Overview

- The midterm
- Architectural support for OSes
- OS modules, interfaces, and structures
- Processes
- Threads
- Synchronization
- Scheduling
Midterm

- Covers material through scheduling
- Based upon lecture material, homeworks, and project
- One 8.5”x11” double-sided sheet of notes

Please, do not cheat
  - Do not copy from your neighbor
  - You will be noticed
  - No one involved will be happy, particularly the teaching staff
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: fault (exceptions), system calls
  - Asynchronous: interrupts, software interrupt

- 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?
OS Modules and Interfaces

- Modules
  - OS services and abstractions

- Interfaces
  - Operations supported by components
Modules

- Processes
- Memory
- I/O
- Secondary storage
- Files
- Protection
- Account
- Command interpreter (shell)
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 NT 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
  - Blocking (Project 1)
- 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) for all single-unit resources
  - Maximum claims graph for Avoidance
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?
1. (15 points) You are heading the programming language group of a large Northwestern software company. One of your employees suggests that a new synchronization construct called tagged critical regions is all that one needs for writing multi-threaded programs. The new language construct brackets code with a statement:

```
critical v1, v2, ..., vn: statements; end_critical
```

where v1, v2, ..., vn is a list of strings called tags. Any two tagged critical regions that have at least one tag in common execute in mutual exclusion with each other. For example, consider the three tagged critical regions:

```
critical x, z:
S1;
end_critical
```

```
critical x, y:
S2;
end_critical
```

```
critical y, u, v:
S3;
end_critical
```

The code S1 and S2 execute in mutual exclusion (they both have x as a tag), and S2 and S3 execute in mutual exclusion (they both have y as a tag), but S1 and S3 can execute at the same time (since they don’t share any tags).

(a) Explain how a tagged critical region can be implemented using semaphores. Illustrate your technique with the three tagged critical regions above. Be sure that your solution is both safe and live.

(b) Is your employee correct: are tagged critical regions as powerful as semaphores or monitors? Explain.
(a) Allocate a mutex semaphore for each tag, each with an initial value of one. To enter a tagged critical region, wait on each semaphore in some order (say, lexicographically). To leave a tagged critical region, signal each semaphore in any order. This prevents deadlock because it uses hierarchical locking, e.g., it does not allow cyclical waiting.

```
wait(x); wait(z); S1; signal(x); signal(z);
wait(x); wait(y); S2; signal(x); signal(y);
wait(u); wait(v); wait(y); S3; signal(u); signal(v); signal(y);
```

(b) No, they're not as powerful; mutex semaphores can't be implemented using tagged critical regions. Tagged critical regions only implement mutual exclusion; they can't enforce ordering of operations like semaphores can (how would you implement wait on a semaphore, for example?)
5. (10 points) Some simple-answer questions, each worth a point.

(a) Can the system call instruction be a privileged instruction? 
(b) True or false: a condition variable is a kind of semaphore. 
(c) Does exec always/sometimes/never create a new process? 
(d) True or false: A single process can be deadlocked by itself. 
(e) Invoking the semaphore wait operation always/sometimes/never causes a thread to block? 
(f) True or false: A process can starve in a system with multi-level feedback queue scheduling. 
(g) A cycle in a resource allocation graph always/sometimes/never indicates a deadlock. 
(h) True or false: Many threads can attempt to enter a monitor at the same time. 
(i) Is shortest job first scheduling a kind of priority scheduler? 
(j) True or false: Busy waiting is never a good idea.
5. (10 points) Some simple-answer questions, each worth a point.

(a) Can the system call instruction be a privileged instruction?  yes
(b) True or false: a condition variable is a kind of semaphore.  false
(c) Does exec always/sometimes/never create a new process? never
(d) True or false: A single process can be deadlocked by itself.  true and false (depends on definition)
(e) Invoking the semaphore wait operation always/sometimes/never causes a thread to block? sometimes
(f) True or false: A process can starve in a system with multi-level feedback queue scheduling. yes
(g) A cycle in a resource allocation graph always/sometimes/never indicates a deadlock. sometimes
(h) True or false: Many threads can attempt to enter a monitor at the same time. true
(i) Is shortest job first scheduling a kind of priority scheduler? yes
(j) True or false: Busy waiting is never a good idea. false