OPERATING SYSTEMS

CS124 – Operating Systems

Spring 2024, Lecture 1

Welcome!

- Detailed exploration of operating system implementation
- Hard prerequisite: CS24
- This is a project class:
  - Every assignment requires substantial programming effort
  - Most programming is C; a small amount is IA32 assembly
  - Use Git version control for managing your code, making checkins, etc.
  - Also, Make/Doxygen tools.
- No exams; grade is taken entirely from assignments
- Course uses the Pintos instructional operating system
  - A small UNIX-like operating system with very limited capabilities
  - Implemented in 2005 for use with Stanford's CS140 OS class
  - Intended to be run on IA32/x86 processor emulator (Bochs, QEMU)
    - Can also run on actual IA32 hardware if properly coaxed…

Assignments

- Five assignments to complete throughout the term:
  - Write a basic operating system shell (1 week)
  - Kernel-level threading and thread-scheduling (2 weeks)
  - Implement kernel system calls for user-mode programs (2 weeks)
  - Implement a virtual memory system for Pintos (2 weeks)
  - Implement an ext2-like filesystem for Pintos (2 weeks)
- The last assignment is due at the end of seniors' finals week
- Assignments are weighted by how many weeks they take
  - Two-week assignments are worth twice the one-week assignment

Assignments and Collaboration

- The assignments are hard
  - Lots of code to understand, significant implementation effort, and lots of debugging to do
- You are required to work in groups of 2-3 students
  - Not allowed to tackle this course individually
- Biggest reason: you will have other people to talk with, when designing and debugging systems
- Students can drop the class, but this will affect others…
  - Please only take this course if you really intend to finish it!
- If students drop later in the term, we can adjust the teams
  - e.g. move a student into another team (student will have to learn the new team's code)

Assignments and Collaboration (2)

- We will be using GitHub Classroom to manage code repositories, and to facilitate collaboration
- See course Canvas page for link to join the Classroom

- Two-step submission process for each assignment:
  - Push your completed work to your team's GitHub repository
  - One teammate submits the commit-hash and other details on Canvas

Assignments and Collaboration (3)

- **Each team's submission must be created entirely by that team alone. Teams cannot share implementation code. Teams cannot use AI coding tools.**
- Cross-team sharing is encouraged in these areas:
  - Design and implementation ideas (but not code or pseudocode!)
  - Pitfalls you encountered, and how to solve them
  - Help with setup and debugging
- Also, Pintos has been around since 2005…
  - **Do not look for solutions to projects online!**
- You are encouraged to look at other resources, e.g. Linux sources, other textbooks, OS dev. websites, etc.
  - **Don't copy code!** (see first point above) Focus on understanding it.
  - Cite any external sources in your submission, so I can share them with the class this year and next year.

Assignments and Due-Dates

- Each assignment specifies a due-date (Thursdays 5pm)
- Late submissions are penalized as follows:
  - 1 day late = 10% deduction
  - 2 days late = 10 + 20 = 30% deduction
  - 3 days late = 10 + 20 + 30 = 60% deduction
  - After 4 days, don’t bother ☹
- Each team has 6 “late tokens”
  - Each token is good for 24 hours of extension, No Questions Asked.
  - In your submitted design doc, note how many tokens you are using
- Students/teams can also request extensions due to health or other reasons
  - Most important thing is to try to do this <u>beforehand</u>, if possible

Development and Testing Platform

- Pintos is designed to be built and tested on 32-bit Linux
- This has become difficult for multiple reasons
  - Who runs a 32-bit OS anymore?
  - Apple has moved away from Intel x86 processors, to an ARM-based platform

- We have multiple possible solutions
- For Intel x86-based platforms:
  - We have a VirtualBox image of 32-bit Mint Linux for you to use
- We also have Docker images for Intel- and ARM-based platforms
- There are also a few other options in the works

One more note…

- This course is significantly UNIX focused…
    - Linux, macOS, PintOS, …
- By "UNIX" we mean UNIX and its many variants
    - SysV, BSD and variants, Linux, macOS, …
    - Sometimes indicated as *NIX

- Concepts appear across <u>all</u> major operating systems
- UNIX is just the easiest one to experiment with

- We will point out major themes of other operating systems, but all your work will be on UNIX-style systems

Operating Systems

- **What is an operating system?**
- Most generally:
  - An operating system provides applications with a **standardized interface** to the computer's hardware resources.
  - An operating system **manages the allocation and sharing of hardware resources** to applications that want to use them.
- Many different variations under this theme!
  - How the operating system is architected
  - What kinds of devices the OS runs on
  - What facilities/services/guarantees the OS provides to applications
- We'll start with the general principles first…

Example: Filesystems

- Many kinds of storage media used in a typical computer
- Hard disks with varying interfaces:
  - Serial ATA (SATA) hard disks
  - SCSI (Small Computer System Interface) or SAS (Serial Attached SCSI) disks
  - On-motherboard SSDs with M.2 SATA or PCIe interfaces
  - USB storage devices that can be added and removed at runtime
- Different size HDDs must be accessed in different ways
  - Old disks used Cylinder-Head-Sector (CHS) addressing, but this imposed limitations on supported disk sizes
  - (Plus, modern disks have multiple zones, each with its own geometry: outer zones can fit more sectors around the disk)
  - Later disks introduced Logical Block Addressing (LBA) which supports much larger disks

Example: Filesystems (2)

- Different storage technologies require different kinds of maintenance
- Magnetic disks are sensitive to fragmentation
  - Large files should be stored in contiguous regions of the disk, or disk-seek times will kill access performance
- SSDs (Solid-State Drives) have a constant seek time; they don't care about fragmentation. But:
  - SSD memory blocks must be erased before they can be rewritten, and the erase-block size is much larger than read/write block size
  - Blocks can only be erased so many times before they wear out
  - To minimize performance and wear issues, the filesystem must interact with SSDs differently than with magnetic disks

Example: Filesystems (3)

- Storage devices may also have many different formats!
- Hard disk drives and solid-state drives:
  - NTFS (Windows)
  - HPFS (older macOS)
  - APFS (newer macOS)
  - ext4, btrfs, and many others (Linux)
- Removable flash storage:
  - FAT32
  - exFAT
- Optical devices:
  - ISO9660 (older CD format)
  - UDF (newer CD format)

Filesystems: Standardized Interface (1)

- UNIX operating systems provide a simple mechanism for interacting with storage devices in the computer:
  - **`open()`** Opens a file for manipulation
  - **`close()`** Closes a file
  - **`read()`** Read a block of one or more bytes from a file
  - **`write()`** Write a block of one or more bytes to a file
  - etc.
- A Virtual File System (VFS) presents a single unified view of all disks and files in the computer
  - Root of the virtual filesystem is "**`/`**"
  - Storage devices are mounted at specific paths, e.g. "**`/mnt/cdrom`**"
  - Every file can be accessed by a path from the root of the filesystem

Filesystems: Standardized Interface (2)

- UNIX operating systems provide a simple mechanism for interacting with storage devices in the computer:
  - **`open()`** Opens a file for manipulation
  - **`close()`** Closes a file
  - **`read()`** Read a block of one or more bytes from a file
  - **`write()`** Write a block of one or more bytes to a file
  - etc.
- In fact, other devices use *essentially the same interface!*
  - Console input and output (**`printf`** / **`scanf`** use **`read`** / **`write`**)
  - Socket communications
  - Pipes between processes
- Only real API difference: how to open each device

Filesystems: Resource Sharing

- In UNIX, multiple processes can manipulate the same file
- Scenario:
  - Process A opens file **`foo.txt`** to read and write it.
  - Later, process B deletes **`foo.txt`**, while A is still using it.
    - (UNIX file deletion is performed using the **`unlink()`** system call)
  - **What should happen?**
- Hardware resources are shared by multiple processes…
- The operating system must coordinate access to these shared resources in a well-defined manner
  - e.g. to maintain system security, correctness, performance, etc.
- In UNIX:
  - When process B deletes **`foo.txt`**, the OS removes the path to the file. But, the actual file still remains until process A terminates!
  - After process A terminates, OS reclaims space used by **`foo.txt`**

Operating Systems: A Brief History

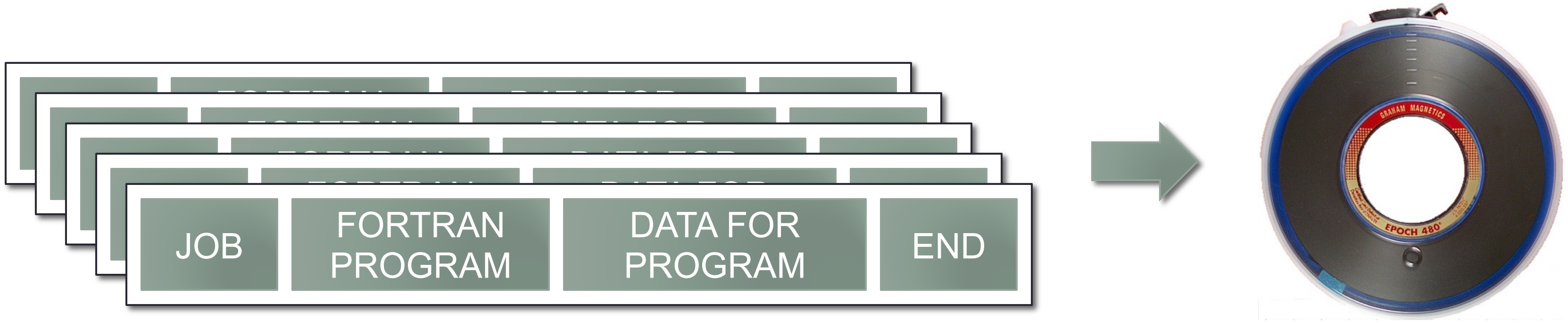
- Early general-purpose computers were **mainframes**
  - Programmers would create jobs from a series of punch cards

- A job would be fed into mainframe by a human operator…
  - …the mainframe does its thing…
  - …then the results are printed out for the programmer to use.

- A lot of time was wasted waiting for programs to be loaded, results to be printed, etc.
  - The mainframe's CPU is sitting idle, blocked on I/O operations

Operating Systems: A Brief History (2)

- Later mainframes used **batch processing**
  - A simpler, lower cost computer transfers multiple jobs onto a single input tape



- The mainframe reads and executes each job in sequence
  - Instead of printing, job output is saved to an output tape
  - Also, system tapes hold common programs like the FORTRAN compiler
- Program output is printed by a simpler, cheaper computer
- Benefit: greatly reduces wasted time!

Operating Systems: A Brief History (3)

- A big problem with batch-processing systems:

  - If job 1 is waiting for I/O to complete (e.g. on tape), the mainframe can't do anything else! The CPU sits idle until I/O completes.
  - This became increasingly common as computer use broadened
- Later-generation mainframes introduced support for **multiprogramming**
  - If one job is blocked on I/O or some other operation, switch execution to another job
  - If mainframe can keep several programs in memory, and switch between them, the CPU can be kept busy most of the time

Operating Systems: A Brief History (4)

- To support multiprogramming, mainframe memory was partitioned into regions for each job

Input Tape:

- New problem to solve:
  - Need to prevent different jobs from accessing each other's memory regions
  - Must provide **process isolation**
  - Requires hardware support to implement effectively
  - Requires **multiple CPU operating modes**, so the OS is the only program able to manipulate the memory partitioning

Mainframe Memory:

Operating Systems: A Brief History (5)

- Another problem with batch-processing mainframes:
  - If a programmer had a bug in their program, *they didn't know* until their job had been batched up, processed, and the results printed
  - Could take *hours* to even discover you had a syntax error in your code! ☹
- **Timesharing** systems were mainframes that provided users with online terminals
  - Timesharing is an extension of multiprogramming, allowing users to issue jobs directly on the mainframe, and receive their own output
  - First appearance of basic **multitasking** in an operating system
- The mainframe was still large and expensive…
  - An individual user won't keep the CPU utilized at 100%...
  - A group of many users will keep CPU much more heavily utilized

Operating Systems: A Brief History (6)

- Integrated circuit technology became widespread, and processors became cheaper and cheaper…
- Instead of an entire university sharing a single computer, each department could have their own computer
  - **Minicomputers** were smaller and less powerful than mainframes
- As hardware prices continued to drop, became feasible to give individual users their own **microcomputers**
- Up to this point, operating systems and programs primarily used text interfaces for user interaction…
  - **Graphical User Interfaces** (GUIs) were developed to make it easy for people to use computers, even if a user had no intention of learning how the computer worked

Operating Systems: A Brief History (7)

- As processors became less expensive, became common to have multiple processors in a single computer
- **Multiprocessor** systems contain multiple processors in separate packages
- **Multicore** systems have multiple processors in a single package
- Multiprocessor/multicore systems require specific support from the operating system
  - Coordinating access to shared data-structures within the operating system becomes much trickier
  - Process scheduling also takes multiprocessor systems into account to maximize cache utilization

Operating Systems: A Brief History (8)

- Modern computers can even run an operating system as an application within another operating system
  - The **host operating system** runs the **guest operating system** as an application
- **Emulation:**
  - A computer with one CPU type simulates another CPU [usually] of a different type, allowing applications or even a guest operating system to be run within the host system
- **Virtualization:**
  - A computer with one CPU type runs a guest operating system compiled for the same CPU type
  - If the CPU has hardware virtualization support, this will be fast!
  - Otherwise, certain CPU features must be emulated by the host OS when running the guest operating system

Operating Systems: A Brief History (9)

- The software that provides a virtual machine for the guest OS is called a **hypervisor**
- Handles many concerns similar to more traditional operating systems
  - Enforce isolation between guest operating systems
  - Management of hardware resources shared between guest OSes
- A few new challenges:
  - Guest OSes expect to access hardware directly; hypervisor must present this abstraction to guest OSes
    - (either emulated, or via hardware support on the host processor)
    - Can the guest OS tell that it is running within a virtual machine?
  - Guest OSes have their own scheduling and caching strategies; host OS should interfere with these as little as possible

Kinds of Operating Systems

- Operating systems are used in many different contexts, for fulfilling many different purposes
- Mainframe and server operating systems must maximize utilization of hardware
  - Operating system doesn't require a graphical user interface
  - Rather, must support very efficient handling of I/O, and possibly scheduling of many processes
- Personal computers must be easy to use, and responsive to user input
  - Maximizing hardware utilization is less important – responding to user interaction is top priority!
  - Much more code is devoted to making the computer easy to use
  - Important to provide a simplified, user-friendly user interface

Kinds of Operating Systems (2)

- Mobile device / tablet OSes have several challenging, often conflicting constraints
- Must be responsive and user-friendly, like PC operating systems
- But, must also try to maximize battery life through careful hardware resource management
- With smartphones, must support download, installation, execution, and uninstallation of wide range of applications
  - But, basic device capabilities (e.g. voice calls, SMS) must also be rock-solid reliable
- Must support intermittent connectivity, especially when programs are using that connectivity

Kinds of Operating Systems (3)

- By far the most common kind of computer now is the **embedded computer**
  - In your microwave oven, your printer, your WiFi router, your DVD player, controlling your car engine, your point-and-shoot camera, …
- Embedded OSes tend to have very limited capabilities
  - Systems tend to support a specific, fixed set of tasks
  - Systems aren't designed to run arbitrary programs on them
- Can still include a variety of basic OS capabilities
  - Basic thread-management and scheduling support
  - Basic memory management capabilities
  - Support for software upgrades
  - Support for peripherals like flash cards, USB drives, networking, …

Kinds of Operating Systems (4)

- **Real-time operating systems** focus on completing tasks by a specific deadline
- Most general-purpose operating systems provide **soft real-time** support, e.g. for media playback
  - Not considered a system failure if the OS misses a deadline from time to time (e.g. your media playback just sounds choppy)
- Some real-time OSes provide **hard real-time** guarantees
  - If the OS misses a deadline, this is considered a fatal error!
- Example: a computer system for running an automobile manufacturing assembly line
  - The OS receives inputs from sensors along the assembly line…
  - If the OS doesn't satisfy guarantees for processing input data and controlling automated machinery, physical damage will occur
  - If OS misses its timing deadlines: Failure! Halt the assembly line!

Next Time

- More details on operating system components and hardware interactions
- Overview of UNIX facilities for user programs