EECS 482
Introduction to Operating Systems

Fall 2017

Manos Kapritsos
manosk@umich.edu

Slides by: Harsha V. Madhyastha
About Me

Manos Kapritsos (manosk@umich.edu)
Area of research: Distributed Systems
Manos  /ˈmænəʊz/  ■  noun

1. An amalgamation of the words ‘Man’ and the acronym ‘OS’ (Operating Systems), Manos is used to refer to a person with great wisdom and a deep knowledge of Operating Systems.

2. A person who occasionally invents fictional dictionary lemmas about their name.
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> <UMID>
  - 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?
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 concurrent execution:**
- 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?
  - How to get a program into memory and have the CPU start executing it?

- **I/O and concurrent execution:**
  - How to read from the keyboard or mouse? Print output to the 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

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

- Develop an understanding of OS impact on application performance and reliability

- 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/~pmchen/eecs482/](http://web.eecs.umich.edu/~pmchen/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:

- 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 9/21) via course web page
  - Post to Piazza if you don’t know anyone
  - We’ll assign private github repositories
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
- 20 hours of office hours per week (48 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: October 24th (6:30-8:30pm)
- Final: December 14th (8-10am)
- No makeup exams
  - Unless dire circumstances
  - Make sure you schedule your 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 section-specific questions

- A few volunteers to move to section 2?

- Talk to me if you are retaking this class
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 discussions
   Read textbook material before class
   Solve posted questions before discussion
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 $\rightarrow$ Threads
  - Memory $\rightarrow$ Address space
  - Persistent storage $\rightarrow$ 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?
OS and Applications

- 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
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
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> <UMID>