Based on slides by Harsha V. Madhyastha

EECS 482 Introduction to Operating Systems

Spring/Summer 2020

Lecture 1: Introduction

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Nicole Hamilton

nham@umich.edu
https://web.eecs.umich.edu/~nham/
C: 425-765-9574

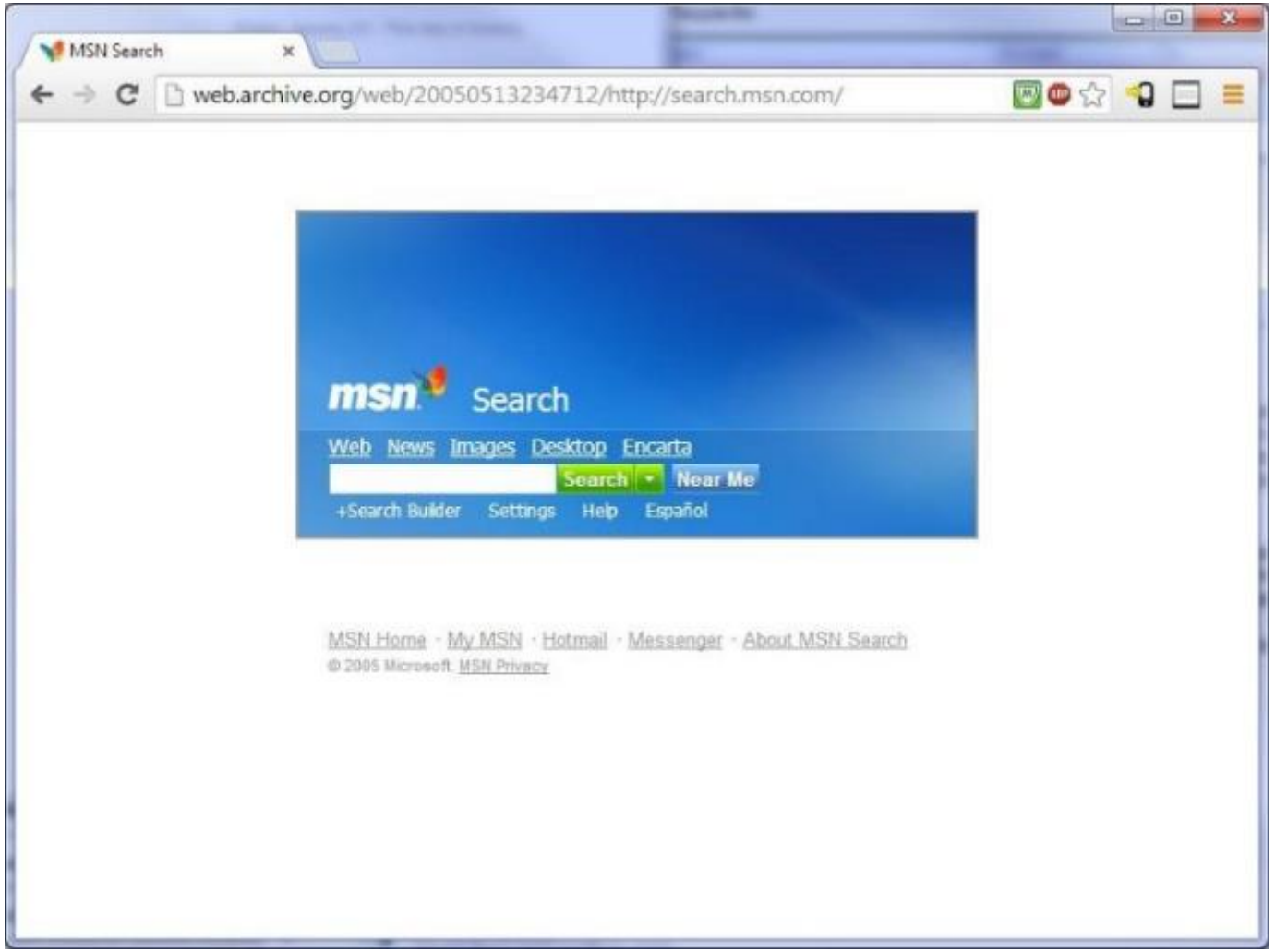
Office hours via Zoom
Meeting ID 285-289-4520
MW 4:30 to 6:00 pm EDT

Education

- BS & MS EE, Stanford, 1973.
- MBA, Boston University, 1987.

Background

- Arrived here in Fall 2017.
- Started my career doing hardware design at IBM but quickly moved into software.
- Spent most of my career as an entrepreneur selling a C shell I wrote for Windows.
- When the dot-com collapse hit, I went to Microsoft, where I wrote the ranker and query language for the first release of what's now the Bing search engine.
- Thought I was retired 6 years ago when I volunteered to advise some Capstone teams of seniors in EE at University of Washington Bothell. Turned out it paid, I loved it, one thing led to another and here I am.



MSN Search in early 2005.

Joined the team in July 2003 as the ninth member.

The ranker was the last major piece no one had taken.

Wrote the ranker and the query language for the first release in January 2005.

Almost 30 KLOC in C++.

Hamilton C shell - Wikipe

Secure | https://en.wikipedia.org/wiki/Hamilton_C_shell

Msnicki Talk Sandbox Preferences Beta Watchlist Contributions Log out

Article Talk Read Edit View history More TW Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item

Hamilton C shell

From Wikipedia, the free encyclopedia

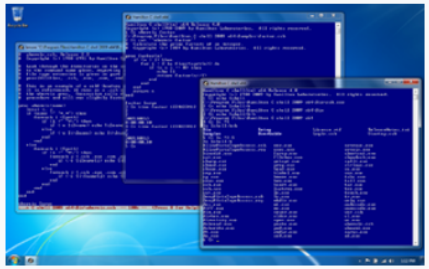
Hamilton C shell is a clone of the Unix C shell and utilities[1][2] for Microsoft Windows created by Nicole Hamilton[3] at Hamilton Laboratories as a completely original work, not based on any prior code. It was first released on OS/2 on December 12, 1988[4][5][6][7][8][9] and on Windows NT in July 1992.[10][11][12] The OS/2 version was discontinued in 2003 but the Windows version continues to be actively supported.

Contents [hide]

Design [edit]

Hamilton C shell differs from the Unix C shell in several respects, its compiler architecture, its use of threads, and the decision to follow

Hamilton C shell



64-bit Hamilton C shell on a Windows 7 desktop.

Original author(s)	Nicole Hamilton
Initial release	December 12, 1988; 28 years ago
Stable release	5.2.g / March 5, 2017; 5 months ago
Written in	C
Operating system	Windows
Type	Unix Shell on Windows
License	Commercial proprietary

Nicole Hamilton

Secure | https://web.eecs.umich.edu/~nham/

COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF MICHIGAN

- Curriculum vitae
- My faculty page at Univerity of Washington
- My personal website
- Hamilton C shell homepage
- Free Hamilton C shell download for UMich students and faculty
- Credentials
- Career
- My accidental path to Ann Arbor
- Personal

Nicole Hamilton

Lecturer III
Computer Science and Engineering
University of Michigan
2649 Beyster
2260 Hayward Street
Ann Arbor, MI 48109-2121

C: 425-765-9574
nham@umich.edu

I'm new to the University of Michigan, starting fall semester 2017. Before this, I was at University of Washington Bothell for four years in the electrical engineering department.

Fall 2017 and winter 2018, I'll be teaching one section of EECS 280, C++ and object oriented programming. Winter 2018, I'll also be teaching EECS 398 System Design in C++. I also serve on our lecturer search committee and as an undergraduate advisor.

We have about 950 students in EECS 280, split across five lecture sections (about 250 in mine). My largest class at Bothell was only 48 students (our room capacity), so I'm enjoying learning how a course this large is managed. (Answer: Staff and autograding.)

Credentials

I have a BS and MS EE from Stanford (1973) and an MBA valedictorian from Boston University (1987). I am a life senior member of the IEEE and a registered professional engineer in Texas and Massachusetts. In 2014, I was inducted as an eminent engineer by the Stanford chapter of Tau Beta Pi.

I have 10 issued patents and my publications have received over 1800 citations. A paper on search engine ranking I co-authored at Microsoft won a *10-year test-of-time award* at the 2015 International Conference on Machine Learning.

Career

Staff

Austin Kiekintveld
akiek

Brandon Kayes
Lab instructor
bkayes

Celine Schluetuer
celinesc

Email sent to eecs482@umich.edu goes to all of us.

Agenda for Today

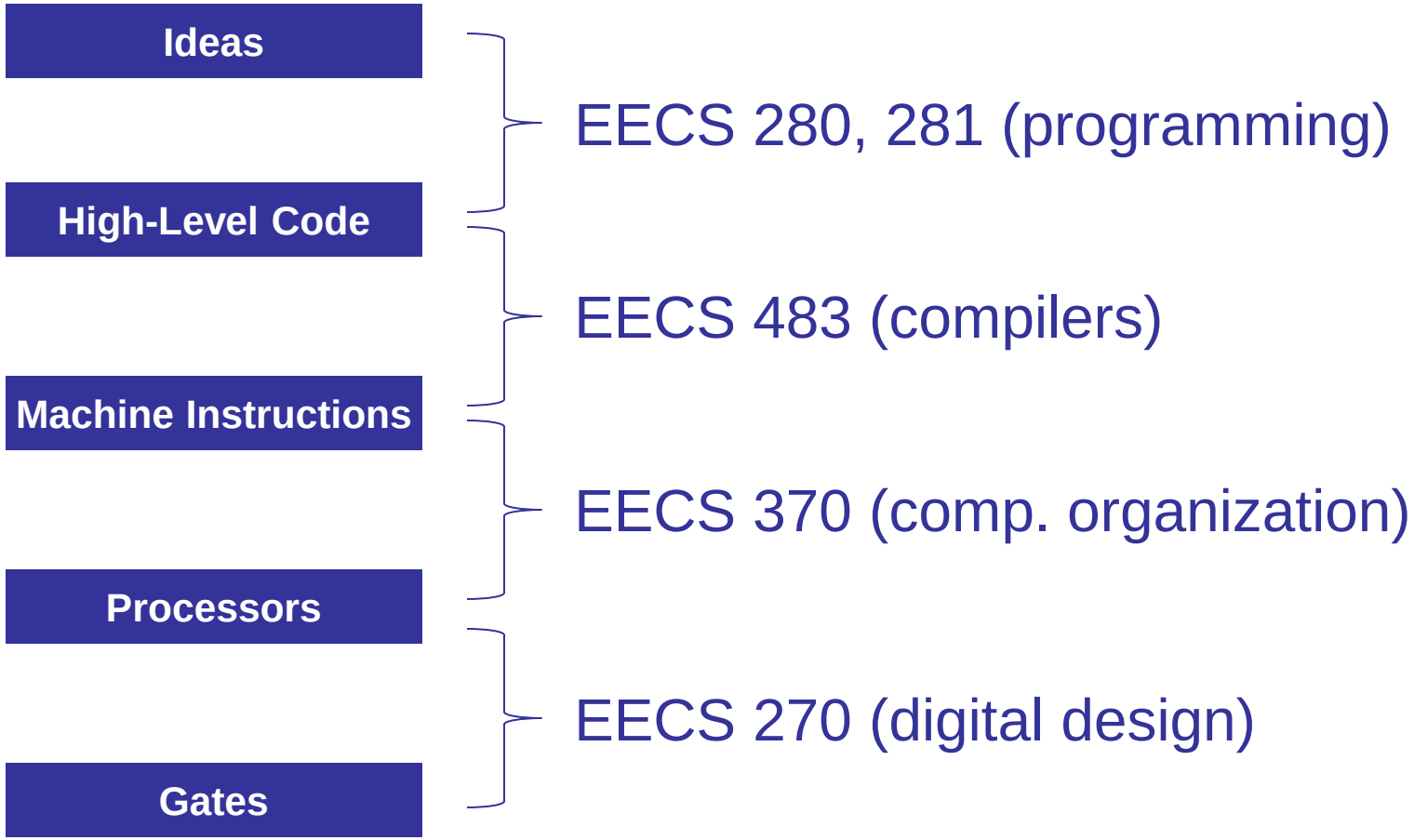
Why do we need 482?

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



What is missing?

Bootstrap:

- How does a computer start when you turn it on?
- How to get CPU to start executing a program?

Concurrent execution with I/O:

- How to read keyboard or mouse? Print output to screen?
- How to run multiple programs without breaking each 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?

How

Concu

How n?

How ther?

Persis

How to save your data when you turn the computer off?

How to prevent other users from accessing your data?

The OS does all of this.

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

Class Material

Class webpage

https://grader2.eecs.umich.edu/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

Lectures are being recorded.

Attendance is not required.

Lecture slides will be posted.

Textbook (highly recommended):

Anderson and Dahlin, "*Operating Systems: Principles and Practice*"

Lab

Fridays 11:30 am to 12:30 pm EDT

Streamed and recorded, details to follow.

Questions to be discussed are posted.

- Do them **before** going to your section
- Prepares you for exams

First lab is this Friday.

Enrollments

Obviously not full, only one lecture and one lab, so there should not be problems with waitlists.

Talk to me if you are retaking this class.

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 May 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!

No 6-credit option this semester

W20 had an optional (experimental) 2-credit EECS 498 that could be taken at the same time.

Differences were mostly some advanced functionality in the projects.

Due to much smaller enrollment over the summer, you can do the advanced features if you like but it won’t get you the extra 2 credits.

Core vs. advanced features

	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

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
- Number of hours and number of lines of code don't count
- Testing is integral process of development

Make good use of help available

- Office hours get tight near project deadlines
- 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 8:00 pm EDT (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: June 24, 2020

Final: August 20, 2020

Exams will be conducted online using the 280 randomized exam server.

Open long enough to accommodate students spread across many timezones.

Grading breakdown

We will be using normal letter grading.

Expect a curve similar to past semesters.

Projects:

- Project 1: 3%
- Projects 2, 3, and 4: 15% each

Mid-term and Final: 26% each

Pro tips for success in 482

1. Start early on projects
2. Pick group wisely

   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 lab sections

   Read textbook, solve questions before discussion

5. Ask questions when something is unclear

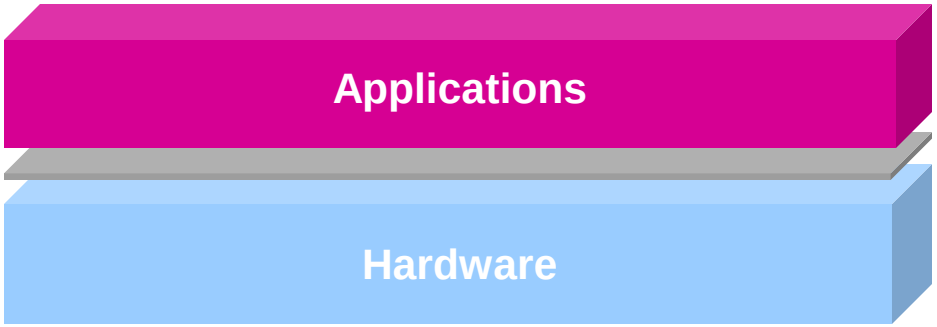
Submit your photo

Have someone take your photo on your phone

https://grader2.eecs.umich.edu/eecs482/self.php

Why have an OS?

What if applications ran directly on hardware?

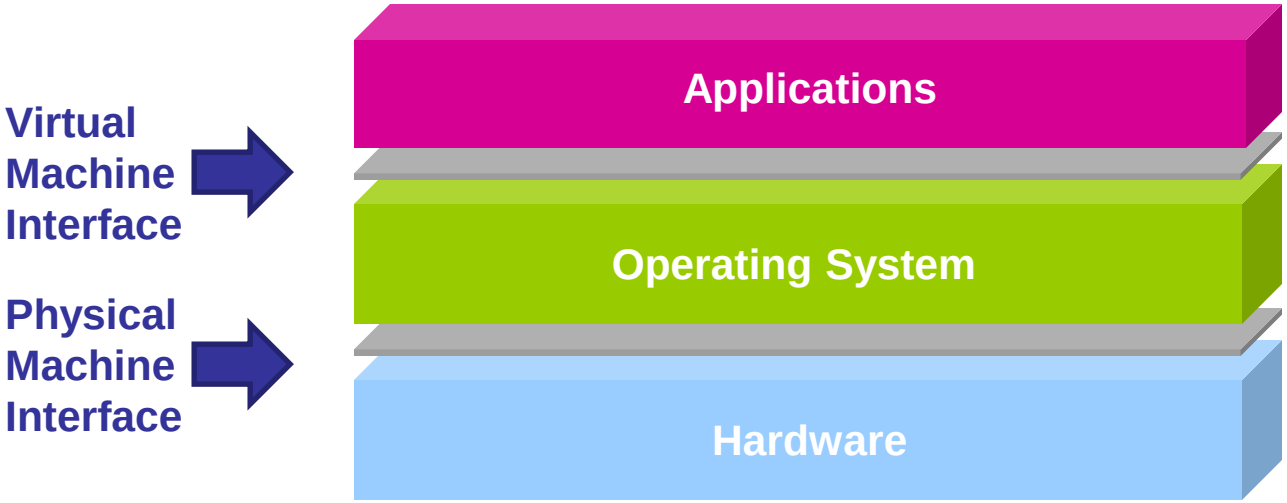


Problems?

- 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: Perspective 1

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 occur outside any program?
  - Example: Receiving network packets
- how do multiple programs run simultaneously without messing each other up?

OS and Apps: Perspective 2

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?

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 apps

Mastering concurrency

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

Operating Systems Take 1

Single operator at console

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

time

Positives:

- Interactive
- Very simple

Downside:

- Poor utilization of hardware

Operating Systems Take 2

Batch processing

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

OS is batch monitor + library of standard services

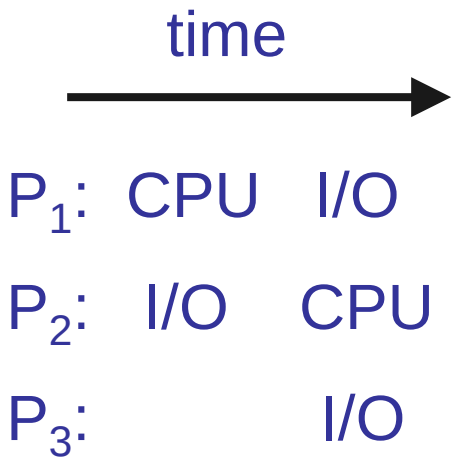
Protection becomes an issue

Why wasn't this an issue for single operator at console?

Operating Systems Take 3

Multi-programmed batch

- Improve utilization further by overlapping CPU and 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

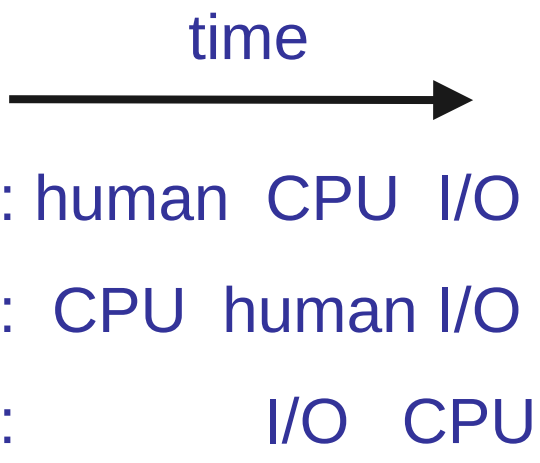
Operating Systems Take 4

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



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 still need to time share between multiple jobs.
- PCs still 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

Things to do …

Browse the course web page

Subscribe to Piazza

Register GitHub ID

Submit photo

Start finding partners for project group

Go to lab section on Friday