Final Review Section

Josh Cho

2023-03-17 Fri

Outline

Admin

Time: Wednesday, March 22nd, 3:30pm-6:30pm
Place: Skilling Auditorium
"The exam is open-note–you can bring any printed materials you want except for the textbook–but you may not use any electronic devices during the exam."

Agenda

Covered in Midterm Review

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

Today

- Memory Allocation
- Device I/O
- File Systems
- Networking
- Security
- Virtual Machines

Agenda

I will focus on content in lectures that were not covered in projects (e.g. discussion on indexed files will be short).

Memory Allocation (L10)

minimize fragmentation

- different lifetimes
- different sizes

allocation strategies (e.g. best fit, first fit)

- tradeoffs/pathologies based on workload characteristics

ramps, peaks, and plateaus

- e.g. arena allocation

Faults and GC (L10)

fault + resumption = power

- level of indirection
- e.g. sub-page permissions, vm, concurrent snapshotting, mmap

garbage collection

- e.g. stop & copy without stop gc
  - mutator runs & collector collects, uses fault + resumption
- dealing with reference counts (e.g. ownership in Rust)

How to communicate with device (L11)

Memory-mapped device registers

- regular read/write interface except access device's registers directly

Memory-mapped device memory

- regular read/write interface except access device's internal memory

Special instructions (e.g. inb, outb)

- communicates using port numbers

DMA (direct memory access)

- CPU offloads read/write of main memory to device/DMA engine

Device Driver (L11)

1. Polling
   - loop until some condition X is true
2. Interrupt-driven devices
   - ask card to interrupt CPU on events

Disk (L11)

- remember that placement & ordering of disk requests is important
  - sector is the unit of atomicity
  - sequential I/O is much faster than random
  - long seeks much slower than short ones
  - see slides 22-29 in L11 for more details on properties of disk

Disk Scheduling (L11)

- FCFS (First come first serve)
- SPTF (Shortest positioning time first)
- "Elevator" Scheduling (or SCAN)
  - seek must be in the same direction

Flash Memory (L11)

- flash memory has completely solid state (no moving parts)
  - e.g. NAND flash, NOR flash, SLC, MLC
  - limited # of overwrites
    - solved with FTL (Flash Translation Layer, see slides 41-45)
  - limited durability

File Systems (L12)

- Contiguous Files (strawman)
- Linked Files
  - FAT (file allocation table)
    - Key optimization for pointer chasing
- Indexed Files
  - Fixed but large size

Directories (L12)

- "Everything is a file." (UNIX)
  - directories are files with special format
- root directory is always inode #2 (0 and 1 are reserved)
- each process has a current working directory "cwd"

Hard and Soft Links (L12)

- Hard link
  - allows more than one dir entry to refer to a file
- Soft/symbolic link
  - synonyms for names
  - inode has special "symlink" bit set and name of link target

Speeding up FS (L12)

- Fragments
  - allows large block size (smaller file index), but also solving internal fragmentation
- Cylinder clustering
  - increase spatial locality wrt filesys objects
- Free map

Handling Crashes (L13)

- must handle shutdown at any point
- data loss is okay, but corruption is not!
- fsck to fix corruption
  - e.g. scans over the entire disk looking for orphaned files, leaked disk blocks

Minimizing Corruption (L13)

- Ordered updates
  - to ensure fsck works
  - e.g. write new inode to disk before directory entry
- Soft updates
  - update order may create cycles
  - break cycles by temporarily rolling back all changes that created the cycle
- Journaling
  - allow operations to act as though they are atomic
  - use a write-ahead log, then replay the log on crash

Networking (L14)

- allow two applications on different machines to communicate
- OS provides abstraction for communication
  - Handles packaging, sending, unpacking, and delivering of information
- TCP implemented by the kernel to provide a "reliable pipe" abstraction over an unreliable network
- The user-level interface provided is called a socket
- Endpoints are named by an IP-address and 16-bit port

Network Layering (L14)

- Networking protocols are organized in layers
- Application data wrapped in TCP layer
  - Contains information for implementing reliable delivery
- TCP packet wrapped in IP packet
  - Contains information for routing packets between networks
- IP packet wrapped in link layer protocol (typically ethernet)
  - Contains information for delivering packets within a network
- Layers are unwrapped to deliver data to the application

Networking Implementation (L14)

- mbuf used to store packet data
  - Packets made up of multiple mbufs
  - mbufs are basically linked-lists of small buffers
- 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

Network File Systems (L14)

- file system where data is potentially stored on other machines
- vnodes
  - virtualize the file system
  - designed for "stateless" operation
  - vnode operations perform RPC (Remote Procedure Calls)
    - request over the network

General Protection (L15)

- how do you limit access to resources (files, devices, etc.)?
- Access Control Lists
  - each "object" has an associated list of who can access "subject"
  - OS checks that the user is on the list
- in Unix, each process has a user id & one or more group id's

Basic Security Issues (L15)

- setuid: how to allow partial privileges?
  - e.g. what to allow the user to change their own password in the password file but don't want the allow reading the password file
  - setuid allows a program to run at with the effective permissions of the files owner
- TOCTOU (Time-of-check, Time-of-use) bug
  - e.g. first check if you are allowed to execute, then execute
  - Problem: attacker can change the state between the check and the execution

Capability-based Approach (L15)

- Confused deputy problem
  - inheriting multiple privileges
- for each process, store a list of objects it can access
  - process explicitly invokes particular capabilities
  - solves confused deputy problem

Advanced Security (L16)

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

MAC (Mandatory Access Control) (L16)

- A security level or label is a pair(c,s) where:
  - c=classification – e.g., 1=unclassified, 2=secret, 3=topsecret
  - s=category-set – e.g., Nuclear, Crypto
- (c1,s1) dominates (c2,s2) iff c1 $\geq$ c2 and s1 $\supseteq$ s2
- Subjects and objects are assigned security levels
- Prevent leaking classified by checking the dominates relationship
  - e.g. kill any process that attempts to write to a with security level (c',s') if it has already read from a file with security level (c,s) where (c,s) dominates (c',s')

LOMAC (Low water Mark Access Control) (L16)

- LOMAC's goal: make MAC more palatable
- Concentrates on Integrity
  - More important goal for many settings
  - E.g., don't want viruses tampering with all your file
- Security: Low-integrity subjects cannot write to high integrity objects
- Subjects are jobs (essentially processes)
  - Each subject labeled with an integrity number (e.g., 1, 2)
  - Higher numbers mean more integrity

OS vs. VM (L17)

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

Virtual Machines (L17)

- Benefits
  - Software compatibility: any OS/App can run (even really old ones)
  - Hardware sharing: allow multiple servers to run on the same hardware
- Ways to virtualize
  - Complete Machine Simulation (too slow)
  - Basics
  - Binary Translation
  - Hardware-assisted virtualization

VMM Basics (L17)

- CPU Virtualization
  - Guest OS to runs in user mode
  - Trap to VMM when Guest OS does sensitive things
- Virtual Memory Virtualization
  - Guest OS controls Guest Virtual to Guest Physical Address mapping
  - VMM controls Guest Physical to Host Physical Mapping
- I/O Device Virtualization
  - Simulate device behavior

Virtual Machine Implementations (L17)

- 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
- Hardware-assisted virtualization
  - Hardware supports “guest mode”
  - VMM transfers control to guest using new “vmrun” instruction
  - Hardware defines VMCB control bits to tell the CPU which instructions should cause guest mode to “EXIT”

Recap

- Processes and Threads
- Virtual Memory
- Concurrency
- Synchronization
- Linking
- Memory Allocation
- Device I/O
- File Systems
- Networking
- Security
- Virtual Machines