**Class Material**

- Class webpage
  - [http://web.eecs.umich.edu/~pmchen/eecs482/](http://web.eecs.umich.edu/~pmchen/eecs482/)
  - Also linked from CTools
- Syllabus, handouts, 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 security and case studies

**Lectures**

- Lecture videos will be posted online
  - But not slides
- Lecture handouts on course web page
  - Bring print outs to class
- Textbook readings:

**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

**Lecture Logistics**

- Raise your hand to answer a question
- State your name first if called out to answer

**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
  - Declare your group (by 9/22) 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

- Reason for being hard: Not number of lines of code, but 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
  - 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 its members
  - Who then has to find a new group willing to let him/her in

Policies

- Submission
  - 1 submission per day to autograder + 3 bonus
  - Due at 6pm on deadline, but okay until midnight
  - 3 late days across all projects

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

Exams

- Midterm: October 29th (6-7:30pm)
- Final: December 17th (8-10am)

- No makeup exams
  - Unless dire circumstances

Grading breakdown

- Projects:
  - Project 1: 4%
  - Project 2: 16%
  - Project 3: 12%
  - Project 4: 18%
- Mid-term (22%)
- Final (28%)
Recipe for success in 482

- Start early on projects
- Take advantage of available help
  - Attend discussion sections and office hours
  - Post and monitor questions on Piazza
- Make sure to attend lectures
  - Read textbook material before class
- Ask questions when something is unclear

Questions for today

- Why do we need an OS course?
- Why do we need an OS and what does it do?
- How did OSes evolve to what we have today?

Relevant EECS Curriculum

- EECS 281
  - High-level ideas → Programs
  - Coding skills learned in 281 critical for projects, especially pointer handling
- EECS 370
  - Assembly language → Hardware execution
  - Registers (SP, PC, ...), page tables, TLB
- EECS 483 (Compilers)
  - High-level language → Assembly language

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?

Why an OS 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 how OS impacts 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?
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

- The OS is “all the code that you didn’t have to write” to implement your application

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

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
- time
  - human I/O
  - CPU
  - I/O
- human

- Positives:
  - Interactive
  - Very simple
    - One thing happening at a time
    - OS is library of standard services
- Downside:
  - Poor utilization of hardware
History of operating systems

- Batch processing
  - Goal: improve CPU and I/O utilization by removing user interaction
  
  ![Diagram](time I/O CPU I/O CPU I/O CPU)

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

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 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

![Diagram](time P1: human CPU I/O P2: CPU human I/O P3: I/O CPU)

- OS is now even more complicated
  - Lots of simultaneous jobs
  - Multiple sources of new jobs

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
- Start finding partners for project group