EECS 482
Introduction to Operating Systems

Winter 2019

Manos Kapritsos
manosk@umich.edu

Thanks to Harsha Madhyastha and Peter Chen for the slides and notes
About Me

Manos Kapritsos (manosk@umich.edu)
Area of research: Distributed Systems
About you and me

- I love teaching and interacting with my students

- I want to get to know you all by name
  - Send me a picture of you
    - My email is: manosk@umich.edu
    - Subject: [EECS482] Picture of <first name (preferred)> <lastname>
    - State your first name when I call on you

- I’m here to help. Come to me with any question!
  - course-related: office hours
  - Life, The Universe, and Everything: any time
Your guardian angels

Our wonderful IAs and GSIs!
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

Ideen

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?

- **I/O and concurrency:**
  - 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?

The OS handles all these issues!

You should be able to answer all these questions by the end of the class.
How will you benefit from this class?

- Understand what you use
  - Understanding how an OS works helps you develop apps
  - System functionality, performance, efficiency, etc.

- Pervasive abstractions
  - Concurrency, caching, indirection, naming, atomicity, authentication, protection
  - Examples: Cloud computing, web services

- Complex software systems
  - Many of you will go on to work on large software projects
Objectives of this class

1. We will study **design principles** of an OS
   - This course is not about specifics of any particular OS
   - Popular OSes have very similar structure

2. Develop an understanding of **OS impact on application** performance and reliability

3. Examples:
   - What causes your code to crash when you access NULL?
   - Why can multi-threaded code be slower than single-threaded code?
Class Material

- Class webpage
  - [http://web.eecs.umich.edu/~manosk/eecs482/](http://web.eecs.umich.edu/~manosk/eecs482/)

- Syllabus, reading material, homeworks, and projects will be posted on class webpage

- Subscribe yourself to Piazza
  - Announcements and class discussion
Lectures Schedule

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

- Lecture videos and slides will be posted on course web page

- Textbook readings:
  - Anderson and Dahlin, "Operating Systems: Principles and Practice"

- Read material before lectures
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
- You can attend any section, as long as there are seats
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

- Submit your github username via course web page
- Declare your group (by 1/30) via course web page
- Post to Piazza if you don’t know anyone
- We’ll assign private github repositories
The projects are hard!
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

- The projects are not long
  - 100-1500 lines each
  - but they introduce new concepts
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
  - 34 hours of office hours per week (68 hours in weeks projects due)
  - Monitor and participate in discussion on Piazza
  - Attend discussion sections on Fridays
Project recommendations

- Choose group members carefully

- We’ll evaluate every member’s contributions
  - Peer feedback
  - git log and github statistics

- Group can fire one of its members
  - Who then has to find a new group willing to let them in
Policies

- **Submission**
  - 1 submission per day to autograder + 3 bonus
  - Due at midnight on deadline
  - 3 late days across all projects

- **Collaboration**
  - Okay to clarify problem or discuss C++ syntax
  - Not okay to discuss solutions
Exams

- Midterm: March 1 (10am-12pm) (tentative)
- Final: April 29 (7-9pm)

- No makeup exams
  - Except in dire circumstances
  - Make sure you schedule your interviews/travel appropriately
Grading breakdown

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

- Mid-term and Final: 26% each
Enrollment

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

- Talk to me if you are retaking this class

- Waitlist: we’ll keep adding students from the waitlist, as long as there are seats open
EECS498-002

- **Two options:**
  - EECS482 only (4 credits)
  - EECS482+EECS498 (4+2 credits)

- Enrollment in 498: normal add/drop rules

- Both options attend same lectures and labs

- **Differences:**
  - 6-credit students may have extra exam questions (with extra time)
  - 6-credit students will have to implement advanced features for each project
# 4-v-6: project differences

<table>
<thead>
<tr>
<th></th>
<th>core project</th>
<th>advanced features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project 1</strong></td>
<td>67 LOC</td>
<td>67 (+0) LOC</td>
</tr>
<tr>
<td><strong>Project 2</strong></td>
<td>1 CPU 221 LOC</td>
<td>&gt;1 CPU 266 (+45) LOC</td>
</tr>
<tr>
<td><strong>Project 3</strong></td>
<td>create process with empty arena 283 LOC</td>
<td>create process with copied-on-write arena 305 (+22) LOC</td>
</tr>
<tr>
<td><strong>Project 4</strong></td>
<td>delete empty directories only 570 LOC</td>
<td>delete non-empty directories space-efficient locking 584 (+14) LOC</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td>1142 LOC</td>
<td>1223 (+81) LOC</td>
</tr>
</tbody>
</table>
Project groups

Having both 4-credit and 6-credit students in a group is allowed, but **strongly** discouraged

- 6-credit students may have to rewrite their project after it passes all core test cases
- 4-credit students may find their 6-credit teammates using up their group’s late days
Grading

- Autograder will evaluate submissions on both core and advanced features
  - Score for advanced features won’t count for 4-credit students
- Test cases
  - Projects 2-4 require you to submit a test suite
  - All groups need to write test suites that exercise all projects features (core and advanced)
- Final grade: separate grades for 482 and 498
Recipe for success in 482

Ingredients: a student

1. **Start early on projects**
2. **Take advantage of available help**
   - Go to off hours, post/monitor questions on Piazza
3. **Attend lectures and labs**
   - Read textbook material before class
   - Solve posted questions before lab
4. **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.
What is an OS?

For any area of OS, ask:
- What interface does hardware present (physical reality)?
- What interface does OS present to applications?
OS and Hardware

- Creates **abstractions** to make hardware easier to use
  - CPU → Threads
  - Memory → Address space
  - Persistent storage → File systems

- Manages **shared** hardware resources
  - Side-effect: Tax on resources
OS and Applications

Perspective 1: application is main program; it gets services by calling kernel (OS)
- Example: rely on OS to modify registers

Problems with this view:
- how does application program start?
- how do tasks get done that occur outside of any program (e.g. receiving network packets)?
- how to run multiple programs simultaneously without messing each other up?
Perspective 2: **OS is main program**; calls applications as subroutines

- Offer illusion that every process is running on its own computer

Lower layer invokes the higher layer!
History of operating systems

- Single operator at console
  - Interactive
  - Very simple
    » One thing happening at a time
    » OS is library of standard services

- Positives:
- Downside:
  - Poor utilization of hardware
History of operating systems

- 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?
History of operating systems

- 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

*Time*

- $P_1$: CPU, I/O
- $P_2$: I/O, CPU
- $P_3$: I/O
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

```
\begin{tikzpicture}
    \node (P1) at (0,0) {\text{P}_1: \text{human \hspace{1cm} CPU \hspace{1cm} I/O}};
    \node (P2) at (3,0) {\text{P}_2: \text{CPU \hspace{1cm} human \hspace{1cm} I/O}};
    \node (P3) at (6,0) {\text{P}_3: \text{I/O \hspace{1cm} CPU}};
    \draw[->] (P1) -- (P2) node[midway,above] {time};
    \draw[->] (P2) -- (P3) node[midway,above] {time};
\end{tikzpicture}
```
History of operating systems

- OS started out very simple, then became more advanced to use expensive 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?

- Personal computing OSes have gradually added back all features from time-sharing systems.
Questions to Ponder

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

What is part of an OS? What is not?
  - Is the windowing system part of an OS?
Things to do ...

- Browse the course web page
- Subscribe to discussion forum on Piazza
- Register your github account on the course webpage
- Start finding partners for project group
- Send me a picture of you!
  - My email: manosk@umich.edu
  - Subject: [EECS482] Picture of <first name (preferred)> <lastname>