CSE 153
Design of Operating Systems

Winter 2018

Midterm Review
Midterm

- in class on Monday

- Covers material through scheduling and deadlock

- Based upon lecture material and modules of the book indicated on the class schedule
  - Closed book. No additional sheets of notes
Overview

- Architectural support for Oses
- Processes
- Threads
- Synchronization
- Scheduling
Arch Support for OSes

- Types of architecture support
  - Manipulating privileged machine state
  - Generating and handling events
Privileged Instructions

- What are privileged instructions?
  - Who gets to execute them?
  - How does the CPU know whether they can be executed?
  - Difference between user and kernel mode

- Why do they need to be privileged?

- What do they manipulate?
  - Protected control registers
  - Memory management
  - I/O devices
Events

- Events
  - Synchronous: faults (exceptions), system calls
  - Asynchronous: interrupts
- What are faults, and how are they handled?
- What are system calls, and how are they handled?
- What are interrupts, and how are they handled?
  - How do I/O devices use interrupts?
- What is the difference between exceptions and interrupts?
Processes

- What is a process?
- What resource does it virtualize?
- What is the difference between a process and a program?
- What is contained in a process?
Process Data Structures

- Process Control Blocks (PCBs)
  - What information does it contain?
  - How is it used in a context switch?

- State queues
  - What are process states?
  - What is the process state graph?
  - When does a process change state?
  - How does the OS use queues to keep track of processes?
Process Manipulation

- What does CreateProcess on Windows do?
- What does fork() on Unix do?
  - What does it mean for it to “return twice”?
- What does exec() on Unix do?
  - How is it different from fork?
- How are fork and exec used to implement shells?
Threads

- What is a thread?
  - What is the difference between a thread and a process?
  - How are they related?
- Why are threads useful?
- What is the difference between user-level and kernel-level threads?
  - What are the advantages/disadvantages of one over another?
Thread Implementation

- How are threads managed by the run-time system?
  - Thread control blocks, thread queues
  - How is this different from process management?

- What operations do threads support?
  - Fork, yield, sleep, etc.
  - What does thread yield do?

- What is a context switch?

- What is the difference between non-preemptive scheduling and preemptive thread scheduling?
  - Voluntary and involuntary context switches
Synchronization

- Why do we need synchronization?
  - Coordinate access to shared data structures
  - Coordinate thread/process execution

- What can happen to shared data structures if synchronization is not used?
  - Race condition
  - Corruption
  - Bank account example

- When are resources shared?
  - Global variables, static objects
  - Heap objects
Mutual Exclusion

- What is mutual exclusion?
- What is a critical section?
  - What guarantees do critical sections provide?
  - What are the requirements of critical sections?
    » Mutual exclusion (safety)
    » Progress (liveness)
    » Bounded waiting (no starvation: liveness)
    » Performance
- How does mutual exclusion relate to critical sections?
- What are the mechanisms for building critical sections?
  - Locks, semaphores, monitors, condition variables
Locks

- What does Acquire do?
- What does Release do?
- What does it mean for Acquire/Release to be atomic?
- How can locks be implemented?
  - Spinlocks
  - Disable/enable interrupts
- How does test-and-set work?
  - What kind of lock does it implement?
- What are the limitations of using spinlocks, interrupts?
  - Inefficient, interrupts turned off too long
Semaphores

● What is a semaphore?
  ◆ What does Wait/P/Decrement do?
  ◆ What does Signal/V/Increment do?
  ◆ How does a semaphore differ from a lock?
  ◆ What is the difference between a binary semaphore and a counting semaphore?

● When do threads block on semaphores?

● When are they woken up again?

● Using semaphores to solve synchronization problems
  ◆ Readers/Writers problem
  ◆ Bounded Buffers problem
Monitors

- What is a monitor?
  - Shared data
  - Procedures
  - Synchronization

- In what way does a monitor provide mutual exclusion?
  - To what extent is it provided?

- How does a monitor differ from a semaphore?

- How does a monitor differ from a lock?

- What kind of support do monitors require?
  - Language, run-time support
Condition Variables

- What is a condition variable used for?
  - Coordinating the execution of threads
  - Not mutual exclusion

- Operations
  - What are the semantics of Wait?
  - What are the semantics of Signal?
  - What are the semantics of Broadcast?

- How are condition variables different from semaphores?
Implementing Monitors

- What does the implementation of a monitor look like?
  - Shared data
  - Procedures
  - A lock for mutual exclusion to procedures (w/ a queue)
  - Queues for the condition variables

- What is the difference between Hoare and Mesa monitors?
  - Semantics of signal (whether the woken up waiter gets to run immediately or not)
  - What are their tradeoffs?
  - What does Java provide?
Scheduling

- What kinds of scheduling is there?
  - Long-term scheduling
  - Short-term scheduling

- Components
  - Scheduler (dispatcher)

- When does scheduling happen?
  - Job changes state (e.g., waiting to running)
  - Interrupt, exception
  - Job creation, termination
Scheduling Goals

- **Goals**
  - Maximize CPU utilization
  - Maximize job throughput
  - Minimize turnaround time
  - Minimize waiting time
  - Minimize response time

- What is the goal of a batch system?
- What is the goal of an interactive system?
Starvation

- Starvation
  - Indefinite denial of a resource (CPU, lock)

- Causes
  - Side effect of scheduling
  - Side effect of synchronization

- Operating systems try to prevent starvation
Scheduling Algorithms

● What are the properties, advantages and disadvantages of the following scheduling algorithms?
  ◆ First Come First Serve (FCFS)/First In First Out (FIFO)
  ◆ Shortest Job First (SJF)
  ◆ Priority
  ◆ Round Robin
  ◆ Multilevel feedback queues

● What scheduling algorithm does Unix use? Why?
Deadlock

- Deadlock happens when processes are waiting on each other and cannot make progress
- What are the conditions for deadlock?
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait
- How to visualize, represent abstractly?
  - Resource allocation graph (RAG)
  - Waits for graph (WFG)
Deadlock Approaches

- Dealing with deadlock
  - Ignore it
  - Prevent it (prevent one of the four conditions)
  - Avoid it (have tight control over resource allocation)
  - Detect and recover from it

- What is the Banker’s algorithm?
  - Which of the four approaches above does it implement?
Lets do some problems
Problem 5: (21 points; 15 minutes)

Explain how you would simulate each of the following; please write the synchronization related pseudocode:

(a) 20 pieces of dominoes that are stacked so that when domino 1 falls, it tips domino 2, and that in turn tips over domino 3, etc...

(b) People trying to get into a restaurant that only has limited seating room

(c) Two players that are playing frisbee with each other
**Problem 3:** (20 pts) Consider a multiple feedback scheduler with three levels. The first level has a quantum of 5ms, the second has a quantum of 10ms, and the third is FCFS. You have a set of processes with a run time of 9ms, 16ms, 4ms, 20ms, and 7ms that arrive at times 0, 3, 6, 8 and 10 ms respectively. What is the average normalized turnaround time for the processes? Show your work, including the state of the queues whenever they change.
Problem 5: (30 pts; 25 minutes) Two or more people are playing a game of frisbee with two or more frisbees. Initially, the frisbees are given to different people and the number of people and frisbees is known. The players stand around in a circle and the frisbees are thrown clockwise along the circle. Each person receives a frisbee and throws it to the next person over. Unfortunately, if a frisbee is thrown at a person when he already has another, he will drop it – we do not want that case to happen.

- (15 points) Implement a pseudocode simulation of this problem (you can use locks, semaphores or condition locks)
- (5 points) Is there a problem if the number of frisbees equals the number of players? Explain
This problem is concerned with the bakery algorithm solution for the critical section problem.

(a) (3 points) Why was the concept of obtaining ticket numbers needed?

(b) (7 bonus) Because the process of obtaining a ticket is inefficient, the following replacement was suggested. Every process keeps track of the number of times it entered the critical section. This number is used in place of the ticket (with the process id used as tie break). Comment on this idea.
Problem 6: (25 pts; 20 minutes) At the coffee shop in the library, people queue up to buy coffee from the register. After placing their order, they go and wait for their coffee. There are three workers, in addition to the person on the cash register, and each works on one order at a time.

(a) Write pseudocode to simulate the coffee shop operation

(b) Since there are many types and flavors of coffee, it's important that everyone picks up their correct order. Assume now that coffees may not be ready in the order that they were placed – for example, regular coffee is ready much faster than a double shot latte. Explain (in words, but be specific) how the implementation could be done now.
Problem 5: (15 pts; 15 minutes) A traffic policeman is in charge of controlling traffic at a busy intersection. Cars arrive from 4 directions (North, East, South and West). The policeman goes through the directions in the order they are listed above (N, E, S, W then back to N). When he starts a new direction, he allows at most 5 cars to go; if there are less than 5 he changes to the next direction after the last car passes through. Show the pseudo-code for the policeman and the cars to simulate this problem.