CS 212 Midterm Review

Spring 2026

Admin — Midterm Exam

- **Exam: In class, Monday May 4**
- Open note (no textbook or electronic devices)
  - Bring lecture note printouts
- SCPD students
  - Must register exam monitor or show up in person
- OAE arrangements
  - Remind us if you need accommodations
  - Email us ASAP
- Non-SCPD with SCPD permission
  - Send us your exam monitor directly

Operating Systems

Layer between applications and hardware

Operating Systems

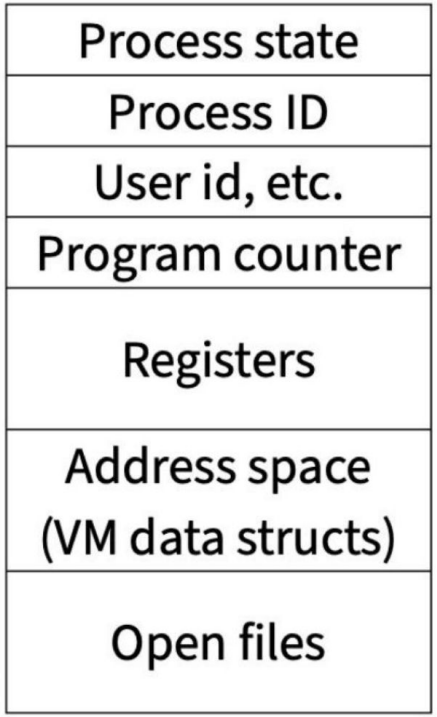
- Abstracts hardware for applications
- Provides protection
  - Preemption
  - Interposition
  - Privileged vs unprivileged
- System calls
- CPU protection → Timer interrupt
- Memory protection → Address translation

Processes & Threads

Execution abstractions

Process

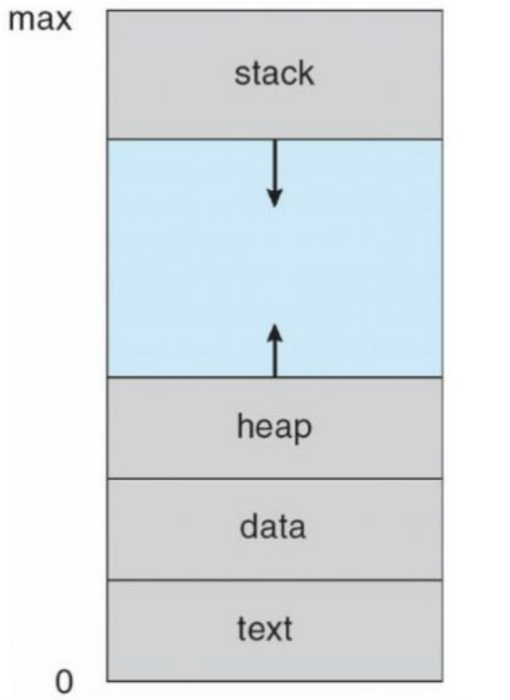
- Instance of a program running
- Why?
  - Multitasking
    - Increased CPU utilization, lower latency
    - But not always better
- Process Control Block (PCB)
  - Stores: state, registers, open files, etc.
  - Equivalent: struct thread in Pintos
  - Fields: process state, PID, user ID, program counter, registers, address space, open files



PCB

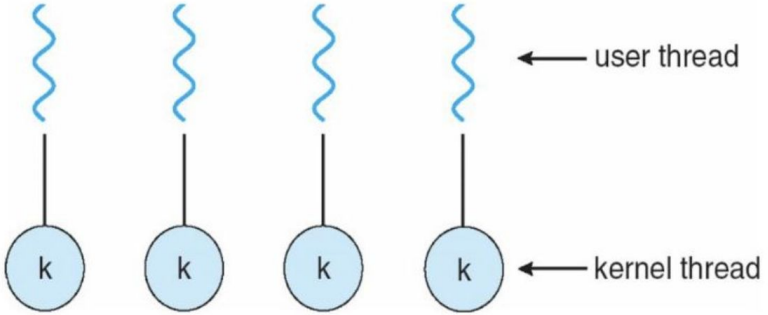
Process

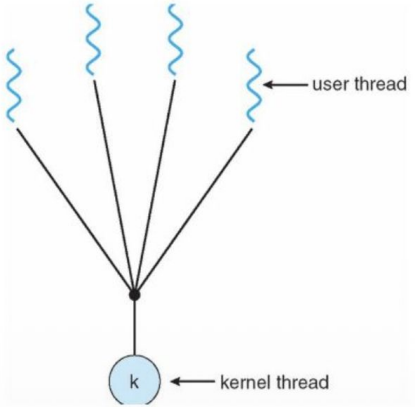
- Each process has its view of the machine
- Process interaction can happen through
  - Through files
  - Passing messages through kernel
  - Sharing a region of physical memory
  - Through asynchronous signals and alerts



Threads

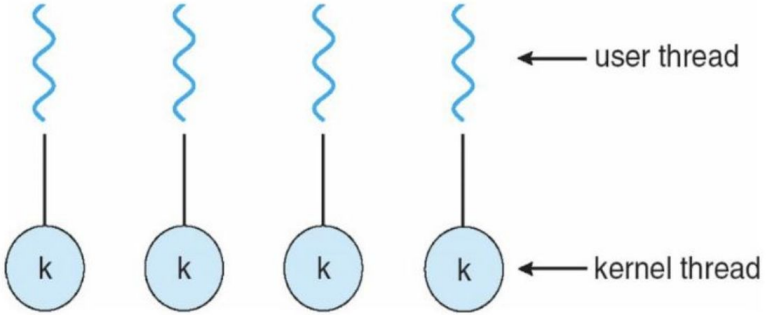
- Schedulable execution context; shares process address space
- Why?
  - More lightweight
  - Per-Process concurrency
  - Possibly per-process multi-core execution

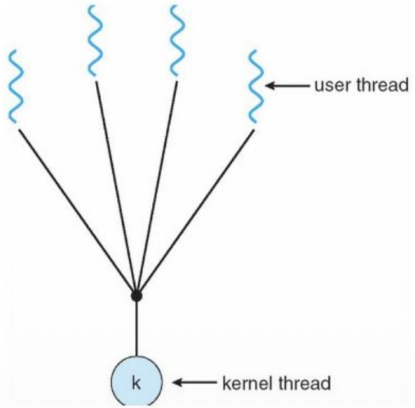




Threads

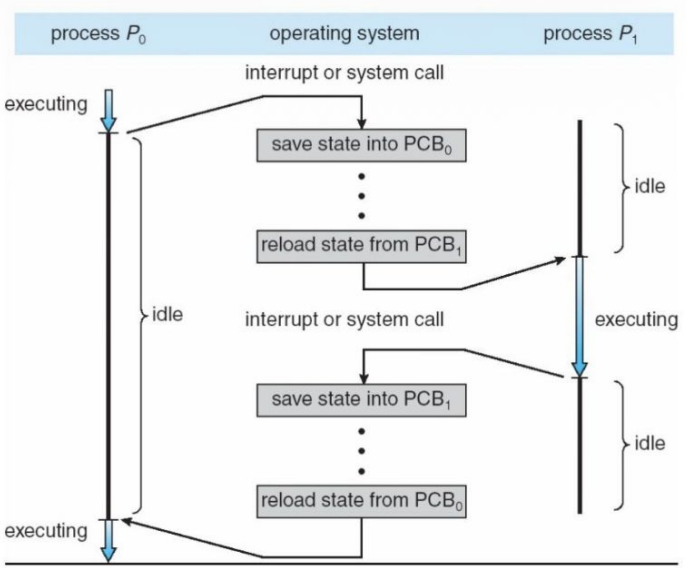
- 1:1 Threading
  - More scheduling control
  - Still heavy weight due to syscalls
  - Everything must go through the kernal
- N:1 Threading
  - Implemented as a library
  - Lightweight and flexible
  - Problems: No multi-core execution, blocking syscalls, page faults
- N:M Threading
  - Same problems as N:1
  - Hard to keep as many CPUs as kthreads





Context Switches

- Change which process is running
- How?
- When?
  - State change: blocking call or device interrupt
  - Preemption via traps (syscall, page fault, illegal instruction)
  - Periodic timer interrupt
- Cost
  - CPU time in kernel
  - Cache, buffer cache, TLB misses (indirect costs)



Scheduling

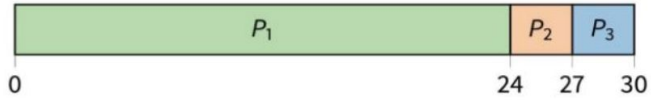
Which process runs next?

Scheduling — Goals & Criteria

- Competing goals & we can’t optimize all of them
  - Fairness
  - Priority
  - Deadlines
  - Throughput
  - Efficiency
- Criteria
  - Throughput: # processes completing per unit time
  - Turnaround time: total time for a process to complete
  - Response time: time from request to first response
  - CPU utilization & waiting time

Scheduling Algorithms

- FCFS
  - Simple; bad for IO-bound jobs behind CPU-bound ones
- Shortest Job First (SJF)
  - Minimizes average wait; can cause starvation
- Round-Robin
  - Fair; context switch overhead with same-sized jobs
- Priority Scheduling
  - Starvation of low-priority jobs
- MLFQS
  - Dynamic priorities; approximates SJF adaptively
- Virtual time (fair-share)



Multiprocessor Scheduling

- Which CPU runs which process?
  - Consider
    - Load imbalance
    - Direct costs (CPU time)
    - Indirect costs (cache misses)
- Affinity scheduling
  - Keep process on same CPU to preserve cache locality
- Gang scheduling
  - Schedule related processes/threads together

Concurrency

Synchronization & coordination

Concurrency

- Data races
  - Multiple threads access same data
  - At least one access is a write
- Critical Section Requirements
  - Mutual Exclusion
  - Progress
  - Bounded Waiting
- Mutexes
  - Pintos uses struct lock
- Condition Variables
  - Avoid busy waiting
  - Useful in producer-consumer problems
- Semaphores
  - Counter-based; differ from condition variables

Implementing Synchronization

- Disable Interrupts
  - Fine for uniprocessors
  - bad for multiprocessors
- Spinlocks
  - Simple
  - Wastes CPU while spinning
- CPU locks memory system around read/write
- Modern OSes: designed for multiprocessors⟶ need fine-grained locks

Deadlock Conditions & Fixes

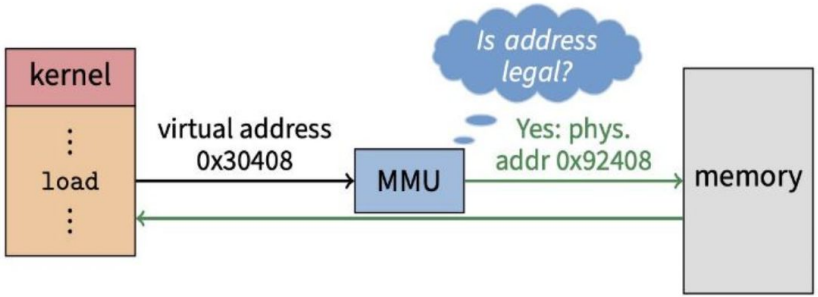
- Necessary conditions for deadlock
  - Limited access (mutual exclusion)
  - No preemption of held resources
  - Multiple independent requests (hold-and-wait)
  - Circular dependency in request graph
- Amdahl's Law→limits of parallelism
- Fixes
  - Restart / rollback
  - Examine
  - Partial ordering of locks
  - Transactions
  - Eliminate one of the four conditions

Virtual Memory

Address spaces & protection

Virtual Memory — Goals & MMU

- Goals
  - Each process gets its own virtual address space
  - Protection– processes can't access each other's memory
  - Transparency– program unaware of physical layout
  - No resource exhaustion
- MMU (Memory Management Unit)
  - Translates virtual address ⟶ physical address
  - Checks legality on every access



Mapping Memory

- Base + Bound
  - Physical = virtual address + base
- Segmentation
  - Divide into variable-size segments
- Paging
  - Fixed-size pages
  - Divide memory into small, equal-sized pages
  - Know what happens during a Page Fault
  - Each process has its own page table
    - Multilevel
    - Translation Lookaside Buffer (TLB) caches recently used translations
  - Eviction: LRU– Use past to predict future or approximate by clock algorithm

Memory Considerations

- Fragmentation
  - Inability to use free memory
  - External: many small holes between allocated regions
  - Internal: unused space within an allocated page/segment
- Speed
  - Disk is much slower than memory
  - 80/20 rule: keep hot 20% ('working set') in memory
- Thrashing
  - Excessive paging when working set > available memory
- Memory coherence vs consistency
  - Coherence: ordering of accesses to a single location
  - Consistency: ordering across multiple locations
  - Sequential consistency matches programmer intuition

Program Lifecycle

From source code to execution

Compiler → Linker → Loader

- Compiler / Assembler
  - Generates one object file per source file (main.c → main.o)
  - References to other files are incomplete (e.g., printf unresolved)
- Linker
  - Pass 1: coalesce segments, build global symbol table, compute virtual addresses
  - Pass 2: patch references using the file and global system table, emit result
- Dynamic Linking
  - Linked at runtime
  - Enables shared libraries
  - Risks: runtime failure, no compile-time type checking
- OS Loader
  - Reads executable into memory and starts execution

Exam Advice

- Old exams may differ in format and material coverage
- Read questions carefully
  - Notice what is and isn't specified
  - State your assumptions explicitly (e.g., sequential consistency)
- Notes strategy
  - Open note, but might be hard to find what you’re looking for
  - Make a focused cheat sheet rather than printing all slides
- Connect the dots to the labs
  - Useful for threads, scheduling, synchronization (VM not yet in labs)

Good Luck!