Course Overview

CS 202: Advanced Operating Systems

Instructor Self-Intro

- Graduated from UNC-Chapel Hill in 2013
- Assistant/associate professor at UT-Dallas during 2013-2022
- Full-time visiting professor at ETH Zurich during 2020.1-2021.6
- Associate professor at UCR since July 2022!
- Autonomous embedded systems, real-time systems, robotics
- https://intra.ece.ucr.edu/~cong
- Check csrankings.org

Today

- Course organization and mechanics
- Introduction to OS

What is this course about?

- How has the role of the OS evolved over time?
  - How does the past inform the present?
- What are the underlying principles of OS?
- What are current and future trends in OS?
- Make it real: lab assignments to get some experience with OS development

- *Get you ready to do Systems research*

Some topics we will cover

- OS design models and how they evolved
  - Monolithic kernels, micro-kernels, …
  - extensibility, protection, performance

Some topics we will cover

- OS design models and how they evolved
  - Monolithic kernels, micro-kernels, …
  - extensibility, protection, performance
- Concurrency:
  - Scheduling (including provable scheduler design for timing-critical systems)
  - Synchronization
  - Multicore OS

Some topics we will cover

- OS design models and how they evolved
  - Monolithic kernels, micro-kernels, …
  - extensibility, protection, performance
- Concurrency:
  - Scheduling (including provable scheduler design for timing-critical systems)
  - Synchronization
  - Multicore OS
- File systems:
  - Sequential, networked, distributed, internet scale

Pre-requisites

- CS 202 is a tier-1 course
  - Lectures will recap basics of OS
  - But you are expected to read textbook chapters for background

- To do well, you must understand all lecture materials and read required materials
- Architecture, networks, distributed systems courses are also a plus

Class format

- For every topic:
  - Some overview
  - Read related book chapters
  - Discuss research papers
- Research papers:
  - Critiques for some required papers
  - You are responsible for the papers and materials discussed in class
- Lecture format: 45 min (first half) + 5 min (break) + 30 min (2nd half)

Questions while reading papers

- What are the primary goals (hypothesis)?
  - 2 sentence elevator pitch
- Pros and Cons
- Why did the authors do what they did the way they did?
  - Understand motivation for design
- What were the driving forces for the paper at the time it was written?
- What parts still hold? What parts don't?
- How did they experiment to support their ideas?
- **Write the review by yourself, do NOT leverage online tools such as chatgpt!**

Reading Research Papers

- Guidelines for reading papers
  - Make sure to identify authors' goals and assumptions. Not always directly stated.
  - Look for high-level takeaways.
  - **Simulate the whole process in your head**
  - Follow a multi-pass reading strategy
    - Pass1: Get overview
    - Pass2: Read details and make notes
    - Pass3: Re-read details to evaluate.
  - Think how techniques used are applicable today. Identify extensions.

Projects

- 3 Lab Assignments
  - Modifications to xv6 (https://github.com/mit-pdos/xv6-riscv)
  - Often not directly connected to the research topics discussed
    - Primary goal is to improve systems building skills
- All labs will be done in groups (3 members recommended)
  - Form your group as soon as possible
  - Group with fewer members acceptable, but the scope remains the same
- Submission
  - Short report & code per group on Canvas
  - Brief summaries of individual contributions
  - Short demo
- Late policy: 10% penalty per day; up to 2 late days permitted

Course Materials

- Textbook: OS, 3 easy pieces: http://pages.cs.wisc.edu/~remzi/OSTEP/
  - Its free!
  - Its excellent!
- Lecture Slides/Schedule/ProjectInfo
  - https://intra.ece.ucr.edu/~cong/teaching/UCR/AOS/F23.html
- Published research papers (course website)

Grading Criteria

- Class participation and paper critiques: 20 %
- Lab assignments: 40 %
- Midterm: 20 %
- Final: 20 %

- Midterm:
  - Closed book with 3 cheat sheet (A4 size, both sides), 80 minutes
- Final exam:
  - Closed book with 3 cheat sheet (A4 size, both sides)
  - Final is non-cumulative
  - 3 hours
- A set of CEP questions will be asked in both Midterm and Final exams

Questions and Feedback

- Grades will be available online at elearn/Canvas
  - https://elearn.ucr.edu/
  - TA will stop by our next lecture and talk about labs, paper critique submission, grades etc.
- You can provide feedback anytime

Cheating and Plagiarism

- Your code and critiques may be compared against others!
- UCR cheating policies
  - Please read and understand: https://conduct.ucr.edu/policies/academic-integrity-policies-and-procedures
- What consequence can happen?
  - Grade penalty
  - Course failure and department notification
  - Academic sanctions: probation, suspension, dismissal..

Situation

- We all have multiple applications running on our smart phones and computers
  - Written by programmers that don't know each other
  - They all just magically work – how?

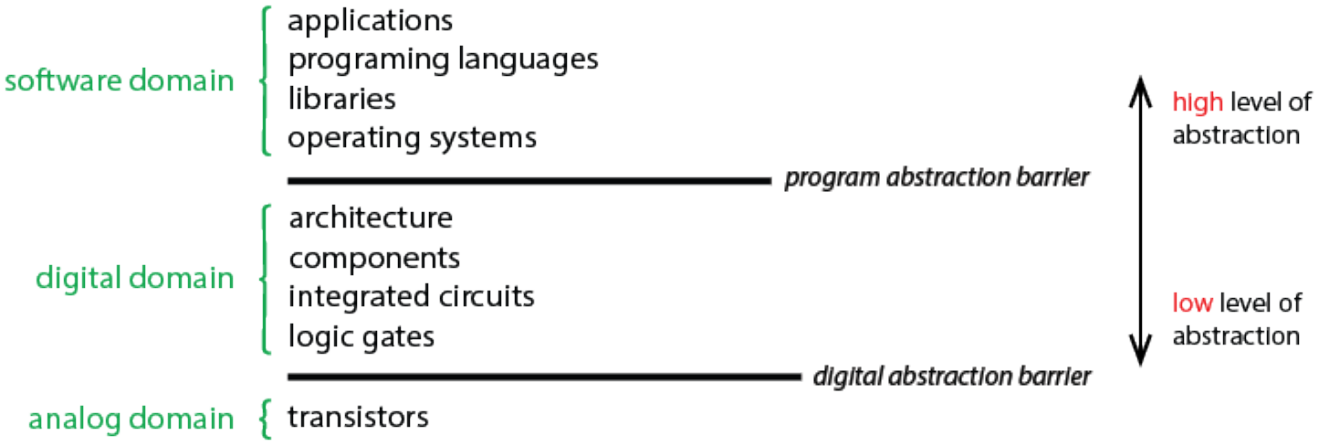
- Goal today: get you ready to discuss OS structure, our first topic

Computing Systems – a hierarchy of abstractions

- Computing systems are a series of abstractions
  - Impossible to think about a system from electrons to application in one shot
  - What are some abstraction layers we have from transistors to applications?

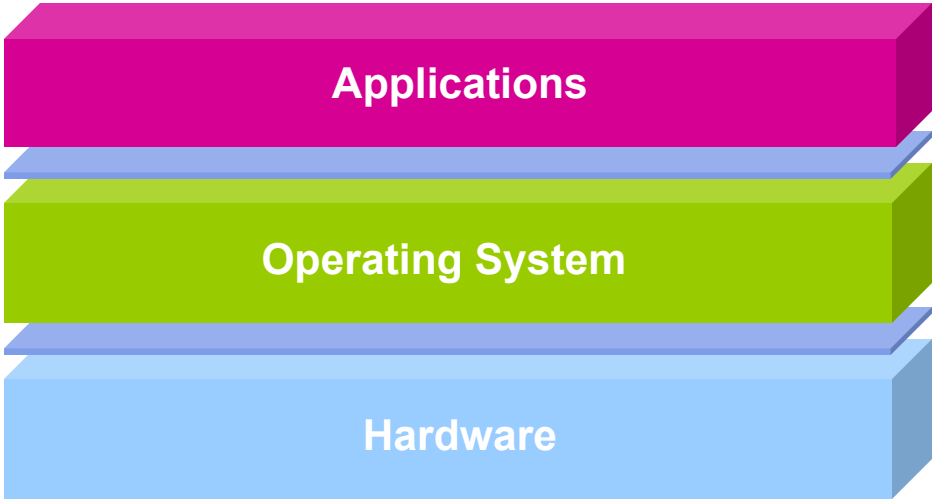
Computing Systems – a hierarchy of abstractions

- Computing systems are a series of abstractions
  - Impossible to think about a system from electrons to application in one shot
  - What are some abstraction layers we have from transistors to applications?



- This class: OS level abstractions

What is an OS?



- Has direct access to underlying hardware
- Hides hardware complexity
  - Offers nice abstractions to the applications through system calls
- Manages the hardware on behalf of one or more applications
- Ensures that applications are isolated and protected from each other

Getting more technical

- What is an OS?
  - A piece of software that *abstracts* and *arbitrates* a computing system

Getting more technical

- What is an OS?
  - A piece of software that *abstracts* and *arbitrates* a computing system
- A manager in a shop
  - Directs resources
    - Controls CPUs, memory, devices…
  - Enforces working policies
    - Fairness, resource limits, …
  - Simplifies complex tasks
    - Abstracts hardware; offers system calls

OS Purposes

- Abstraction of hardware
- Multiplexing
- Isolation
- Sharing
- Security
- Performance
- A range of uses

Abstraction and Arbitration

- OS offers **abstractions** and **arbitration**

- Example of arbitration
  - Allocate memory or CPU time
  - Arbitrate access to a shared device

- Examples of abstractions
  - Files, sockets, process, thread, …

Abstractions, mechanisms, policies

- Memory management example
- *Abstraction*: memory page
- *Mechanisms*: allocate, deallocate
- *Policies*: page replacement, LRU, LFU, …

Design principles

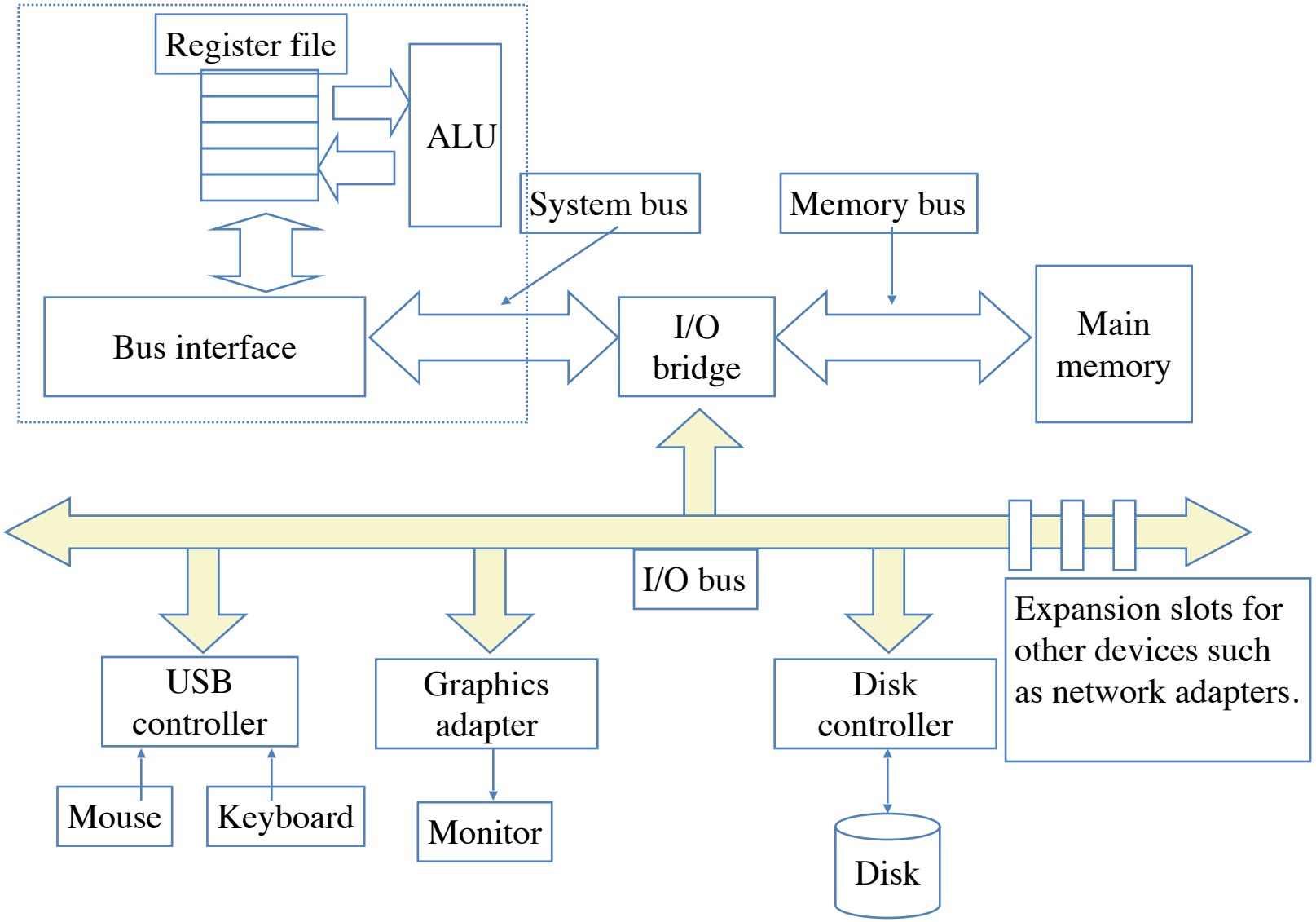
- Separation of **mechanism** and **policy**
  - Implement flexible mechanisms to support many policies
  - Develop better policies based on the same machanism

- Policies must optimize for use cases
  - Where will the OS be used?
  - What will the user want to execute?

Hardware and Resources

- Good understanding of the hardware is essential to understanding OS
- What hardware?
  - Smart phone/tablets?
  - Desktops?
  - Servers?
  - Computing clusters?
  - Cloud?
- How different are these?

They are not that different!



How does the OS interact with the hardware?

- OS
  - Has protected access to hardware resources
  - Arbitrates among competing requests
  - Receives and handles events from the hardware

What support does the hardware provide?

- Manipulating privileged machine state
  - Protected instructions
  - Manipulate device registers, TLB entries, etc.
  - Controlling access
- Generating and handling “events”
  - Interrupts, exceptions, system calls, etc.
  - Respond to external events
  - CPU requires software intervention to handle fault or trap
- Mechanisms to handle concurrency
  - Interrupts, atomic instructions

Catering to Applications

Catering to Applications

- Provide resource needs of an application
  - CPU, memory, device access

Catering to Applications

- Provide resource needs of an application
  - CPU, memory, device access
- When applications launch, the OS loads the program from file into memory
  - Allocates memory for code, data, heap and stack
  - Can the application ask for more resources?
    - Yes, OS receives additional requests and provides resources as needed

Catering to Applications

- Provide resource needs of an application
  - CPU, memory, device access
- When applications launch, the OS loads the program from file into memory
  - Allocates memory for code, data, heap and stack
  - Can the application ask for more resources?
    - Yes, OS receives additional requests and provides resources as needed
- OS also reacts to events in the system

CPU management

- Abstractions
  - Program: static entity
  - Process: program in execution
    - Unit of resource allocation and one thread of execution

Memory management

- Abstractions:
  - Address space for each process
  - Pages to manage memory

- OS implements these abstractions using the available hardware support
  - Paging, segmentation, TLBs, caches…

Storage/file system

- Abstraction: files and directories
  - Others possible: e.g., object store
- Implemented in a variety of different ways
  - Traditional file system: mapping of files to storage
  - Network file system
  - Distributed FS
  - Internet scale FS

Some thoughts on why OS is hard/interesting

- Unforgiving
- Tensions
  - Efficient – Abstraction
  - Powerful – Simple API
  - Flexible - Secure
  - Performance - consistency/determinism/predictability/ robustness

4-way intersection demo (coordinating via an RSU)

ROS2-based Autoware.auto

XronOS-based Autoware.auto

4-way intersection demo (coordinating via an RSU)

ROS2-based Autoware.auto

XronOS-based Autoware.auto

4-way intersection demo (coordinating via an RSU)

ROS2-based Autoware.auto

XronOS-based Autoware.auto

4-

J)

Relinquishing predictability in middleware is unacceptable for safety-critical AS

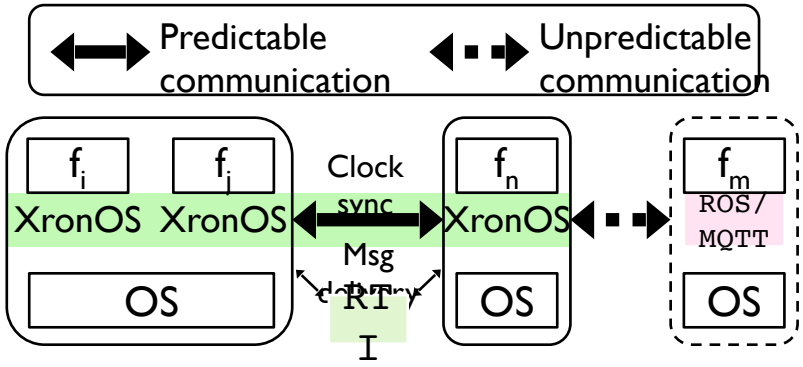
XronOS: An event-driven middleware that provides predictable coordination without sacrificing performance

ROS2-based Autoware.auto

XronOS-based Autoware.auto

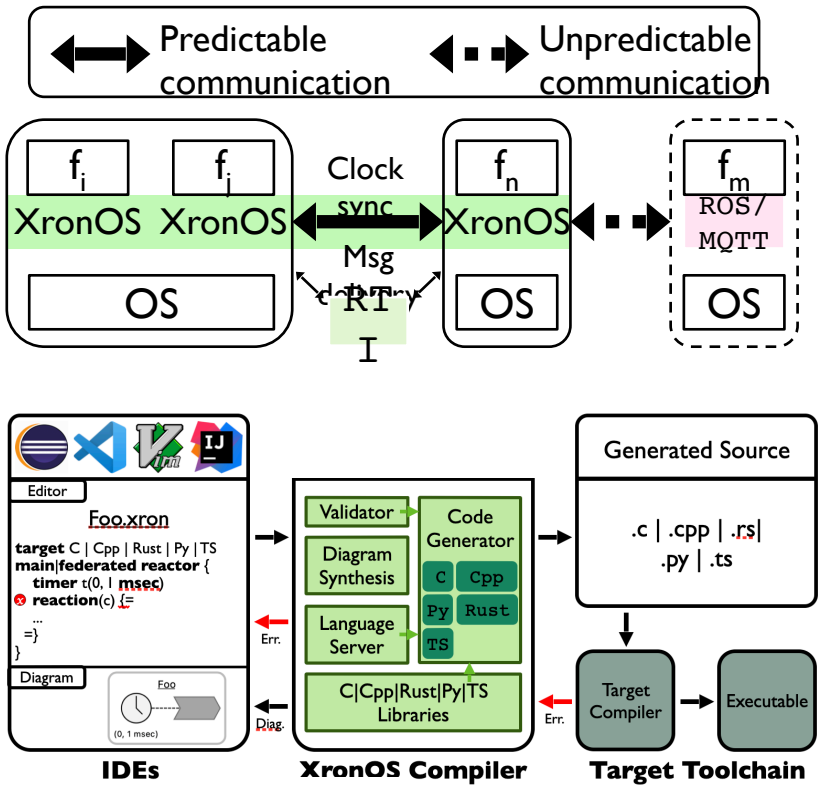
Design goals of XronOS

- **Enable predictable coordination via explicit temporal semantics based on logical time**
  - Reflect multiplicity of distinct logical & physical timelines
    - Predictable coordination via the exposure of the relationships between events across timelines
  - Allow the re-use of existing ROS and MQTT libraries and nodes
  - Trading off availability v.s. consistency in light of Brewer's CAP theorem
    - Centralized coordination (consistency over availability)
    - De-centralized coordination (availability over consistency)



Design goals of XronOS

- **Enable predictable coordination via explicit temporal semantics based on logical time**
  - Reflect multiplicity of distinct logical & physical timelines
    - Predictable coordination via the exposure of the relationships between events across timelines
  - Allow the re-use of existing ROS and MQTT libraries and nodes
  - Trading off availability v.s. consistency in light of Brewer's CAP theorem
    - Centralized coordination (consistency over availability)
    - De-centralized coordination (availability over consistency)



Systems research is all about optimizing tradeoffs and tensions

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

  - **Performance**

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

  - **Performance -> Distributing**

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

  - **Performance -> Distributing -> Faults**

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

  - **Performance -> Distributing -> Faults -> Fault Tolerance**

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

- **Performance -> Distributing -> Faults -> Fault Tolerance -> Replicas <-**

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

- **Performance -> Distributing -> Faults -> Fault Tolerance**
  Inconsistency <- Replicas <-

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

- **Performance -> Distributing -> Faults -> Fault Tolerance**

Consistency <- Inconsistency <- Replicas <-

Systems research is all about optimizing tradeoffs and tensions

- Think about creating a scalable, fault-tolerant, and high performance distributed file system that could meet the demands of FAAMG's rapidly growing infrastructure

- **Performance -> Distributing -> Faults -> Fault Tolerance**

Low <- Consistency <- Inconsistency <- Replicas <- performance

Why this class could be challenging

- From the instructor's perspective
  - The "comprehensive" way of teaching AOS yet in a QUARTER system!
  - Tier 1 course: need to consider students who have not taken undergraduate OS classes
- From students' perspective
  - Digesting the comprehensiveness in ~9-10 weeks
  - Some (parts) of the lectures could be boring for those who have taken OS classes before, yet could be rather challenging for students who have not taken any OS classes
  - However, in order to get a good enough grade (e.g., A- and above), you have to get a solid, true understanding of all materials, particularly the research papers covered

Conclusions

- Today was a quick overview of the role of an OS
- Check course schedule and *reading* materials
- Form a group for lab assignments