operating systems

- OS started out very simple
  - *Became complex to use hardware efficiently*

- Consider PCs and workstations:
  - *Is the main assumption (hardware is expensive) still true?*

- How does this affect OS design?
  - *Don't PCs need to time share between multiple jobs?*
  - *Don't PCs need protection between multiple jobs?*

PCs gradually added back time-sharing features

What about today?

- Cloud computing (e.g. Amazon EC2)
  - *Is hardware expensive?*
  - *What other OS features are needed?*
- Mobile computing (e.g., Android/iOS)
  - *What drives efficiency?*
  - *What OS features are needed?*

Questions to Ponder

- Somewhat surprisingly, OSes continue to evolve
  - What are the drivers of OS change?
    - New hardware, security, energy

    Linux virtual memory system overhaul:
    https://www.theregister.co.uk/2018/01/02/intel_cpu_design_flaw/

- What is part of an OS? What is not?
  - Is the windowing system part of an OS?
  - OS research has become Dist. Systems research

TODOs

- Browse the course web page
- Subscribe to Piazza
- Register your GitHub id
- Start finding partners for project group (Jan 22)