Midterm Review

CS 212 Winter 2024

Admin

- When?
  - Monday, Feb 12, 1:30 - 2:50 pm (in class)
- What resources are allowed?
  - Open note, printing lecture slides is ok
  - No internet, textbook, or electronics
- What material is covered?
  - All lecture material up to (and including) Wednesday
- How does it count towards my grade?
  - 50% of overall grade is: max(midterm > 0 ? final : 0, (midterm + final)/2)

Content

- Processes and Threads
- Scheduling
- Virtual Memory (HW/OS)
- Concurrency
- Synchronization
- Linking

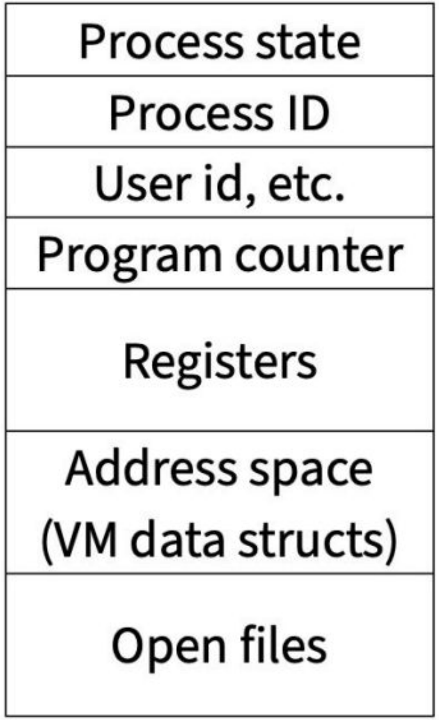
Themes

- Memory Models
  - Sequential Consistency
- Data Races
  - Implementing locks
  - Producer/Consumer
- Design Tradeoffs
  - Complexity, using past to predict future, hardware support

Processes and Threads

Processes

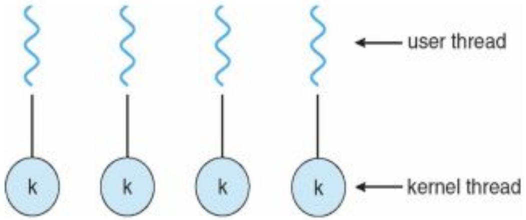
- Process
  - An instance of a program running
  - Has its own view of the machine: address space, open files
- Process control block (PCB)
  - Stores information about the process, including:
    - State (running, ready, waiting)
    - Registers
    - Virtual memory mappings
    - Open files
  - struct thread in pintos
- Why use processes?
  - Higher throughput
  - Lower latency



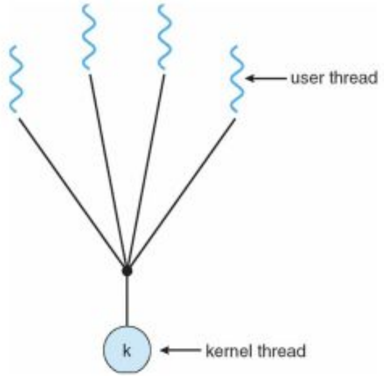
PCB

Threads

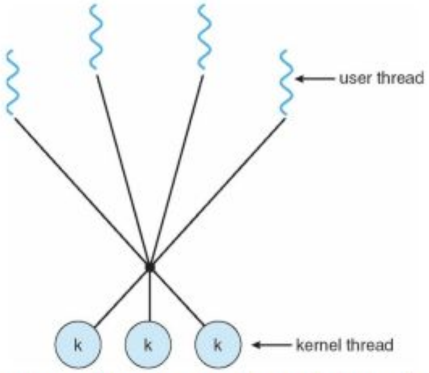
- Schedulable execution context
- Why use threads?
  - Concurrency
  - Multi-core execution
- Kernel threads
  - More scheduling control
  - Heavy weight
  - Everything must go through kernel
- User threads
  - Lightweight and flexible
  - Less control



1 user thread : **1** kernel thread



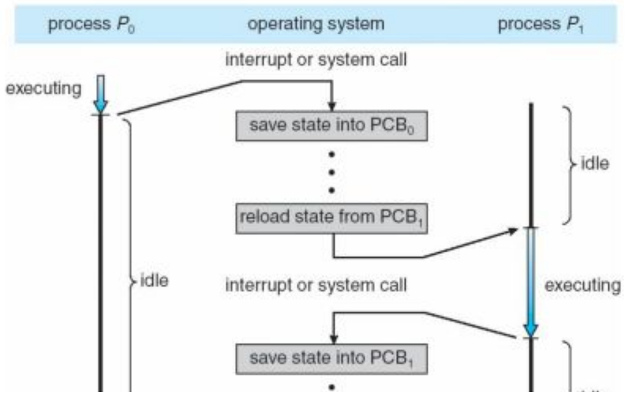
n user threads : **1** kernel thread

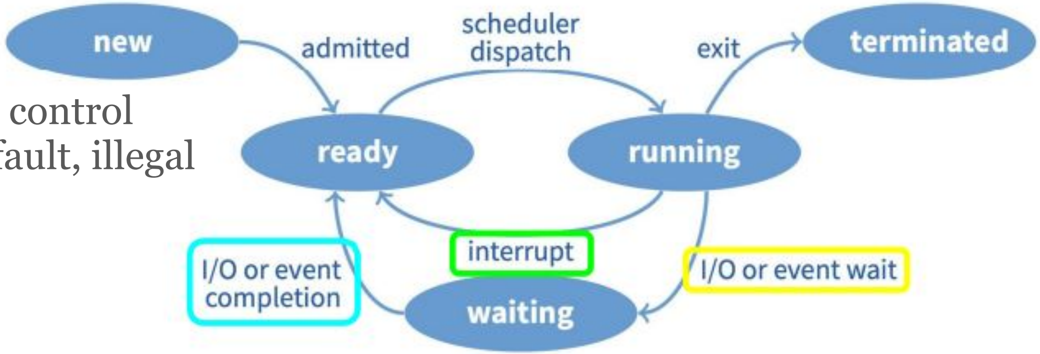


n user threads : **m** kernel threads

Context Switches

- Context switch
  - Change which process is running
- How?
  - Save registers of current thread
  - Restore registers of next thread
  - Return into next thread
- When?
  - State change
    - Blocking call
    - Device Interrupt
  - Can preempt when kernel gets control
    - Traps: system call, page fault, illegal instruction
    - Periodic timer interrupt

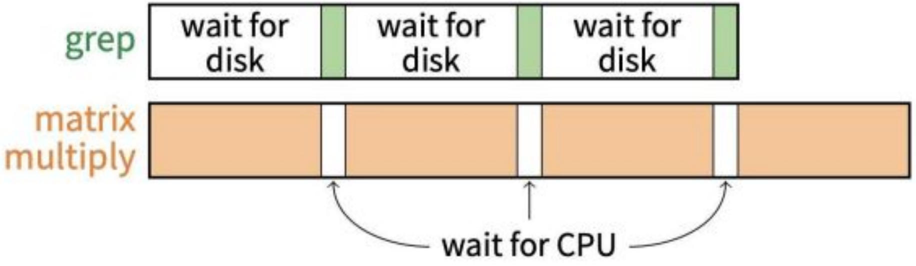




Scheduling

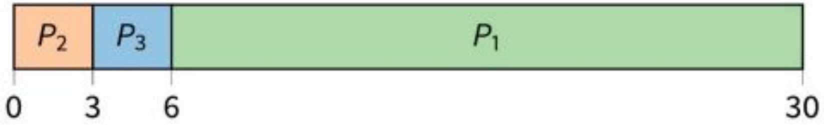
Scheduling

- Problem
  - Given a list of processes, which do we run?
- Goals
  - Throughput (number of process that complete per unit time)
  - Turnaround time (time for each process to complete)
  - Response time
  - CPU Utilization (fraction of time CPU doing productive work)
  - Waiting time
- Context switch costs
  - CPU time in kernel
  - Indirect costs



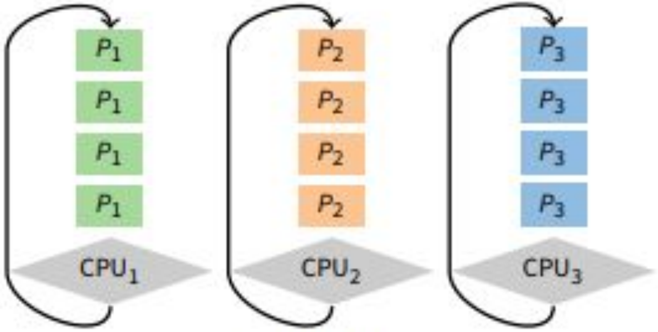
Scheduling Algorithms

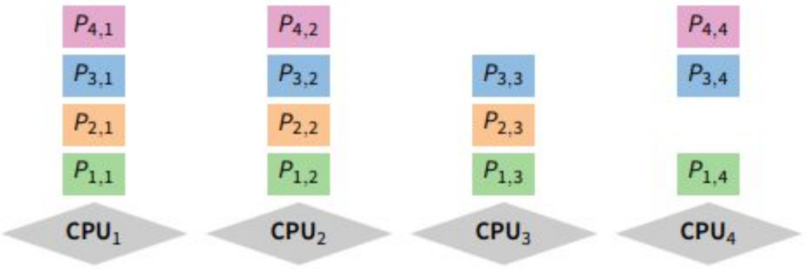
- First Come, First Serve (FCFS)
  - CPU-bound vs IO-bound jobs
- Shortest job first
  - Preemptive or non-preemptive
  - Can lead to unfairness and starvation
- Round-robin
  - Struggles with same-sized jobs
- Priority Scheduling
  - Must handle priority inversion
  - You’ve implemented this in pintos!
- MLFWS (multilevel feedback queues)



Multiprocessor Scheduling

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

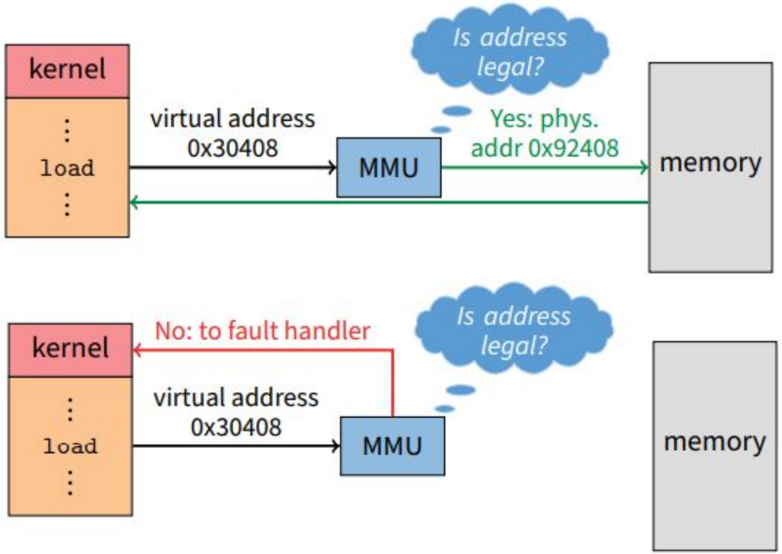




Virtual Memory

Virtual Memory Hardware

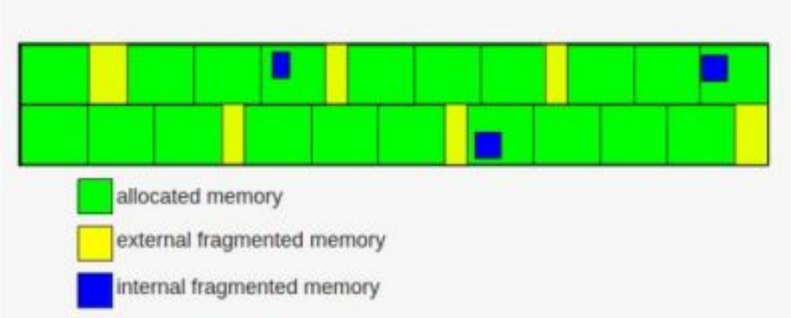
- Problem
  - Want multiple processes to co-exist
  - How should processes interface with memory?
- Issues with using physical addresses
  - Protection
  - Transparency
  - Resource exhaustion
- Solution
  - Give each program its own virtual address space
  - Memory Management Unit (MMU)
    - Translates between physical and virtual addresses



Mapping Memory

- Base + bound
  - Physical address = virtual address + base
- Segmentation
  - Divide memory into segments
- Demand Paging
  - Divide memory into small, equal-sized pages
  - Each process has its own page table
    - Translation Lookaside Buffer (TLB) caches recently used translations
  - Idle pages are stored on disk, paged in on demand
- Paging algorithms
  - FIFO: Strawman algorithm, not used in practice
  - LRU: Use past to predict future
  - CLOCK (single or multi handed): like a second-chance FIFO

VM 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”
- Local or global page allocation
- Thrashing

Concurrency & Synchronization

Memory System Properties

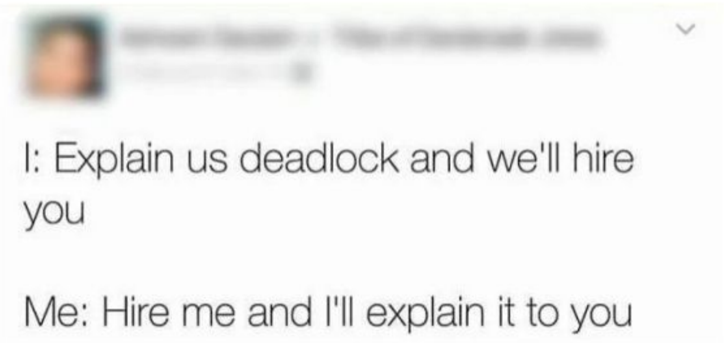
- Coherence
  - Concerns access to a single memory location
    - If A writes x=1 and B writes x=2, all processes should see the same ordering
- Consistency
  - Concerns ordering across multiple memory locations
    - If x=1,y=2, A reads x,y and B writes x=3,y=4, could A ever see x=1,y=4?
- Sequential consistency
  - Matches our intuition of consistency
  - As if all operations were executed in some sequential order
  - Downsides
    - Thwarts hardware/compiler optimizations (e.g. prefetching/reordering)
  - Requirements
    - Maintain program order on individual processors
    - Ensure write atomicity

Data Races

- **There is no such thing as a benign data race**
- Requirements to get Sequential Consistency in critical sections
  - Mutual exclusion
  - Progress
  - Bounded waiting
- How to meet requirements?
  - Synchronization primitives
    - Locks/mutexes
    - Semaphores
    - Condition variables
- What if sharing data with interrupt handler?
  - Uniprocessor: disable interrupts
  - Multiprocessor: disable interrupts + spinlock

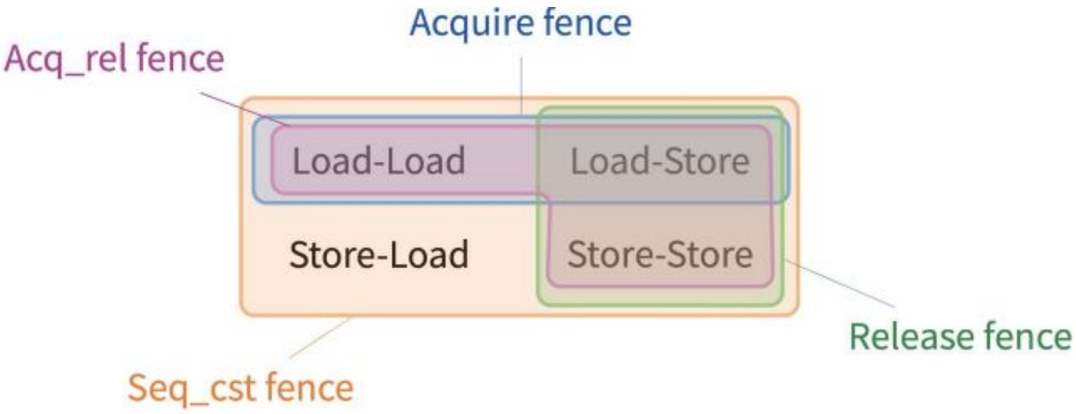
Data Races (cont.)

- Amdahl's law
  - Ultimate limit on parallel speedup if part of task must be sequential
- Necessary conditions for data race
  - Multiple threads access the same data
  - At least one of the accesses is a write
- Necessary conditions for deadlock
  - Limited access (mutual exclusion)
  - No preemption
  - Multiple independent requests (hold and wait)
  - Circularity in graph of requests
    - A holds mutex x, wants mutex y; B holds y, wants x



Memory Ordering and Fences

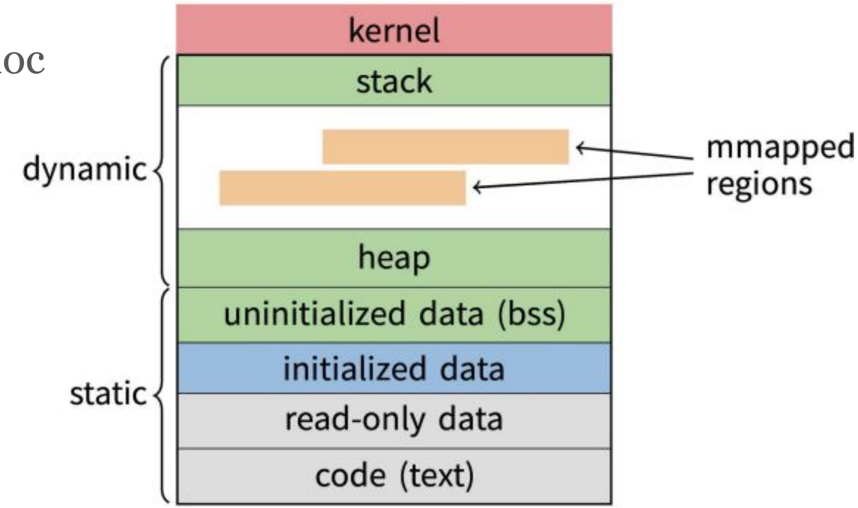
- What if we don't need sequential consistency?
  - Weaker consistency models
  - Atomics, lock-free data structures
- X-Y fence
  - operations of type X sequenced before the fence happen before operations of type Y sequenced after the fence



Linking

Memory Layout

- Heap
  - Allocated and laid out at runtime by malloc
- Stack
  - Allocated at runtime, layout by compiler
- Global data/code
  - Allocated by compiler, layout by linker
- Mmapped regions
  - Managed by programmer or linker



Program Lifecycle

- Source code → program running
- Compiler/Assembler
  - Generates one object file for each source file (e.g. main.c → main.o)
  - References to other files are incomplete (e.g. printf is in stdio.o)
- Linker
  - Combines all object files into executable file
- OS Loader
  - Reads executables 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
    - Fix addresses of code and data using global symbol table
- Dynamic Linking
  - Linked at runtime
  - Helps with shared libraries
  - May lead to runtime failure
  - No type checking

Misc. Advice

Advice

- Old exams won't necessarily cover the same material or have the same format
- Understand core themes
  - Identify races in code
  - Identify pros/cons of new approaches
  - Given a workload, be able to select a good approach
- Notice what is/isn't specified in a question (and state assumptions!)
  - Sequential consistency
  - Uniprocessor vs. multiprocessor
- Rely on notes for facts
  - Might be time-constrained
  - Create a cheat sheet instead of printing all lecture slides (or both?)
- Deep understanding of most material > cursory understanding of all

You’ve got this!