CS 140: Final Review

March 13, 2020

Plan of attack

- **High level view of all the topics cover in the class**
  - Idea: find out what you need to revisit
- **Focus more on the post midterm material**
  - Final will be cumulative
- **Chance to think about how all these pieces fit together**

Topics

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

What is an Operating System?

- **Layer between applications and hardware**
  - Allows hardware to be shared
  - Makes hardware useful to the programmer
  - Provides abstractions for applications
  - Provides protection
- **The view of the OS from the application**
- **The view from within the OS**

Processes and Threads

- **User abstraction for a collection of work that uses the CPU**
- **A *process* is an instance of a single program**
- **A *thread* is a single execution context**
  - One process can have multiple threads
  - Threads within the same process have shared access to memory

Kennel-level vs User-level Threads

- **Kernel-level threads**
  - Created using a sys-call (can be slow)
  - Execution order (scheduling) determined by the kernel
  - Synchronization primitive provided by the kernel
- **User-level (green) threads**
  - Implemented in user space and layered on top of kernel-level threads
  - Must wrap sys-calls that can cause the kernel-level thread to block
  - Thread creation is often faster

Scheduling Thread Execution

- **Given a number of runnable threads/process, which should we run?**
- **Considerations:** throughput, response time, CPU utilization, etc.
- **Scheduling Policies:**
    - First come first server
    - Shortest job first
    - Round robin
    - Priority
    - MLFQS (multi-level feedback queues)

Virtual Memory

- **Want each process to have illusion of a very large memory**
- **Create mapping from Virtual to Physical memory**
    - Give each process a large virtual address space
    - Dynamically assign virtual address regions to real physical memory
- **Current mapping held in a per process page table**
- **MMU manages translation from virtual to physical memory on each access using page table information**
- **TLB caches page table information to make lookups faster**
- **Currently unused virtual memory regions can be evicted**

Concurrency

- **What is our execution model for interleaved threads of execution?**
- **Sequential consistency**
  - the order in which things actually happen to the order in which they are written in your code
  - optimizing for faster code execution hard for both compiler and CPU
- **More relaxed consistency models used in practice**
  - Use atomics and fences to enforce cases where consistency is needed
- **Must to careful to handle races in concurrent programs**
  - Race conditions: Timing/ordering of thread exception shouldn’t affect correctness
  - Data races: two threads shouldn’t simultaneous access the same memory region if one of the accesses is a write

Synchronization

- **Want to ensure no races from concurrent execution**
- **Use synchronization primitives**
  - Locks, semaphores, condition variables
  - Need to be careful to avoid deadlocks
- **Use carefully designed concurrent algorithms with atomics**
  - Atomics use to enforce exclusive access to single variables and well as define desired consistency semantics
- **Other related techniques**
  - RCU, FUTEX, Transactional memory

Linking

- **Combine object files in to a run-able executable**
  - Compiler generates object file from single source file but doesn’t know the final location of function and variables in other files
  - Need to convert symbols (names of functions and variables) to memory addresses and patch them up in the code
- **Linker 2 pass execution**
  - Pass 1:
    - Decide where each object file’s code and data will resided in memory
    - Collection information about all locations of functions and variables (symbol table)
    - Collection information about all the references that need to be updated (relocation table)
  - Pass 2: Use the symbol table to patch all locations specified in the relocation table
- **Linking can happen at link time (static linking) or load/run time (dynamic linking)**

Memory Allocation

- **Dynamically give programs arbitrary size chucks of memory**
- **The core fight: minimize fragmentation**
  - Allocation have different sizes and life-times leaving “holes” in the memory space
  - Various allocation policies to try to mitigate
- **Can use garbage collection in languages that control pointers**
  - Move live data to compact use of memory to free up contiguous blocks

Ways for OS (drivers) to do IO

- **Special instructions (e.g. inb, outb)**
  - Communicates with devices using specified “port” numbers
- **Memory-mapped device registers**
  - Regular memory read/write interface except access go directly to a device’s registers
- **Memory-mapped device memory**
  - Regular memory read/write interface except access go directly to a device’s internal memory
- **DMA (Direct Memory Access)**
  - CPU offloads read/write of main memory to device/DMA engine

File systems

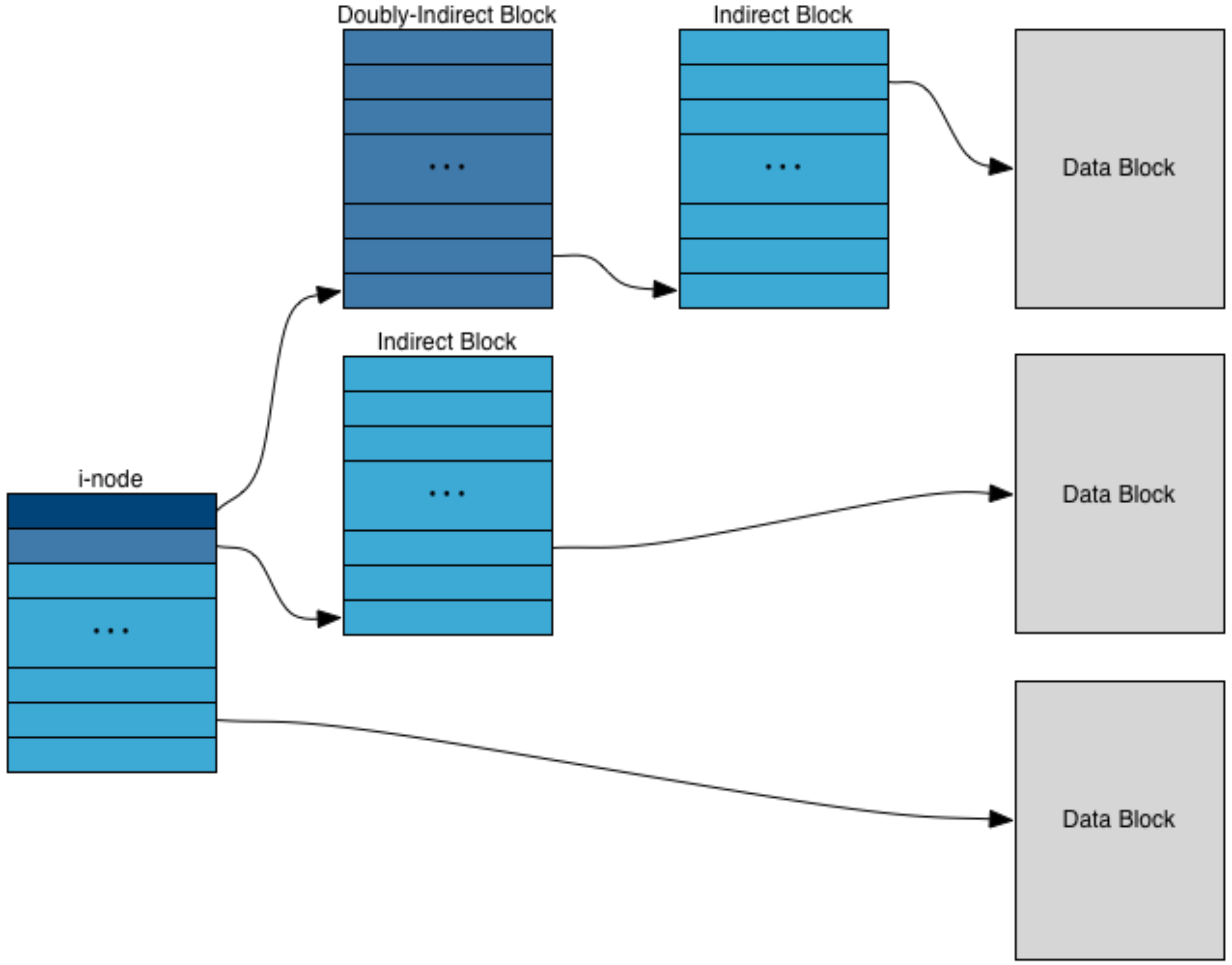
- **Need a way to persist and organize data between restarts**
- **Associates names with bytes on disk**
    - Want an organization and naming that humans can remember
- **Most file systems designed around disks**
    - Optimized for fast sequential access and slow random access
- **Need to handle unexpected crashes**

File systems on Disk

- **How do you track the blocks associated with a file?**
- **Contiguous allocation “extent-based”**
  - Know the started block location and the length
- **Linked files**
  - Each block contains the location of the next block
- **FAT (File Allocation Table)**
  - Like linked files but keep link information for all files in one (or two) blocks
- **Indexed Files**
  - Keep an index for each file (inode)

Muti-level indexed files

- **Files divided into blocks of 4 Kbytes**
- **Blocks of each file managed with multi-level arrays of block pointers**
- **File descriptor (i-node) = 14 block pointers, initially 0 ("no block")**
  - First 12 point to data blocks (direct blocks)
  - Next entry points to an indirect block (contains 1024 4-byte block pointers)
  - Last entry points to a doubly-indirect block
- **Maximum file length is fixed, but large**
- **Indirect blocks aren't allocated until needed**



File Naming and Directories

- Directory contains a mapping from name to an inode
- Directories are just files with a specified format
- Name to inode mapping can be name to file or name to directory
- Multiple directories can contain file names that point to the same inode (hard-links)
- Names can also point to a string that resolves at time of access (soft-links)

Handling Crashes

- **Machine could shut down at literally any point**
- **Need to make sure that the file system is never corrupted**
    - Ok with (some) data loss
    - NOT ok with corruption
- **Possible solution: Fix corruption (fsck)**
    - After crash fsck can be run to try to fix disk corruption and clean up the disk
    - Scans over the entire disk looking for orphaned files, leaked disk blocks, etc
    - Issue: need to make sure that no corruption can occur that is beyond repair

Minimizing Corruption

- **Ordered Updates**
  - Ensure write are permitted back to disk in an order that is recoverable
  - e.g. add the new inode before updating the directory
- **Soft Updates**
  - Update order may create cycles
  - Break cycles by temporarily roll back all changes that created the cycle
- **Journaling**
  - Allow operations the act as though they are atomic
  - Use a write-ahead log to persist the intent; replay the log if there is a crash

Networking

- **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

- **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

- **mbuf used to store packet data**
    - Packets made up of multiple mbufs
    - mbufs are basically linked-lists of small buffers
    - Allows easy adding and removing of data from the ends
- **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

Basic Security

- **How do you limit access to resources (files, devices, etc.)?**
- **Access Control Lists**
  - Each “object” has an associated list of who has access
  - OS checks that a user is on the list before granting access to the object
- **Capabilities**
  - Each user (program) has a list of “objects” that it’s allowed to access
  - OS checks that a user has the capability before granting access

Basic Security Issues

- **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
  - Solution: support method of doing the check and execution atomically

Advanced Security

- **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

Mandatory Access Control (MAC)

- **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 $s1 \supseteq s$**
- **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′)
  - e.g. kill any process that tries to write to an unclassified memo after reading a classified intelligence report

LOMAC (Low water Mark Access Control)

- **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
  - Subjects can be reclassified on observation of low-integrity data
- **Objects (files, pipes, etc.) also labeled w. integrity level**

Advanced Security Issue: Side Channels

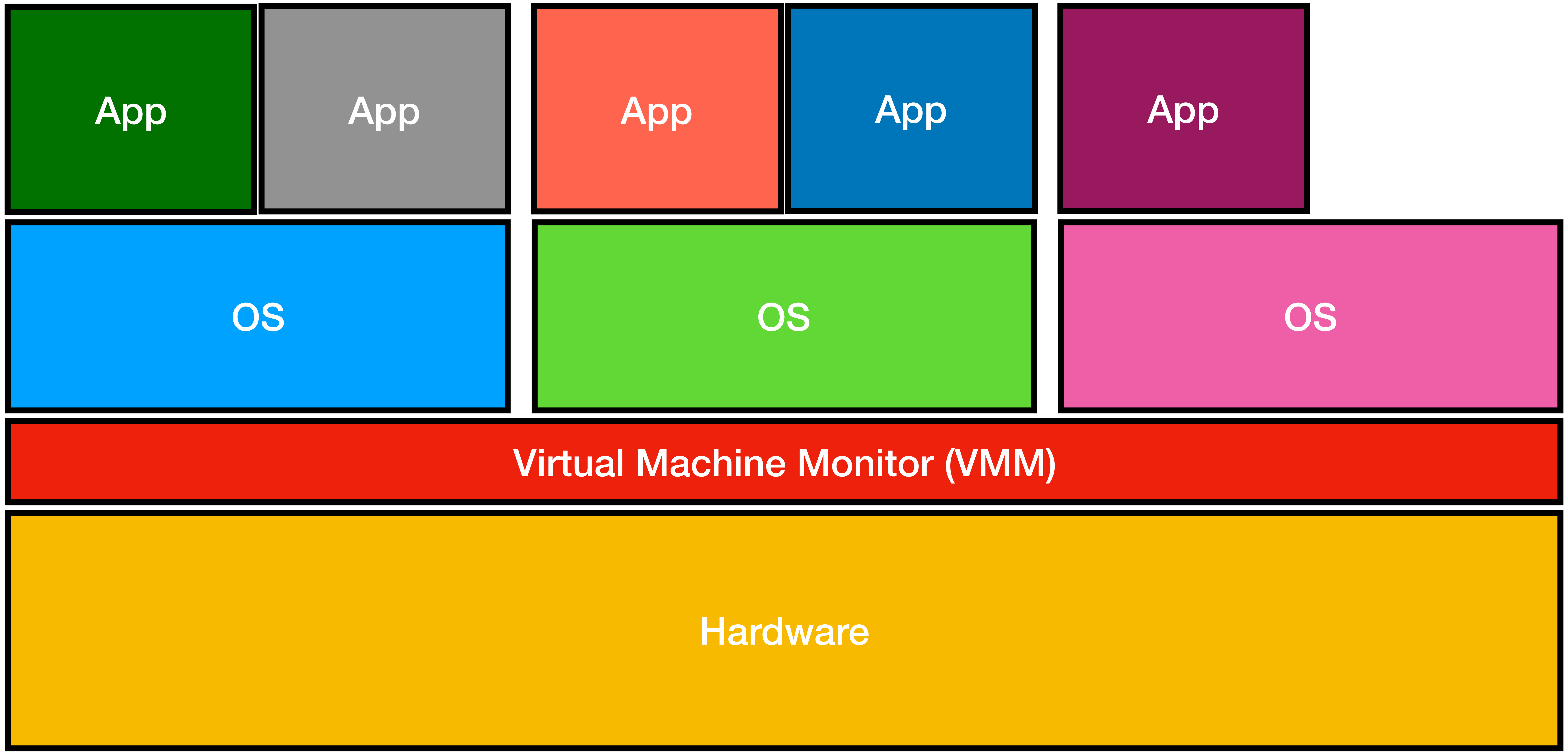
- **Even with access controls process can communicate in an unauthorized manner**
- **Covert storage channels**
    - e.g., high program inherits file descriptor-Can pass 4-bytes of information to low program in file offset
- **Timing channels**
    - e.g. use high and low CPU utilization to single 1s and 0s; monitor progress of busy loop to detect CPU utilization
- **In general, can only hope to bound bandwidth of covert channels**

Operating Systems vs Virtual Machines

- **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 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 Machine

- **Thin layer of software that virtualizes the hardware**



Virtual Machines

- **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

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

Virtual Machine 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
- **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”

Topics

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

Good luck!