CS 212 Midterm Review

Winter 2023

Admin

Lab 2 was due at 10 am unless a member of your team is here now or you were granted an extension

Midterm

- When?
  - On Monday Feb 13th (1:30 pm PST) at Skilling
- How long?
  - 80 minutes
- What can you use?
  - Open notes
  - No textbook or electronics
- What?
  - Anything from the beginning to Wednesday’s lecture
- How does it factor into your grade?
  - 50% of CS212 grade: max (midterm > 0 ? final : 0, (midterm + final) /2)

Agenda

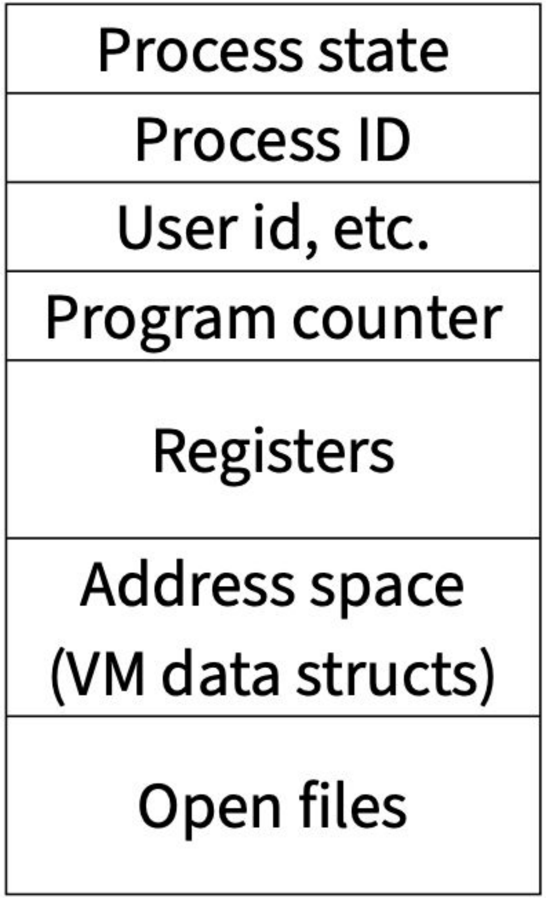
1. Review of Midterm topics
2. General Tips

Midterm Content

- Threads & Processes
- Concurrency
- Scheduling
- Virtual Memory
- Synchronization
- Linking

Process

- Instance of a program running
- Why?
  - Increased CPU utilization
  - Reduced latency
- Process control block (PCB)
  - Stores state, registers, open files, etc
  - Equivalent: struct thread in pintos



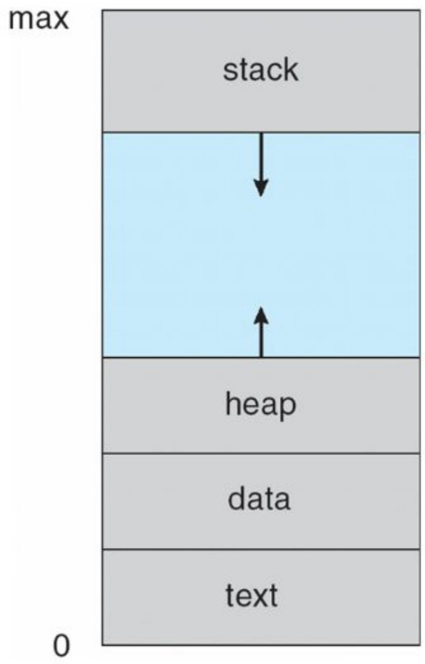
PCB

Processes Cont'd

Each process has its view of the machine

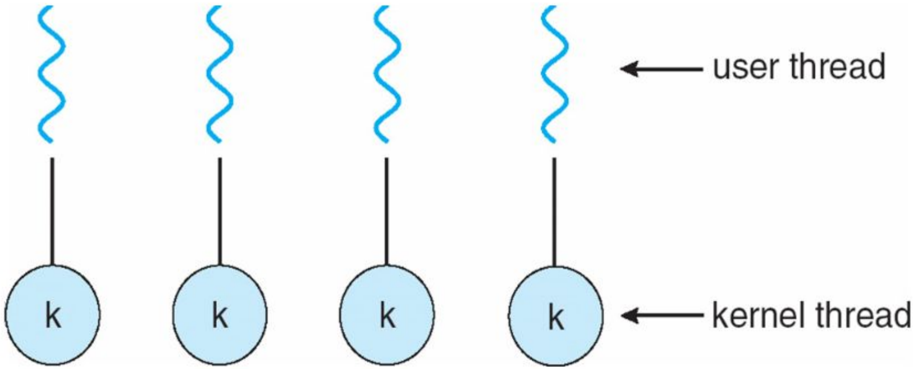
Process interaction can happen through

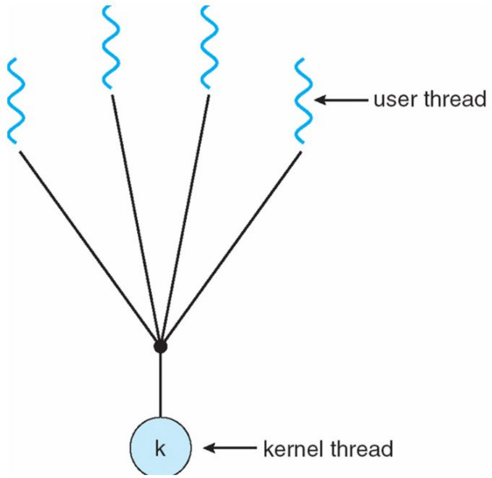
- Through files
- Passing messages through kernel
- Sharing a region of physical memory
- Through asynchronous signals and alerts



Threads

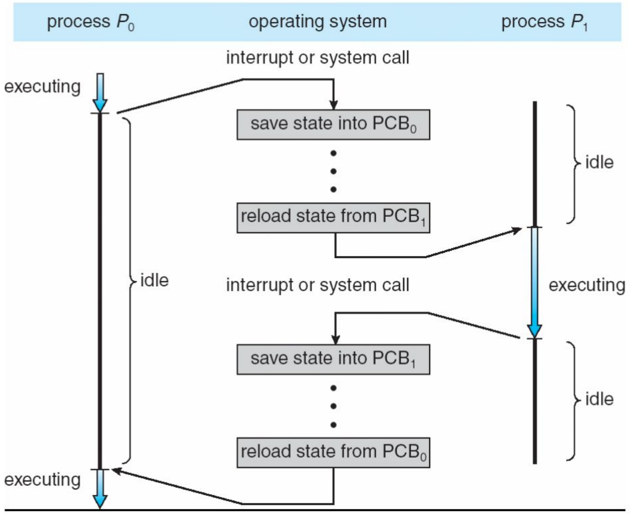
- Schedulable execution context
- Why?
  - Concurrency
  - Multi-core execution
- Kernel threads
  - More scheduling control
  - Heavy weight
  - Everything must go through kernel
- User threads
  - Lightweight and flexible
  - Less control





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



Concurrency

- Data races
- Critical Section
  - 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

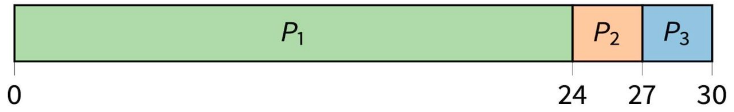
Implementing Synchronization

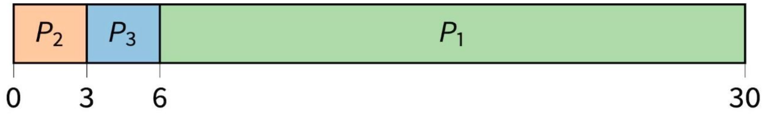
- Disable Interrupts
  - Bad for multiprocessors
  - May be efficient for uniprocessors
- Spinlocks
  - Wastes CPU
- CPU locks memory system around read and write
- Modern OSes design for multiprocessors
  - Need fine-grained locks

Scheduling

- Problem
  - Given n > 1 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 cont'd





- FCFS
    - CPU-bound vs IO-bound jobs
- Shortest job first
    - Unfairness and starvation
- Round-robin
    - Same-sized jobs
- Priority Scheduling
- MLFWS (multilevel feedback queues)

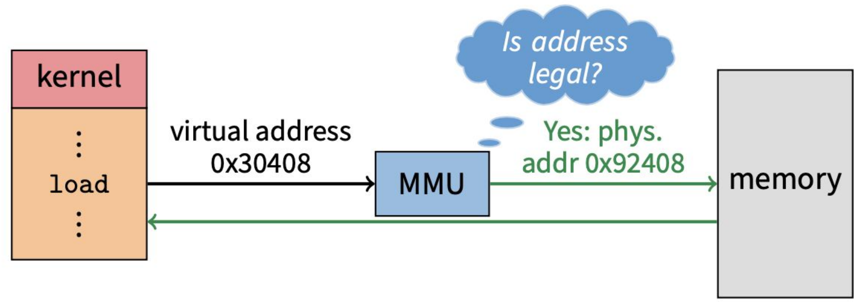
Multiprocessor Scheduling

- 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 processes/threads together

Virtual Memory

How should processes interact with memory?

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

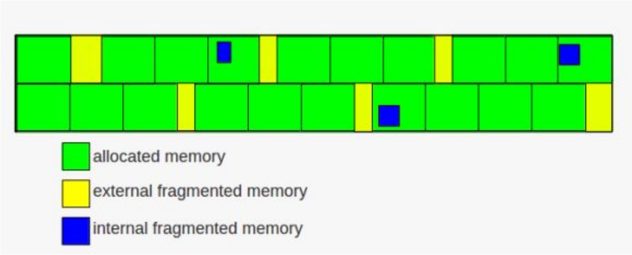


Mapping Memory

- Base + bound
  - Physical address = virtual address + base
- Segmentation
  - Divide memory into segments
- Demand 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

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



Memory System

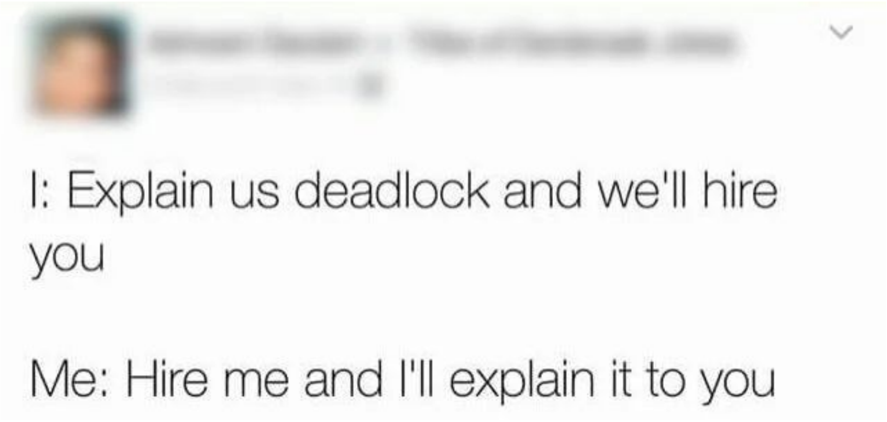
- Coherence
  - Concerns access to single memory location
  - Multiple processes writing to same variable
- Consistency
  - Concerns ordering across multiple memory locations
    - If x=1,y=2, A reads x,y and B writes x=3,y=4, could A ever see x=1,y=4?
  - Sequential consistency matches our intuition

Misc Synchronization

- Multicore cache coherence
  - MESI coherence protocol
- Test and set spinlock
  - Simple, one memory location
  - Lots of traffic over memory interconnect
- Fine-grained locks allow for more parallelism
- Coarse-grained locks are good for global data
- C11 atomics -> direct access to synchronized lower level operations
  - Atomic counters
  - X-Y fence = operations of type X sequenced before the fence happen before operations of type Y sequences after the fence
- Read-copy update
  - Data is read more often than written
  - Relies on dependency ordering in hardware

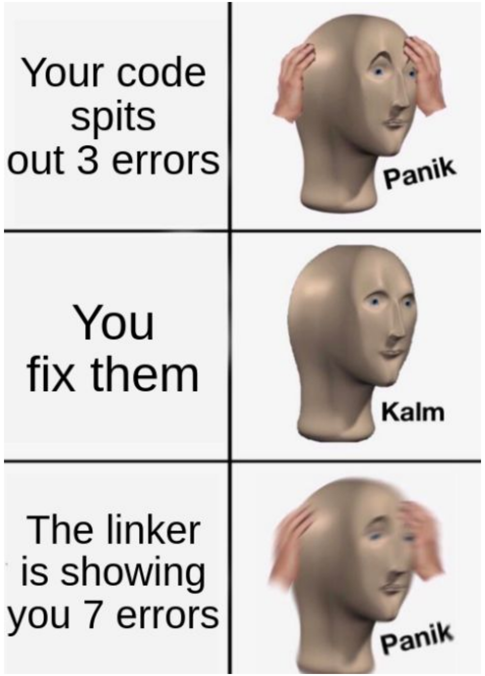
Considerations

- 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



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
        - 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

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
- Rely on notes
    - Might be time-constrained
    - Create a cheat sheet instead of printing all lecture slides (or print both?)
- Deep understanding of most material >> cursory understanding of all
- 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)
- Get a good night’s sleep! You may have to stare at code/memory models/hexadecimals during the exam

Good luck!

(Don’t panic if things go wrong)