CS 212 Final Review

Spring 2025

Admin

Final exam 3:30pm-6:30pm next Monday, June 9, in **380-380C** (not Skilling!)

- **Cumulative** — covers entire course
- Open note, but no textbook or electronic devices
- Bring lecture note printouts
- SCPD must register exam monitor or show up in person
  (no need to request permission to show up in person)
- Please remind us if you need OAE arrangements
- Please send us your exam monitor if you are a non-SCPD with permission to take the exam under SCPD rules. (SCPD won't send the exam to your monitor, so we have to do it directly.)
- Please post any questions under **Lectures+Exams** on Edstem!

Topics

Covered in Midterm Review:

- Processes and Threads
- Scheduling
- Concurrency
- Virtual Memory
- Synchronization
- Linking

Today:

- Memory Allocation
- I/O and Disks
- File Systems
- Networking
- Protection and Security
- Virtual Machines

Memory Allocators: Overview

- Must support allocations of arbitrary number/size/lifetime, and can’t move them
- The core fight: minimize fragmentation
  - Different lifetimes + different sizes of allocations leave “holes” in memory
  - No universal solution
- Allocation strategies (e.g. best fit, first fit)
  - Tradeoffs/pathologies based on workload characteristics
- Program allocation behavior: ramps, peaks, and plateaus
  - e.g. arena allocation exploits peak phases

Memory Allocators: Faults & Garbage Collection

- Useful tricks: set permissions to cause fault, and do action in fault handler
  - e.g. sub-page permissions, virtualization, concurrent snapshotting
- Garbage collection
  - Stop & copy
    - Optimization: without the “stop”
      - mutator runs & collector collects, uses fault + resumption
  - Reference counting

I/O & Disks: Device communication

- Memory-mapped device registers
  - regular read/write interface + have special physical addresses correspond to device registers
- Memory-mapped device memory
  - regular read/write interface except access device's internal memory
- Special instructions (e.g. inb, outb)
  - communicates using port numbers
  - x86 special instructions "slow and clunky"
- DMA (direct memory access)
  - CPU offloads read/write of main memory to device/DMA engine
  - Allows overlapping the slow transfer with other computation

I/O & Disks: Device communication

- Polling
  - Keep looping to ask device if something happened yet: action complete, data ready, etc
  - Uses up CPU
- Interrupts
  - Have the device send an interrupt whenever something happens
  - Bad under high arrival rate
  - Best: adaptively switch between polling/interrupts

I/O & Disks: Disks & Disk Scheduling

- Disks
  - (Physical) sectors are the unit of transfer
  - Placement/ordering of requests makes a massive difference
    - Sequential I/O much faster than random
      - Contiguous is best, so organize blocks of files into sequential disk blocks/sectors
    - Long seeks are much slower than short seeks
    - Crashes can lead to inconsistent state (revisit in file systems)
- Scheduling
  - Try to order requests to minimize seek times; maximize concurrency
  - FCFS
    - Fair, but can’t exploit locality, so higher latency + lower throughput
  - SPTF/SSTF
    - Exploits locality, but can lead to starvation (fix: “Aged SPTF”)
  - “Elevator” (SCAN)
    - Sweep across disk, with batches of requests in same direction

I/O & Disks: Flash/Solid State

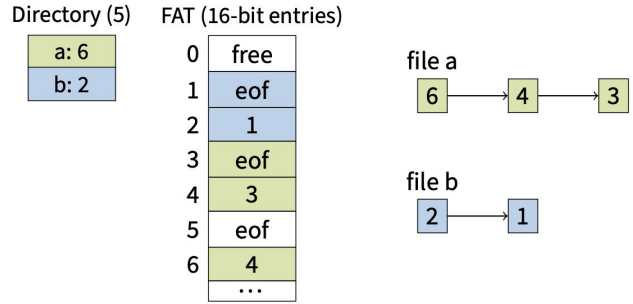
- No more moving parts, but…
- Blocks wear out after some number of overwrites
  - Flash translation layer & wear leveling make random writes very expensive
- Limited durability
  - Too long without power can lead to data loss

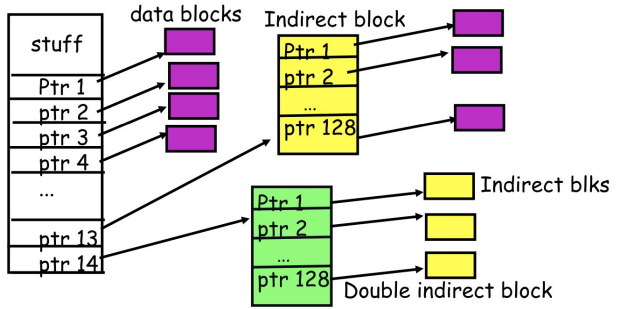
File Systems: Files & Disk

- Filesystem has to translate reads/writes of names/offsets to disk blocks…
  - …while optimizing for disk properties and providing resilience against crashes
- Patterns/observations:
  - All blocks in file tend to be used together, sequentially
  - All files in directory tend to be used together, all names in directory tend to be used together
  - Most files are small, but most of disk consumed by large files (and lots of I/O to large files)
  - Thus, need both good sequential access and good random access

File Systems: Storage Schemes

- “Straw men”
  - Contiguous/extent-based allocation
    - Fast access + simple, but run into external fragmentation
  - Linked files
    - Easy growth + sequential access, but pointer chasing on disk = bad random access
    - FAT stores links separately from data blocks, so pointer chasing is much faster
- Indexed files
  - File has array of pointers to disk blocks, which are allocated on demand
  - Need hierarchical structure to avoid using massive contiguous chunk
  - Multi-level indexed files use “asymmetric” direct and indirect pointers (Project 4!)





File Systems: Directories and Links

- Directories:
  - map names to inodes
  - are also files, just with a special format
  - Hierarchical name space
  - Processes track current working directory
- Hard links: multiple names point to the same inode
  - Can’t create arbitrary hard links to directories
- Soft (symbolic) links: multiple names point to the same original name
  - Can go across file systems

File Systems: Speeding Up Unix FS

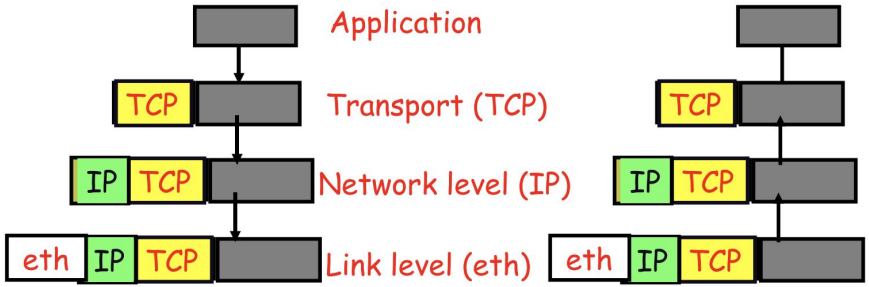
- Problems:
  - Small blocks = too much stuff in inode, too much indirection
  - Poor clustering for consecutive file blocks, inodes & data blocks, inodes in same dir, etc
  - Usability: short filenames, lack of atomic + crash-proof file update
- FFS optimizations:
  - Larger block size
    - To avoid internal fragmentation, allow splitting unused space into smaller “fragments”
  - Cluster related objects
    - Sequential blocks in sequential sectors
    - inodes/data in the same disk cylinder group (same cylinder or at least adjacent cylinders)
    - Bitmap of free blocks rather than free list
  - Do metadata writes asynchronously or with journaling
  - Write to /tmp in memory

File Systems: Crashes & Corruption

- Shutdown can happen at any time
  - Can lose data, but can’t have corruption!
- fsck can scan system after crash, repair corruption
  - Must ensure that corruption is repairable by fsck!
- Ordered updates
  - ex: update inode before dir entry
  - Structure before pointer, null out pointers before reuse, don’t null out all ptrs to live object
- Soft updates
  - Problem with ordered updates: cyclic dependencies. Solution:
  - Write in any order, but track dependencies, temporarily roll back uncommittable changes
- Journaling
  - Set aside portion of disk as log; group an atomic unit of operations into a log entry
  - Write to log before performing operation; on crash, start at oldest relevant log entry and replay

Networking: Overview & Issues

- Communication between two applications on different computers
- Layering and encapsulation
  - Application data encapsulated by transport layer, which is encapsulated by IP, etc
  - “Wrap” to transport, “unwrap” to deliver application data
  - IP provides abstraction of unreliable transport…
  - …so OS implements TCP on top for reliability — sending acks, tracking congestion, etc
- OS provides user-level abstraction of socket: bidirectional byte stream (pipe)



Networking: Implementation

- mbuf used to store packet data
  - Packets made up of multiple mbufs
  - mbufs are basically linked-lists of small buffers
  - Don’t want to store packets in contiguous memory
    - Avoid moving data when adding new headers
- Socket has send/recv buffers, queues of incoming connections (if listen), protocol control block (for protocol-specific info), and protosw*
  - protosw structure as abstract network protocol interface
    - Goal: abstract away differences between protocols
    - In C++, might use virtual functions on a generic socket struct
    - Here, just put function pointers in protosw structure

Networking: Networked file systems

- From application perspective, same as local FS, but data could be remote!
- Easier sharing, access more data than available locally
- But now have to deal with network problems
- NFS: virtualize FS with “vnode” generic interface
  - vnode operations send RPC over network
  - Designed to be stateless (apart from files stored), for easier crash recovery
  - Requests are self-contained + (mostly) idempotent

Protection: Basics/Unix

- How to limit access to resources (files, devices, etc.)?
- Access Control Lists
  - each "object" has an associated list of “subjects” (e.g. processes, users) who can access
  - OS checks that the user is on the list
- in Unix, each process has a user id & one or more group id’s
  - inode stores: owner, file group, permissions for: user, anyone in file group, and other

Protection: Security Holes

- setuid/setgid allow temporarily/partially privileged access
  - e.g. allow the user to change their own password in the password file, but don’t want to allow reading the password file
  - allows a program to run with the effective permissions of the file’s owner/group
  - Introduces a bunch of security problems…
- Time-of-check to time-of-use (TOCTTOU) bugs
  - Attacker can change state while system is in between checking permission and doing action

Protection: Capabilities

- Process explicitly invokes particular capabilities rather than “ambient authority”
  - e.g. for each process, store a list of objects it can access
  - Kind of opposite to Access Control Lists
- Avoid the “confused deputy problem”
  - program inherits privileges from multiple sources
  - can be tricked into malicious misuse of one privilege by a different, unrelated source/user

Security: DAC vs. MAC

- Discretionary Access Control (DAC)
  - Prevents unauthorized access to resource
  - Does NOT prevent authorized access from leaking information
  - e.g. Unix permissions
- Mandatory Access Control (MAC)
  - Prevents both unauthorized access and unauthorized disclosure
  - e.g. stop an infected virus scanner from leaking your data

Security: Security Levels

- A security level/label is a pair (c,s) where:
  - c = classification (e.g. 1 = unclassified, 2 = secret, 3 = top secret)
  - s = category-set (e.g. Nuclear, Crypto, Russia)
- (c1,s1) dominates (c2,s2) iff c1 ≥ c2 and s1 ⊇ s2
  - inverse of dominate = “can flow to”
- Subjects and objects are assigned security levels
- DAC
  - If subject S observes object O, level(S) must dominate level(O)
  - e.g. unclassified subject can’t read top secret file
- MAC
  - If any subject both observes O1 and modifies O2, then level(O2) dominates level(O1)
  - e.g. no subject can read a top secret file, then write a secret file
- Covert channels will be present when resources are shared
  - Generally, can only hope to bound bandwidth

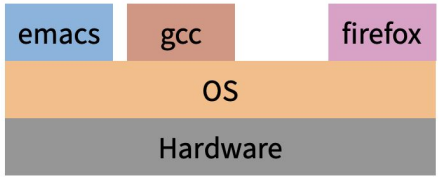
Security: Low-Water Mark Access Control (LOMAC)

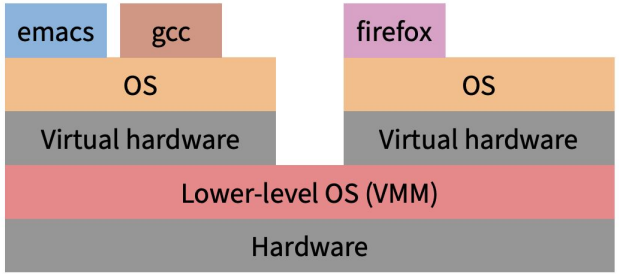
- Make MAC more palatable (outside military)
- Focus on integrity
  - often more important goal
  - e.g. don’t want viruses tampering with all your files
- Subjects are jobs (essentially processes), labeled with integrity number
  - Subjects can be reclassified on observation of low-integrity data
- Objects (files, pipes, etc.) also labeled with integrity level
  - Object integrity level is fixed, and inherited from creator
- Low-integrity subjects cannot write to high-integrity objects

Virtual Machines: OS vs. VM

- OS and virtual machines allow sharing of hardware with protections
- OS exposes hardware through a process abstraction
  - Abstracts hardware to makes applications portable
  - Makes finite resources (memory, # CPU cores) appear much larger
  - Protects processes and users from one another
- Virtual machine monitor (VMM) exposes hardware through a hardware abstraction
  - Makes hardware resources appear larger or smaller
  - Allows almost any software {OS + Apps}
  - Protects {OS + Apps} from each other

Process	Hardware
Non-privileged registers and instructions	All registers and instructions
Virtual memory	Both virtual and physical memory, MMU functions, TLB/page tables, etc.
Errors, signals	Trap architecture, interrupts
File system, directories, files, raw devices	I/O devices accessed using programmed I/O, DMA, interrupts





Virtual Machines: Benefits/Methods

- Benefits
  - Software compatibility: any OS/app can run (even really old ones)
  - Hardware sharing: allow multiple servers to run on the same hardware
  - Isolation between VMs
- Ways to Virtualize
  - Complete Machine Simulation (too slow)
  - “Basic” CPU + memory + I/O virtualization
  - Binary translation
  - Hardware-assisted virtualization

Virtual Machines: Basic Virtualization

- CPU Virtualization
  - Guest OS runs in user mode
  - Trap to VMM when Guest OS executes privileged instructions
- Virtual Memory Virtualization
  - Guest OS controls Guest VA → Guest PA mapping
  - VMM controls Guest PA → Host PA Mapping
  - From hardware POV: shadow page tables directly map Guest VA → Host PA
- I/O Device Virtualization
  - Trace/trap communication + simulate device behavior

Virtual Machines: Advanced Implementations

- Binary translation
  - Dynamically rewrite code to replace sensitive instructions with jumps into the VMM
  - Most instructions are not sensitive so they can be translated identically
  - Post-translation, code acts safely, so can run in privileged kernel mode
- Hardware-assisted virtualization (Class covers AMD)
  - Hardware supports “guest mode”
  - VMM transfers control to guest using new “vmrun” instruction; load state from VMCB
  - Hardware defines VMCB control bits to tell the CPU which instructions should cause guest mode to “EXIT” (save state to VMCB on exit)

Admin

Final exam 3:30pm-6:30pm next Monday, June 9, in **380-380C** (not Skilling!)

- **Cumulative** — covers entire course
- Open note, but no textbook or electronic devices
- Bring lecture note printouts
- SCPD must register exam monitor or show up in person
  (no need to request permission to show up in person)
- Please remind us if you need OAE arrangements
- Please send us your exam monitor if you are a non-SCPD with permission to take the exam under SCPD rules. (SCPD won't send the exam to your monitor, so we have to do it directly.)
- Please post any questions under **Lectures+Exams** on Edstem!

Good luck!