CS 212 Midterm Review

Spring 2025

Admin

Midterm exam in class next Monday May 5

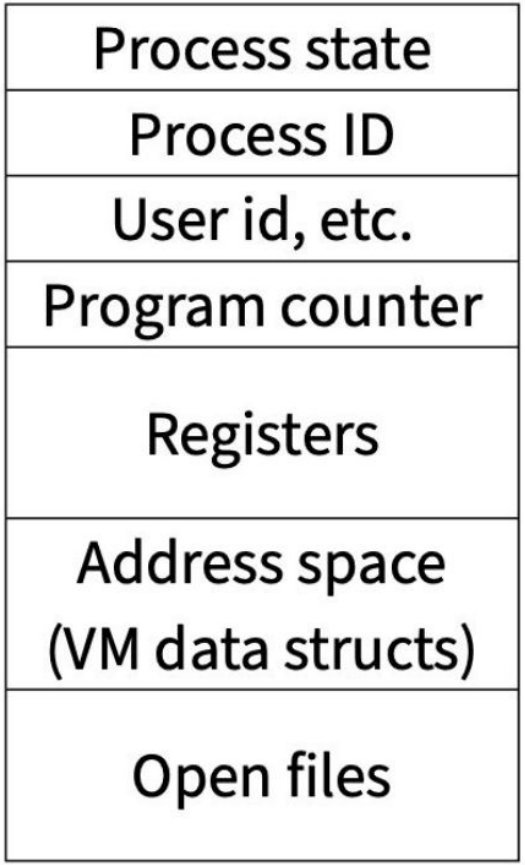
- Open note, but no textbook or electronic devices
- Bring lecture note printouts
- SCPD must register exam monitor or show up in person
  (no need to request permission to show up in person)
- Please remind us if you need OAE arrangements
- Please send us your exam monitor if you are a non-SCPD with permission to take the exam under SCPD rules. (SCPD won't send the exam to your monitor, so we have to do it directly.)

Operating Systems

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

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



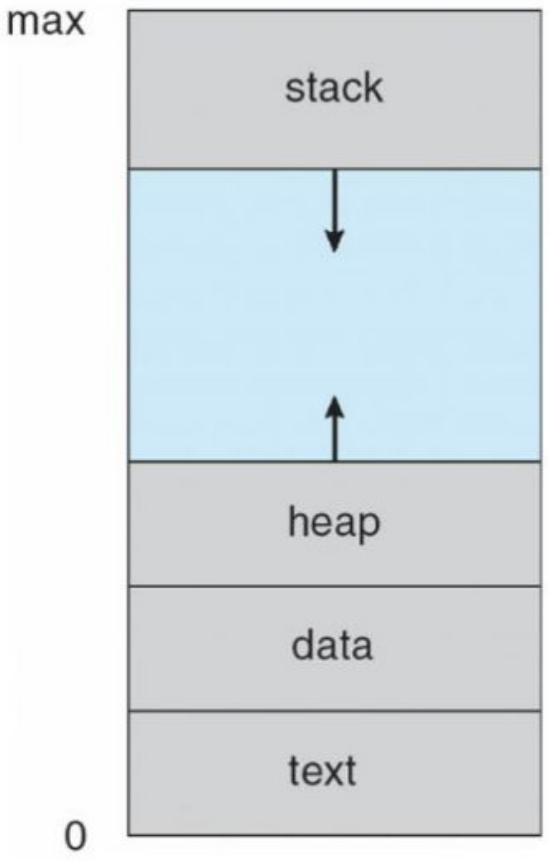
PCB

Processes Cont’d

Each process has its view of the machine

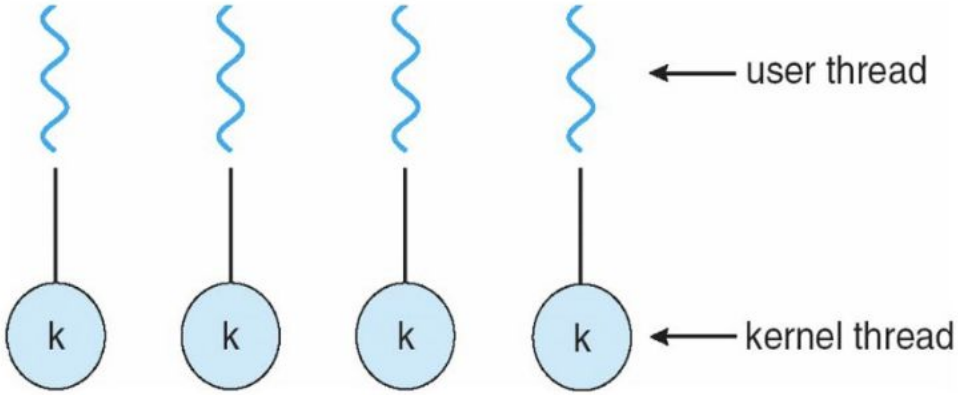
Process interaction can happen through

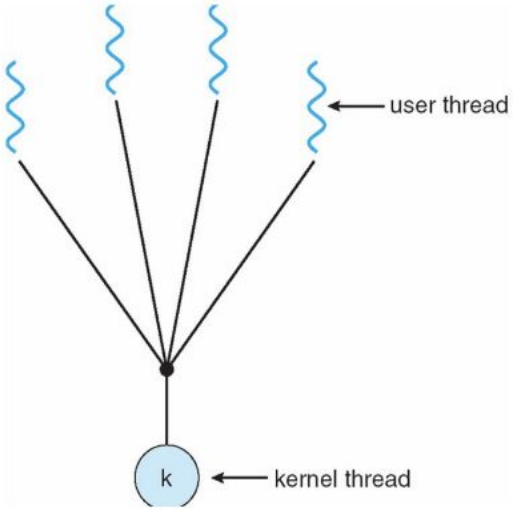
- Through files
- Passing messages through kernel
- Sharing a region of physical memory
- Through asynchronous signals and alerts



Threads

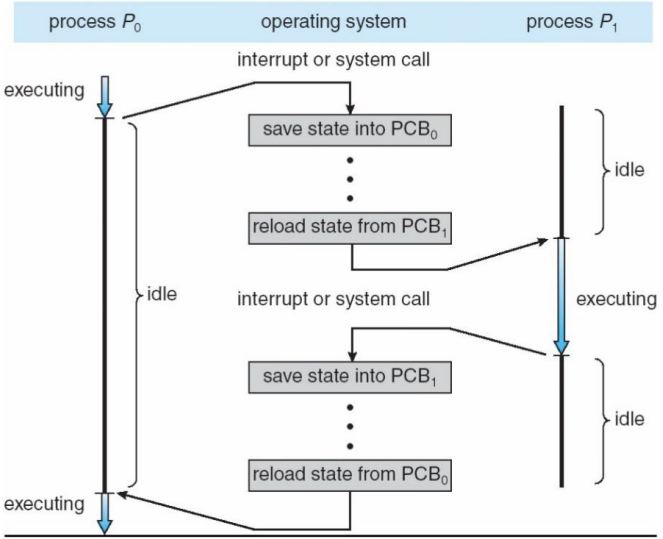
- Schedulable execution context
- Shares process's address space
- Why?
  - More lightweight
  - Per-process concurrency
  - Possibly per-process multi-core execution
- 1:1 threading
  - More scheduling control
  - Still heavy weight due to syscalls
  - Everything must go through kernel
- N:1 threading
  - Implemented as library
  - Lightweight and flexible
  - Problem: No multi-core execution
  - Problem: blocking syscalls, page faults
- N:M threading
  - Faces similar problems as previous
  - Hard to keep as many CPUs as kthreads





Context Switches

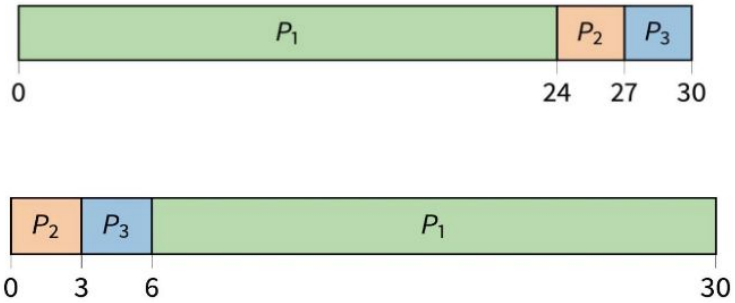
- Change which process is running
- How?
- When?
  - State change
    - Blocking call
    - Device interrupt
  - Can preempt when kernel gets control
    - Traps: system call, page fault, illegal instruction
    - Periodic timer interrupt
- Cost?
  - CPU time
  - Cache, buffer cache, TLB misses



Scheduling

- Problem
  - Given n > 1 processes, which do we run
- Multiple goals - but can’t optimize all
  - Fairness
  - Priority
  - Deadlines
  - Throughput (good overall performance)
  - Efficiency (minimize overhead of scheduler)
- Criteria
  - Throughput (# of processes that complete per unit time)
  - Turnaround time (time for each process to complete)
  - Response time (time from request to first response)
  - CPU utilization
  - Waiting time
- Context switch costs
  - CPU time in kernel
  - Indirect costs

Scheduling cont’d



- FCFS
  - CPU-bound vs IO-bound jobs?
- Shortest job first
  - Unfairness and starvation?
- Round-robin
  - Same-sized jobs, context switching?
- Priority Scheduling
  - Starvation?
- MLFQS (Multilevel Feedback Queues)
  - Dynamic priorities
- Virtual time

Multiprocessor Scheduling

- Which CPUs do we run our process on?
- Consider
  - Load balancing
  - Minimize direct / indirect costs
- Approaches
  - Affinity scheduling
    - Keep processes on same CPU
  - Gang scheduling
    - Schedule related process/threads together

Concurrency

- Default sequential consistency great for intuition
  - Not so good for hardware / compiler optimization
- Data races
- Critical Section Problem
  - Mutual Exclusion
  - Progress
  - Bounded Waiting
- Mutexes
  - Pintos uses struct lock
- Condition Variables
  - How are they useful in consumer-producer situations?
  - Avoid busy waiting
- Semaphores
  - How are they different from condition variables?
  - Counter

Implementing Synchronization

- Disable Interrupts
  - Bad for multiprocessors
  - May be efficient for uniprocessors
- Spinlocks
  - Wastes CPU
- CPU locks memory system around read and write
- Modern OSes design for multiprocessors
  - Need fine-grained locks

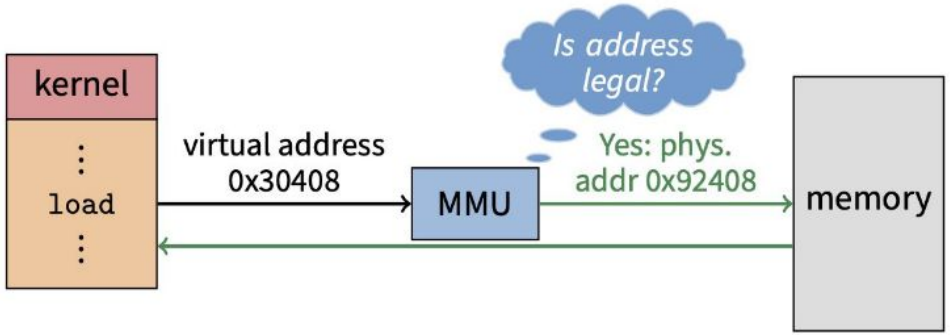
Considerations

- Amdahl’s law
- Necessary conditions for data race
  - Multiple threads access same data
  - At least one access is a write
- Necessary conditions for deadlock
  - Limited access
  - No preemption
  - Multiple independent requests
  - Circle existing in graph of requests
- Fixing deadlocks
  - Restart/examine/partial order/transactions/eliminate one condition

Virtual Memory

How should processes interact with memory?

- Goals
  - Each process → own virtual address space
  - Protection, Transparency, No resource exhaustion
- Memory Management Unit (MMU)



Mapping Memory

- Base + bound
  - Physical address = virtual address + base
- Segmentation
  - Divide memory into segments
- Paging
  - Divide memory into small, equal-sized pages
  - Each process has its own page table
    - Multilevel
    - Translation Lookaside Buffer (TLB) caches recently used translations
  - Any process can have a page
  - What happens during a page fault?
  - Eviction?
    - LRU: Use past to predict future
    - Approximate using clock algorithm

Considerations

- Fragmentation
  - Inability to use free memory
  - External fragmentation
    - Many small holes between memory segments
  - Internal fragmentation
    - Unused memory within allocated segments
- Speed
  - Disk much slower than memory
  - 80/20 rule
    - Hot 20 in memory = “working set”
- Thrashing

Memory System

- Coherence
  - Concerns access to single memory location
  - Multiple processes writing to same variable
- Consistency
  - Concerns ordering across multiple memory locations
    - If x=1, y=2, A reads x,y and B writes x=3,y=4, could A ever see x=1,y=4?
  - Sequential consistency matches our intuition

Program Lifecycle

- Source code -> program running
- Compiler/Assembler
  - Generate one object file for each source file (main.c -> main.o)
    - References to other files are incomplete (printf is in stdio.o)
- Linker
  - Combines all object files into executable file
- OS Loader
  - Reads executable into memory

Linker

- Goal
  - Object files -> executable
- How
  - Pass 1
    - Coalesce like segments
    - Construct global symbol table
    - Compute virtual address of each segment
- Pass 2
  - Patch references using file and global symbol table
  - Emit result

Dynamic Linking

- Linked at runtime
- Helps with shared libraries
- May lead to runtime failure
- No type checking

Memory allocation

- See most recent lecture

Advice

- Old exams won’t necessarily cover the same material or have the same format
- Notice what is/isn’t specified in a question (and state assumptions)
  - Sequential consistency
- Regarding relying on notes
  - Might be time-constrained
  - Create a cheat sheet instead of printing all lecture slides (or print both?)
- When reviewing the material, it may be helpful to think about the labs to connect the dots (not always the case though, VM hasn’t been covered in labs yet)

Good luck!