CSE 120
Principles of Operating Systems

Spring 2023

Lecture 14: Storage Devices and File System API

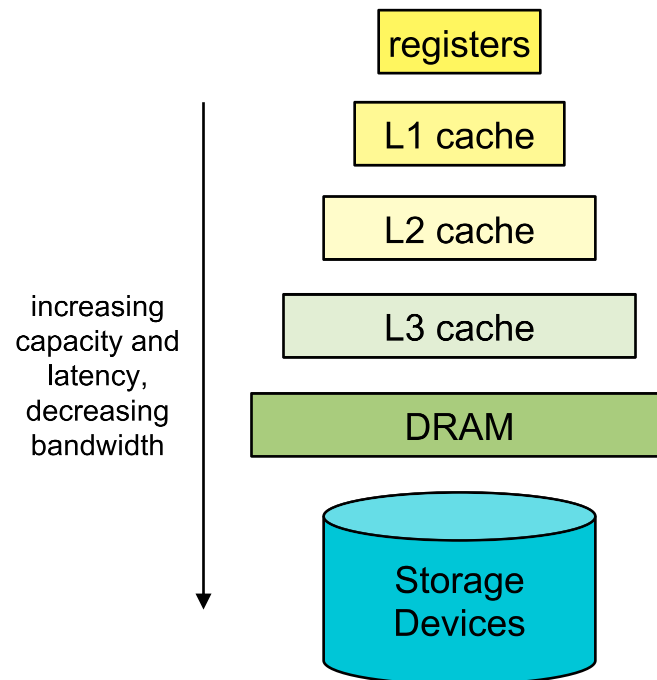
Amy Ousterhout

Administrivia

- Homework #3
  - Due today
- Homework #4
  - Will be released today
- Project 3
  - Deadline to change teams: Thursday, 5/25
  - Due June 10th – there will be no extensions
  - Get started early!

Memory Hierarchy

- Different levels have different:
  - Latency
  - Capacity
  - Bandwidth
  - Cost
- Storage devices are persistent or non-volatile
  - They retain their data when power is turned off
- Storage devices are external to the CPU
  - Managed by device drivers



Storage Devices and File Systems

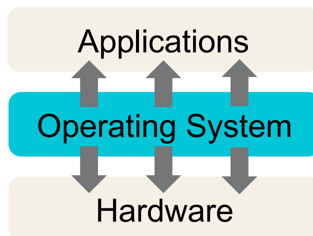
- What are the characteristics of physical storage devices?
  - Structure, performance
- What API should users and programs use to access storage devices?
  - Files, directories, protection
- How can we implement these APIs?
  - File system data structures
  - File buffer cache
  - Reliability

Today's Outline

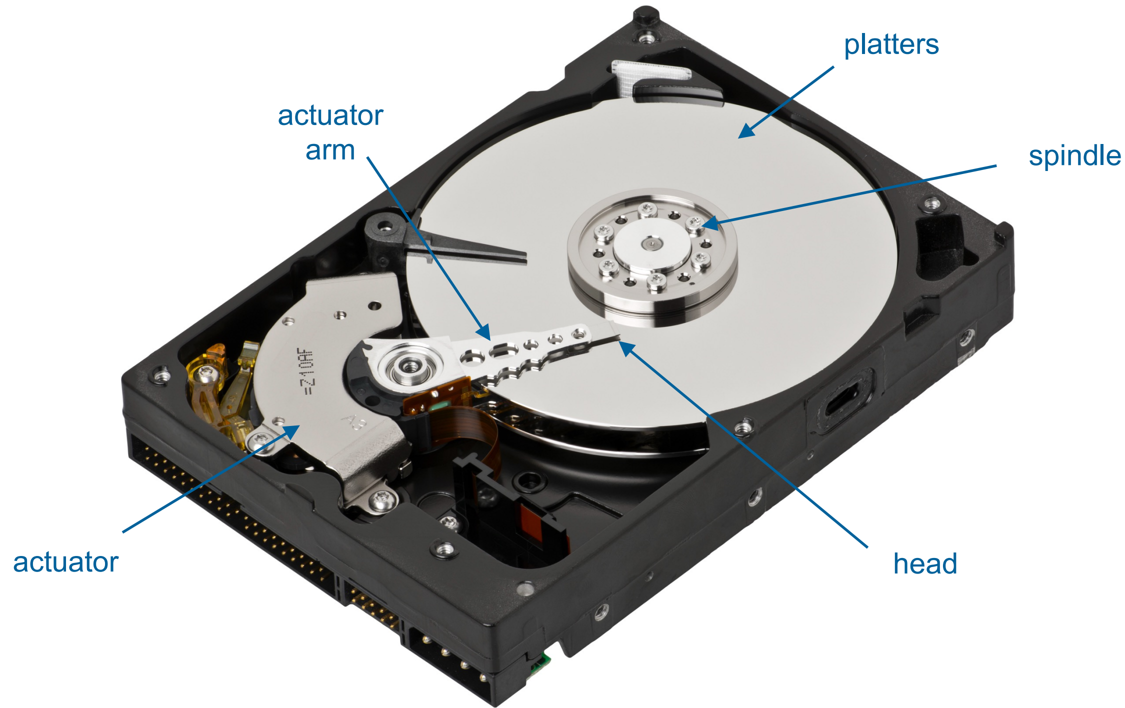
- Physical storage devices
  - Disks, flash, NVM, RAID
  - Performance
- File system APIs for users and programs
  - Files
  - Directories
  - Sharing
  - Protection

Disks and the OS

- Disks are messy physical devices
  - Many physical parts
  - Errors, bad blocks, missed seeks, etc.
- The job of the OS is to hide this mess from higher level software
  - Low-level device control (initiate a disk read, etc.)
  - Higher-level abstractions (files, databases, etc.)

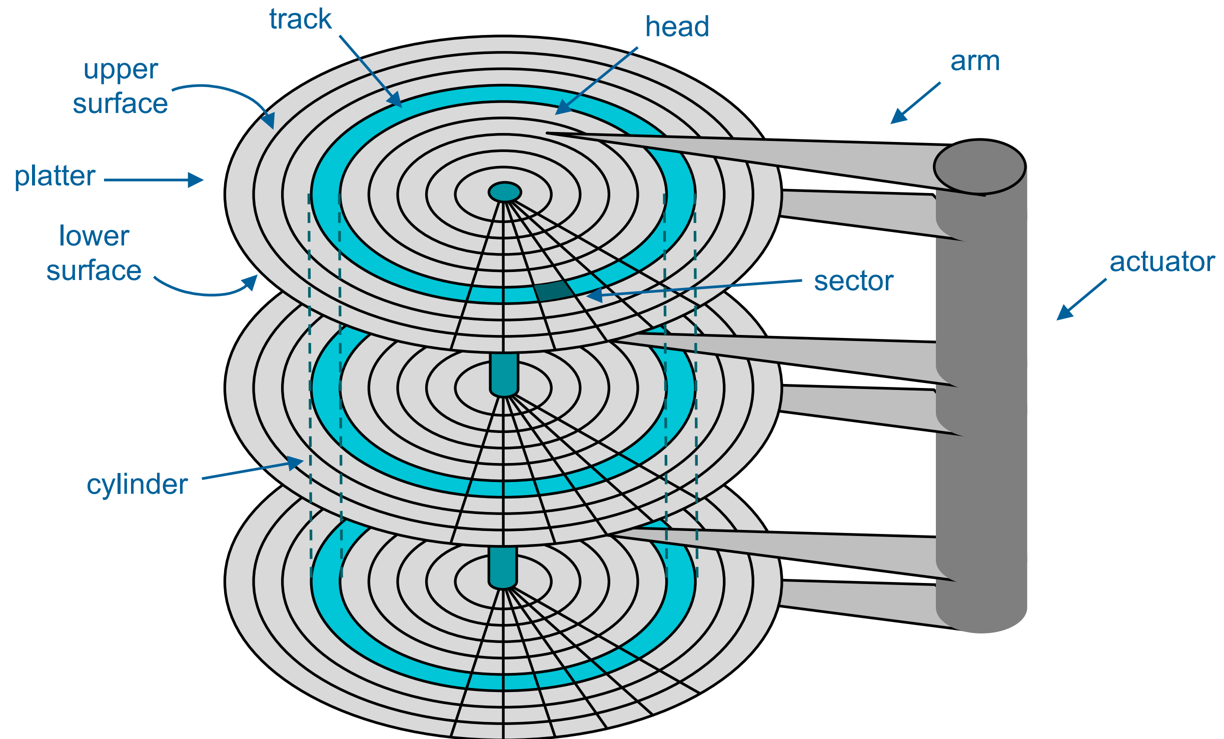


What’s Inside a Hard Disk Drive (HDD)?

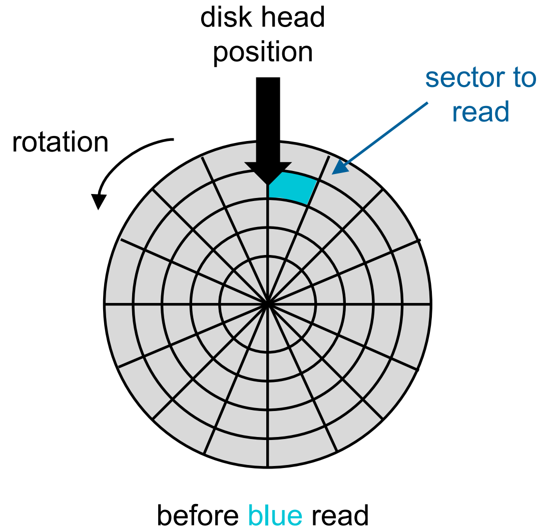


https://commons.wikimedia.org/wiki/File:35-Desktop-Hard-Drive.jpg

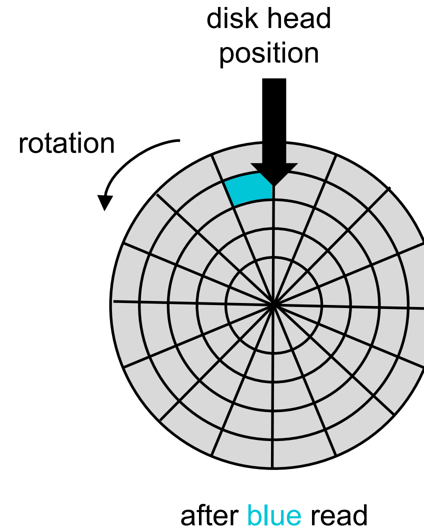
Disk Components



Reading a Disk Sector

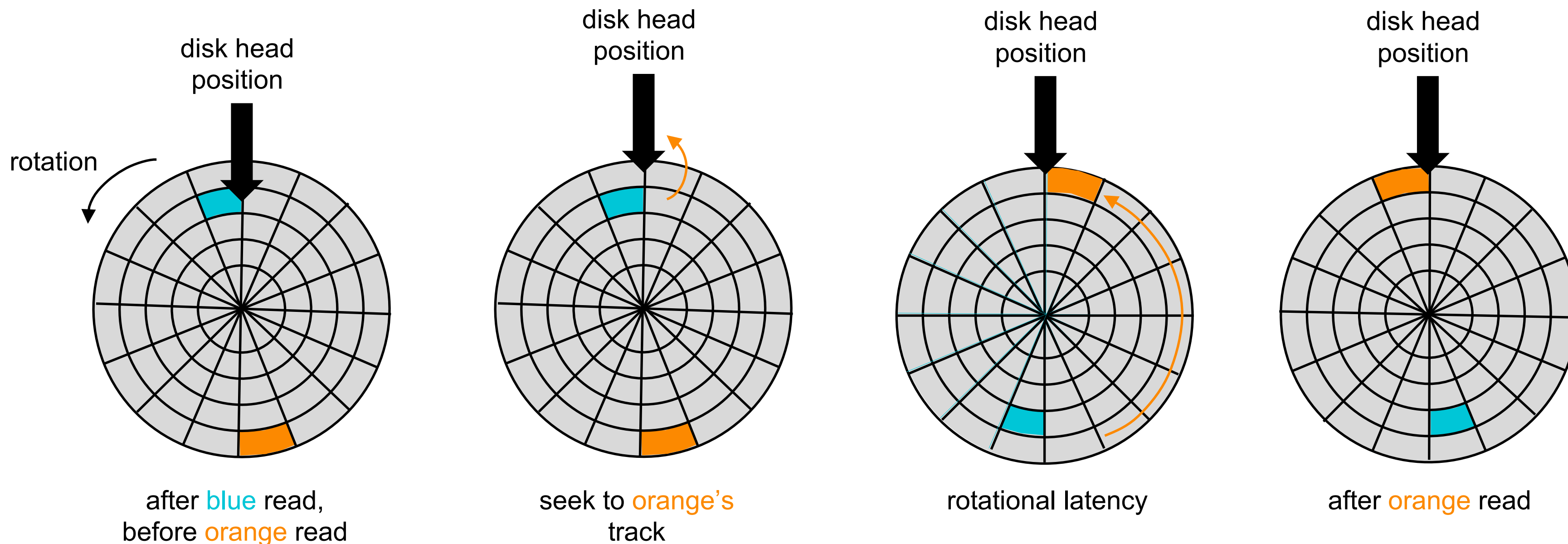


before blue read



after blue read

Reading Another Disk Sector



Disk Performance

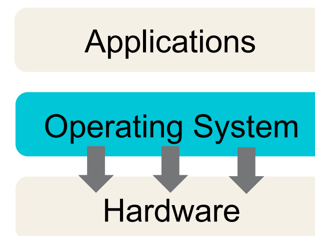
- Performance depends on three steps:
  - Seek – moving the disk arm to the correct cylinder
    - » Slowest part, bound by physical laws
    - » Depends on how fast the arm can move (increasing slowly)
  - Rotation – waiting for the sector to rotate under the head
    - » Depends on rotation rate of disk (increasing slowly)
  - Transfer – transferring data from surface into disk controller electronics, sending it back to host
    - » Depends on density (increasing quickly)
- OS goal: minimize the cost of all these steps

Disk Performance Example

- Disk latency = seek + rotation + transfer
- Time to read 2 512-byte sectors:
  - Seek: 4 ms
  - Rotation: 2 ms
    - » 15000 RPM, half a rotation on average
    - » 1 / 15000 RPM * 60 sec/m * ½ = 2 ms
  - Transfer: 8 μs
    - » Read at 125 MB/s
    - » 2 * 512 B / 125 MB/s = 8 μs
  - Total latency: 4 ms + 2 ms + 8 μs ≈ 6 ms

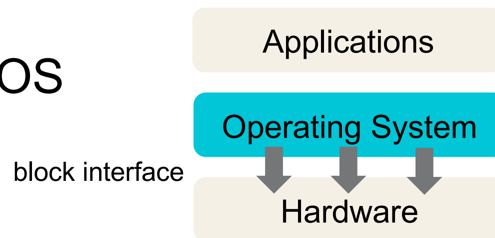
API Between Disks and the OS - Past

- Specifying disk requests requires a lot of info:
  - Surface, track, sector, transfer size…
- Older disks required the OS to specify all of this
- Cons:
  - The OS needed to know all disk parameters
  - Modern disks can be even more complicated



API Between Disks and the OS - Present

- Modern disks provide a higher-level interface
- Disk exports its data as a logical array of blocks [0…N]
  - Disk maps logical blocks to surface/track/sector
  - Called the block interface
- Simpler API, but disk parameters are hidden from the OS



Disk Scheduling Policies

- Seek time and rotational latency are very expensive
- Goal: schedule disk requests to reduce latency
  - First-in first-out (FIFO)
    - Pros: fair, no out-of-order requests
    - Cons: may have long seeks, low throughput
  - Shortest seek time first (SSTF)
    - Pick the closest request on disk, minimize arm movement (seek time)
    - Pros: tries to minimize seek time
    - Cons: starvation, favors middle blocks
  - Elevator (SCAN)
    - Pick the closest request in the direction of travel
    - Pros: bounded time for each request
    - Cons: requests at the other end will take a while

Flash-Based Solid State Disks (SSDs)

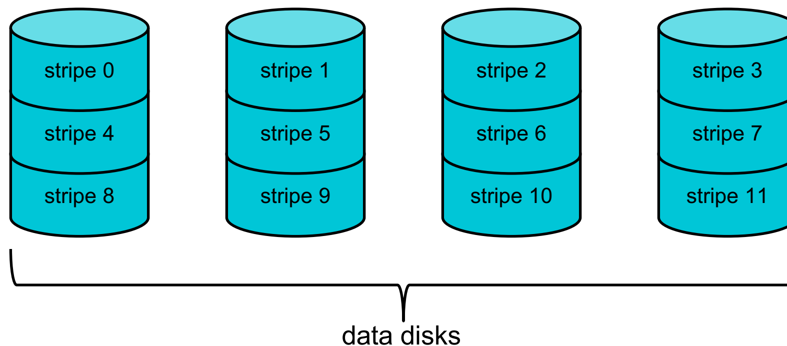
- Solid State Disks (SSDs) are now standard in personal devices
  - Persistent, just like hard disks
- No physical moving parts
  - Faster (100x) and more reliable than hard disks
  - No seek and rotation overhead
- Cost 5-10x more per bit than disk
- Flash suffers from wear-out
  - Once a page has been erased many times it stores data less reliably
- Block interface and file systems typically remain unchanged with SSDs
  - Some optimizations no longer necessary (e.g., layout policies, disk scheduling)
  - New file systems designed for flash and SSDs

Non-Volatile Memory (NVM)

- New technologies that provide non-volatile (persistent) memory
  - Phase change (PCM), spin-torque transfer (STTM), etc.
  - Intel Optane (3D Xpoint) was commercially available
- Performance close to DRAM
  - But persistent!
- Byte addressable
  - Rather than sectors (disk) or pages (SSD)
- Requires both the OS and applications to adapt
- Recently discontinued…

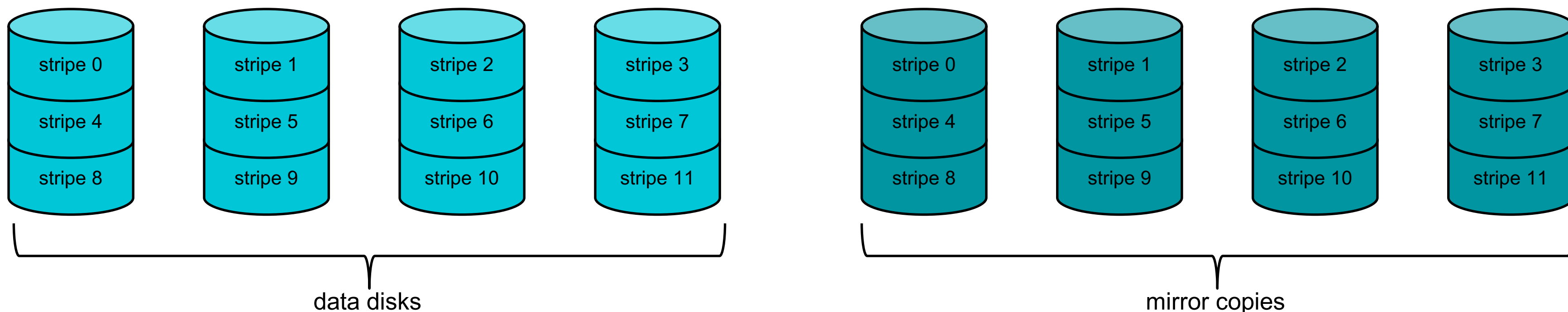
Redundant Array of Inexpensive Disks (RAID)

- Use redundancy to increase throughput and reliability
- Striping
  - Split data blocks across multiple disks
  - Read/write disks in parallel
  - Improves throughput
- Raid level 0



Redundant Array of Inexpensive Disks (RAID)

- Mirroring
  - Store redundant information on a mirror/shadow disks
  - Every write is performance on both disks, but read from either disk
- Raid level 1



Today's Outline

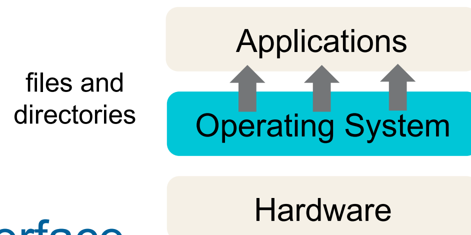
- Physical storage devices
  - Disks, flash, NVM, RAID
  - Performance
- File system APIs for users and programs
  - Files
  - Directories
  - Sharing
  - Protection

File System Components

- Naming
  - How to refer to data with files and directories
- File access
  - Read, write, other operations
- Disk management
  - How to allocate and arrange data on the storage device, map data to blocks
- Protection and permissions
  - Protect data from different users
- Reliability, durability
  - When system crashes, data on storage should be durable

File System Abstractions

- Files - an abstraction for secondary storage
- Directories - a way to organize files logically
- Both enable sharing and protection
- OSes abstract different file systems behind a common interface
  - Interface defines a set of methods and data types
  - Unix: virtual file system (VFS)
  - Windows: installable file system (IFS)
- OS can use any file system that implements this interface
  - Linux: ext3, ext4, xfs, btrfs
  - Windows: FAT16, FAT32, NTFS

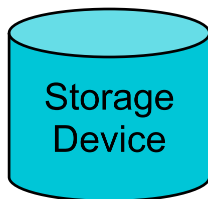


Abstraction: Files

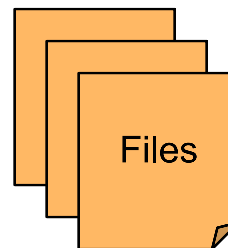
- File: a named collection of bytes stored on durable storage such as disk
- Files properties
  - Size, owner, last modified time, permissions, etc.
- File types
  - Understood by the file system: block, character, link, etc.
  - Understood by other parts of the OS or runtime libraries: text, source, object, executable, application-specific, untyped
  - Can be encoded in the file name
    - .com, .jpg, .exe, .bat, .pptx, .dll
  - Can be encoded in the contents
    - Magic numbers, initial characters (e.g., #! for shell scripts)

File Abstraction vs. Physical Reality

- Physical reality
  - Block oriented
  - Logical block address
  - No protection among users
  - Data might be corrupted if the machine crashes



- File system abstraction
  - Byte oriented
  - Named files
  - Users protected from each other
  - Robust to machine failures



Basic File Operations

Unix

- creat(name)
- open(name, how)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)
- rename(old, new)

Windows

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle, …)
- FlushFileBuffers(handle, …)
- SetFilePointer(handle, …)
- CloseHandle(handle, …)
- DeleteFile(name)
- MoveFile(name)
- CopyFile(name)

File Access Patterns

- Sequential
  - Read bytes one at a time, in order
- Random access
  - Address any byte in the file without accessing earlier bytes first
  - E.g., swap file, databases
- Indexed access
  - File system contains an index to blocks with particular contents
  - E.g., hash tables, dictionaries, databases

Directories

- Directories – a way to organize files logically
- Provide a structured way to organize files
- Convenient naming interface
  - Separate logical file organization from physical file placement on the disk
- File systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, …)
- OSs support the notion of a current directory
  - Relative names are specified with respect to the current directory
  - Absolute names start from the root of the directory tree
  - Maintained on a per-process basis

Directory Internals

- A directory is a list of entries
  - <name, location>
  - Name is just the name of the file or directory
  - Location depends upon how file is represented on disk
- List is usually unordered (effectively random)
  - Entries usually sorted by the program that reads the directory
- Directories stored as files in Unix
  - Only need to manage one kind of storage entity
  - “Everything is a file”
  - Use file ops to create and read directories
  - Some language libraries provide higher-level APIs

(Live Demo of Directories)

Path Name Translation

- Suppose you want to open "/one/two/three"
- What does the file system do?
  - Open directory "/" (well known, can always find it)
  - Search for the entry "one", get location of "one" (in directory entry)
  - Open directory "one", search for "two", get location of "two"
  - Open directory "two", search for "three", get location of "three"
  - Open file "three"
- Spend a lot of time walking directory paths!
  - This is why we separate open from read/write
  - OS can cache prefix lookups for performance

File Sharing

- File sharing has been around since timesharing
  - Easy to do on a single machine
  - Networks give us remote sharing
- File sharing is the basis for communication and synchronization
- Two key issues when sharing files:
  - Semantics of concurrent access
    - » What happens when one process reads while another writes?
    - » What happens when two processes open a file for writing?
    - » What are we going to use to coordinate?
  - Protection

Protection

- File systems implement a protection system
  - Who can access a file
  - How they can access it
- A protection system dictates whether a given action performed by a given subject on a given object should be allowed
  - You can read and/or write your files, but others cannot
  - You can read "/etc/motd" but you cannot write it

UNIX Access Rights

- Mode of access: read, write, execute
- Three classes of users:
  - Owner
  - Group
  - Public
- Ask manager to create a group (unique name) and add some users to the group
- For a particular file or directory, define appropriate access
- For example, try `ls -al` on ieng6:
  - drwxr-x--- 15 cs120sp23b ieng6_cs120sp23b 4096 May 23 14:45 nachos

type
owner
public
group
file owner
file group
permissions

Root and Administrator

- The user “root” is special on Unix
  - Bypasses all protection checks in the kernel
- Administrator is the equivalent on Windows
- Always running as root can be dangerous
  - A mistake or exploit can harm the system
    - “rm” will always remove a file
  - Advice: create a user account on Unix even if you have root access
    - Only run as root when you need to modify the system

Storage Devices and File Systems

- Structure and performance of physical storage devices
  - Disks, flash, NVM, RAID
- File system APIs
  - Files, directories, sharing, protection
- How can we implement these APIs?
  - File system data structures
  - File buffer cache
  - Reliability

For next class…

- Read chapter 40