CSC 453

Operating Systems

This class

Who am I?

- https://pschmitt.net
- Academics
  - → PhD at UCSB
  - → Researcher (summers) at ICSI
  - → Postdoc then research faculty at Princeton
  - → Some other stops… USC, Stanford, UH
- Privacy startup co-founder & CEO, measurement startup COO
- Main focuses
  - → Integrating privacy and security into systems that we all use
  - → ML for network management
  - → Connectivity / privacy in challenging environments


UNHCR

This class

- Too much to cover in one term: survey of important topics
- Course Canvas: https://canvas.calpoly.edu/courses/168889
- Syllabus: https://schmittpaul.github.io/CSC453W26/syllabus.pdf
- Waitlists…
- What OSes do you all run?

Who are you?

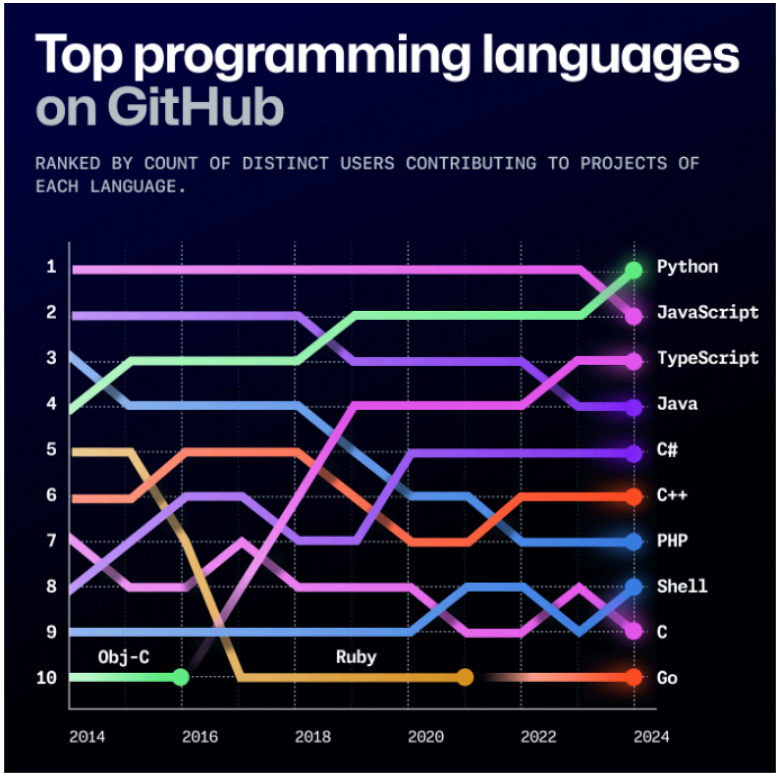
- Name / Major / Year
- Where do you identify as your hometown?

How to succeed in this class

- Turn things in on time
- Attend class
  - → … on time. Lectures will start at 10 after the hour
- Ask questions
  - → I'm very flexible about how much we cover this semester
  - → I would rather teach less and have everyone understand it
  - → Our back-and-forth during class is the one of the few indicators I have of how much you are absorbing
- Talk to me if you are struggling

What are we doing here?

- How many of you have participated in OS development?
- How many of you regularly program in languages that use operating system abstractions directly?
- And C is a decreasingly popular language!
- So why study operating systems? Why is this class even offered? Why is it required?



Why take this class

- You are required to do so in order to graduate
- Reality: this is how computers really work, and as a computer scientist or engineer you should know how computers really work
- Ubiquity: operating systems are everywhere and you are likely to eventually encounter them or their limitations
- Beauty: operating systems are examples of mature solutions to difficult design and engineering problems. Studying them will improve your ability to design and implement abstractions

Course progression

- Introduction to operating system abstractions and structures
- Abstracting and multiplexing:
  - → CPU: interrupts, context, threads, processes, processor scheduling, thread synchronization, deadlocks
  - → Memory: memory layout, address translation, paging and segmentation, address spaces, translation caching, page fault handling, page eviction, swapping
  - → File systems and storage: disk scheduling, on-disk layout, files, buffer cache, crash and recovery
  - → Security (time permitting)

Background

Questions to consider

- What is an operating system?
- What do they do?
- How did OSes evolve?
- How do the requirements differ for different OSes?

What is an operating system?

OS history - how did we get here?

- Started out as libraries to provide common functionality across programs
  - → See any issues with this approach?
- Later, evolved from procedure call to system call
  - → What's the difference?
- Evolved from running a single program to multiple processes concurrently
  - → New issues to solve?

Types of operating systems

- Desktop
- Time share/Mainframe
- Mobile
- Web
- Real-time
- Embedded
- Virtual Machines

TI;dr

OSes continue to evolve as environments (i.e., constraints) change

What isn’t clear?

Comments? Thoughts?

OS abstractions

Questions to consider

- What are the main abstractions OSes provide?
- What are the abstraction challenges?

Abstractions

- Abstractions *simplify application design* by:
  - → hiding undesirable properties,
  - → adding new capabilities, and
  - → organizing information
- Abstractions provide an interface to application programmers that separates policy—what the interface commits to accomplishing—from mechanism—how the interface is implemented.

What are the abstractions?

- CPUs
  - → Processes, threads
- Memory
  - → Address space
- Disk
  - → Files

Example OS abstraction: file systems

- What undesirable properties do file systems hide?
  - → Disks are slow!
  - → Chunks of storage are actually distributed all over the disk
  - → Disk storage may fail!
- What new capabilities do files add?
  - → Growth and shrinking
  - → Organization into directories, searchability
- What information do files help organize?
  - → Ownership and permissions
  - → Access time, modification time, type, etc.

Abstraction tradeoffs - discussion

- Identify undesirable properties hidden by, new capabilities added, and info organization provided with these abstractions:
  - → Process / threads
  - → Address space

Abstraction pros / cons

- Advantages of OS providing abstractions?
  - → Allow applications to reuse common facilities
  - → Make different devices look the same
  - → Provide higher-level or more useful functionality
- Challenges?
  - → What are the correct abstractions?
  - → How much should be exposed?

OS design requirements - what do we need?

- Convenience, abstraction of hardware resources for user programs
- Efficiency of usage of CPU, memory, etc.
- Isolation between multiple processes
- Reliability, the OS must not fail
- Other:
  - → Security
  - → Mobility

What isn’t clear?

Comments? Thoughts?

Resource management

Questions to consider

- How does the OS manage access to resources?

OS as a resource manager

- Another view: resource manager - shares resources “well”
- Advantages of the OS managing resources?
  - → Protect applications from one another
  - → Provide efficient access to resources (cost, time, energy)
  - → Provide fair access to resources
- Challenges?
  - → What are the correct mechanisms?
  - → What are the correct policies?

Resources are managed via services

- Program Execution (loading, running, monitoring, terminating)
- Performance (optimizing resources under constraints)
- Correctness (overseeing critical operations, preventing interference)
- Fairness (access to and allocation of resources)
- Error detection & recovery (network partition & media failure)

Services (cont'd)

- Communication (inter-process, software-to-hardware, hardware-to-hardware, system-to-system, wide-area)
- I/O: reading & writing, support for various mediums, devices, performance, and protections
- Data Organization (naming), Services (search) & Protection (access control)
- Security (isolation, enforcement, services, authentication, accounting and logging, trust)
- User interfaces (command-line, GUIs, multiple users)

Each service has challenges and tensions

Example 1: We have limited RAM, and we want to run more programs that can be stored.

- How do we allocate space?
- Who stays?
- Who goes?
- What if we're wrong?
- What if the system is under extremely heavy load?
- Is there a way to predict the future?

Each service has challenges and tensions

Example 2: We have two process (producer / consumer); how do they communicate?

- Message passing? Shared memory?
- How do they synchronize?
  - → How to we prevent over-production? Over-consumption?
  - → Context-switching?

What isn’t clear?

Comments? Thoughts?