CSE 120
Principles of Operating Systems

Spring 2023

Lecture 9: Midterm Review

Amy Ousterhout

Administrivia

- Project 1
  - Due today
  - Make sure to read the submission instructions and try submitting before the deadline
- Homework #2
  - Due today

Today's Outline

- Midterm logistics
- Midterm topics
- Practice problems

Midterm Logistics

- Thursday May 4th, FAH 1450 (this room), 3:30-4:50 pm
- You may bring one 8.5”x11” double-sided sheet of notes to the exam
  - Typed or handwritten
- Bring your ID to show to a proctor when you hand in your exam
- Covers all material so far
- Based on lectures, homework, and programming projects
  - All lectures up through CPU scheduling
  - Homeworks #1 and #2
  - Project 1
- Obligatory: complete your exam yourself without assistance from others
  - Sign an agreement on the first page indicating this

Types of Questions

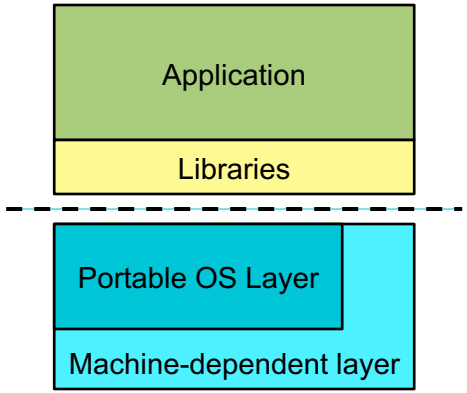
- True or False
- Multiple choice
- Short answer
- Larger problems
  - Including writing code

Today's Outline

- Midterm logistics
- Midterm topics
- Practice problems

Interactions with Apps and Hardware (1/2)

- Dual-mode operation
  - Difference between user and kernel mode
  - What causes a mode switch?
  - What happens during a mode switch?
- Privileged instructions
  - What types of instructions should be privileged?
  - When can they be executed?
- Which OS tasks rely on hardware support?

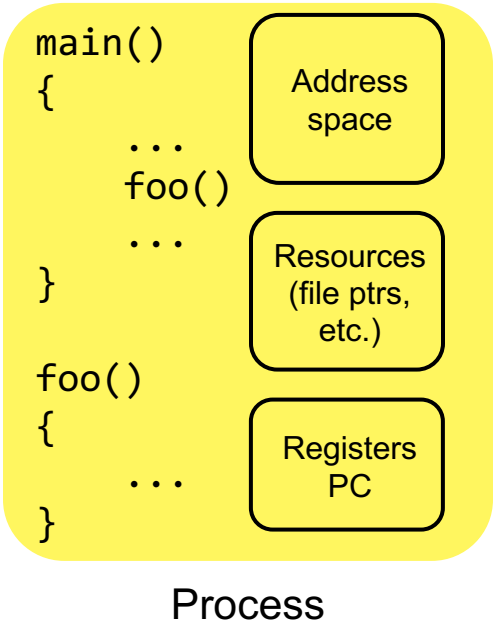


Interactions with Apps and Hardware (2/2)

- Events
  - What happens during an event?
  - What are the different types and the differences between them?
- Exceptions
  - What is a fault? How is a fault handled?
  - What is a system call? How are they handled?
- Interrupts
  - What is an interrupt? How is an interrupt handled?

Processes (1/2)

- Process abstraction
  - What is a process?
  - What is the difference between a process and a program?
  - What resources does a process manage?
- Process Control Blocks (PCBs)
  - What information does a PCB contain?
  - How is it used in a context switch?
- Process state
  - What are process states?
  - How does the OS use queues to keep track of processes?

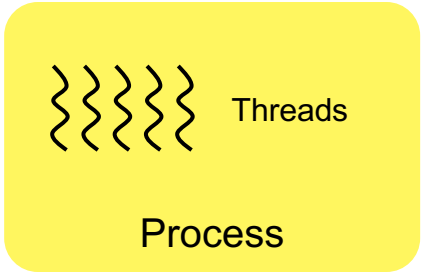


Processes (2/2)

- Process APIs
  - 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 are `fork` and `exec` used to implement shells?

Threads (1/2)

- What is a thread?
  - What is the difference between a thread and a process?
  - How are they related?
- Why are threads useful?
- How are threads managed?
  - Thread control blocks, thread queues
- User-level vs. kernel-level threads
  - What are the trade offs between them?
  - How are they managed?
  - What is M:N threading?

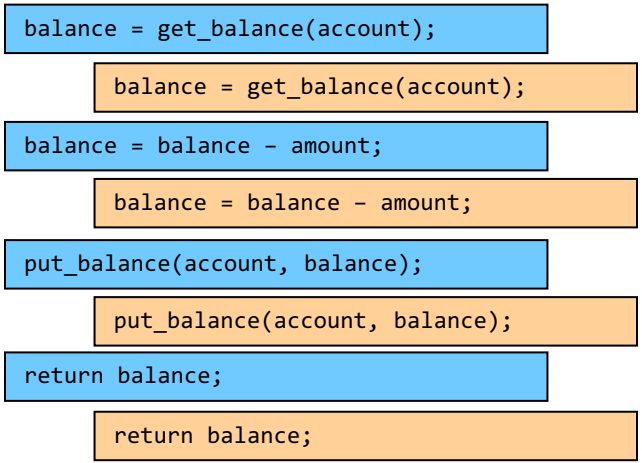


Threads (2/2)

- Thread scheduling
  - What is a context switch?
  - What do `sleep`, `yield`, `finish`, `join`, etc. do?
  - What is the difference between preemptive and non-preemptive scheduling?

Synchronization

- Why do we need synchronization?
  - Coordinate access to shared resources
  - Coordinate thread/process execution
- What resources are shared?
  - Global variables, static objects, heap objects
  - Not shared: local variables, stacks
- What can happen to shared data structures if synchronization is not used?
  - Race condition
  - Corruption
  - Bank account example



Mutual Exclusion

- What is mutual exclusion?
- What is a critical section?
  - What are the requirements of critical sections?
    - Mutual exclusion (safety)
    - Progress (liveness)
    - Bounded waiting (no starvation: liveness)
    - Performance
- How are mutual exclusion and critical sections related?

Locks

- What do `acquire` and `release` do?
- What does it mean for `acquire/release` to be atomic?
- How can we implement locks?
  - Spinlocks
  - Disable/enable interrupts
  - Blocking using a queue
- How does `test-and-set` work?
- What are the downsides of using spinlocks or disabling interrupts?

```
acquire(lock)
…
Critical section
…
release(lock)
```

Semaphores

- What is a semaphore?
  - What does `wait/P` do?
  - What does `signal/V` 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?
- How can you use semaphores to solve synchronization problems?
  - How many?
  - How to initialize them?
  - Where to call `wait/signal`?
  - Where is the critical section?

Condition Variables

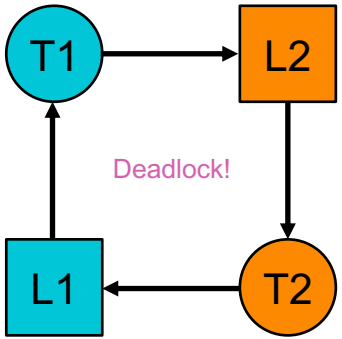
- What is a condition variable used for?
  - Coordinating the execution of threads
  - Not mutual exclusion
- Operations
  - What are the semantics of `wait/sleep`?
  - What are the semantics of `signal/wake`?
  - What are the semantics of `broadcast/wakeAll`?
- How are condition variables different from semaphores or locks?

Monitors

- What is a monitor?
- What guarantees does it provide?
- What are the benefits of using a monitor?

Deadlock (1/2)

- When does deadlock happen?
  - Threads 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 can we visualize deadlock?
  - Resource allocation graphs

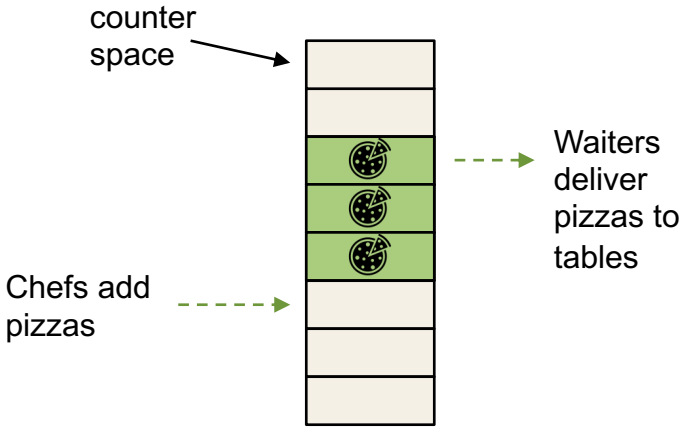


Deadlock (2/2)

- How can we deal with deadlock?
  - Ignore it
  - Prevent it (prevent one of the four conditions)
  - Avoid it (maintain tight control over resource allocation)
  - Detect it and recover from it

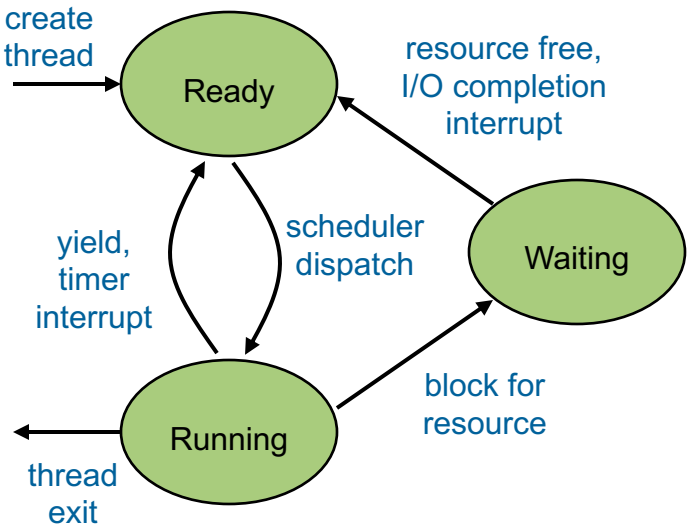
Synchronization Problems

- What are common synchronization problems?
  - Readers-writers problem
  - Producer-consumer problem
  - Dining philosophers problem



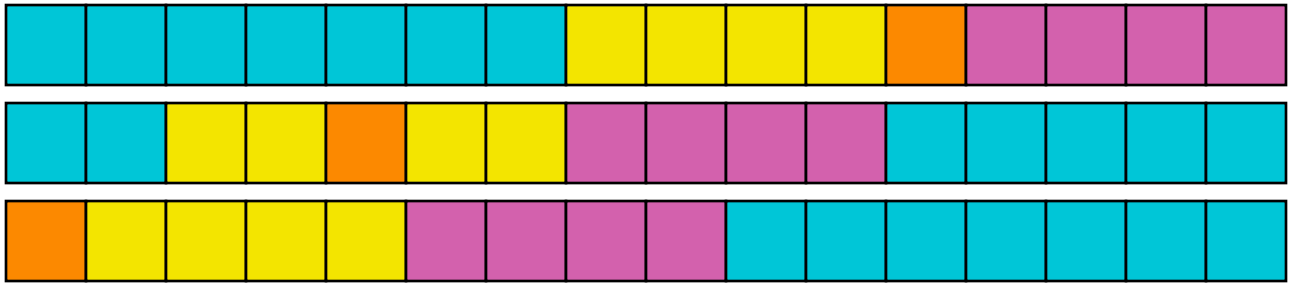
Scheduling

- When does scheduling occur?
- What are the possible states and what causes transitions between them?
- What are possible goals with scheduling?
  - Maximize CPU utilization
  - Maximize job throughput
  - Minimize turnaround time
  - Minimize response time
- What kinds of applications have each goal?
- What is starvation and what causes it?



Scheduling Algorithms/Policies

- How do each of the following algorithms work and what are their pros/cons?
  - First-come first-served (FCFS) / First-in first-out (FIFO)
  - Shortest job first (SJF)
  - Shortest remaining time to completion first (SRTCF)
  - Round robin
  - Priority scheduling
  - Multi-level feedback queues (MLFQ)



Today's Outline

- Midterm logistics
- Midterm topics
- Practice problems

Privileged Instructions

- Which of the following instructions are privileged?
  - ADD – add two numbers ✗
  - CPUID – returns information about the kind of CPU you're running on and what features it supports ✗
  - INVD – invalidates caches without writing data to memory ✓
  - INT – initiates a system call ✗
  - RET – return from a function ✗ in the common case

Race Conditions

- Suppose one thread calls `AddToX` once and another calls `SubFromX` once
- What are the possible values of x after both threads have completed?

```
int x = 0;
int i, j;

void AddToX() {
  for (i = 0; i < 10; i++)
    x++;
}

void SubFromX() {
  for (j = 0; j < 10; j++)
    x--;
}
```

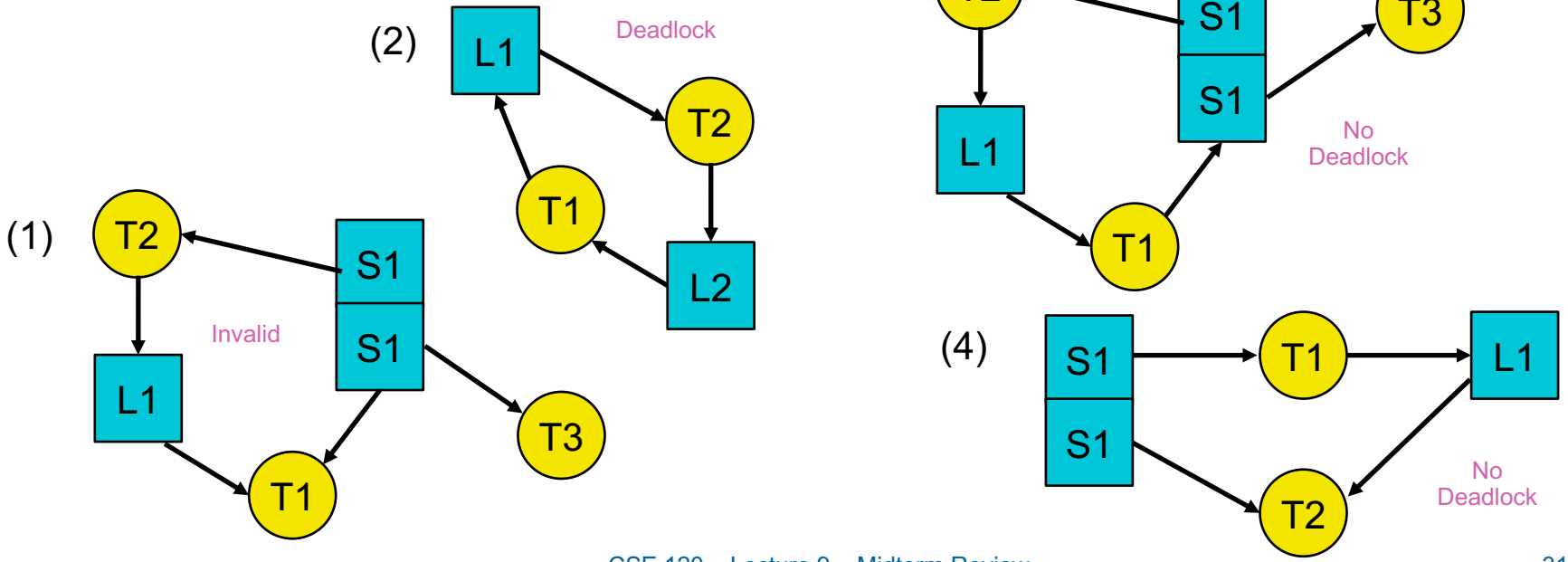
Anywhere from
-10 to 10

Scheduling

- What can cause each of the following state transitions? (some may not be possible)
  - Ready -> running — scheduler runs (i.e., context switch)
  - Ready -> waiting — not possible
  - Running -> ready — yield or timer interrupt
  - Running -> waiting — block for resource
  - Waiting -> running — not possible

Deadlock

- For each of the following does it show deadlock, no deadlock, or an invalid resource allocation graph?



Crowded Lab

- Goal: at most 8 people in the lab at once
- Write out a solution using synchronization primitives
  - Include the `init`, `enter_lab`, and `leave_lab` functions

```
init () {
  s = new Semaphore(8);
}

enter_lab () {
  s.wait();
}

leave_lab () {
  s.signal();
}
```

This is a simple example of how to use a semaphore to manage a pool of shared resources.

Hikers-Bikers Problem

- You have been hired to manage a local trail
  - Hikers and bikers don't share the trail well
  - Too many bikers damage the trail
- Goals:
  - Do not allow hikers and bikers on the trail at the same time
  - At most 2 bikers on the trail at once
- Write out a solution using locks and condition variables
  - Locks: `acquire, release`
  - Condition variables: `sleep, wake, wakeAll`

Hikers-Bikers Problem

- There can be multiple ways to solve the same problem. Here is one:

Initialization

```
lock = new Lock();
cv = new Condition(lock);
bikers = 0;
hikers = 0;
```

Hiker

```
void hiker(void) {
  acquire(lock);
  while (bikers > 0)
    wait(cv);
  hikers++;
  release(lock);

  // go hiking

  acquire(lock);
  hikers--;
  broadcast(cv);
  release(lock);
}
```

should we check if this is the last hiker? (if hikers == 0)

Biker

```
void biker(void) {
  acquire(lock);
  while (bikers > 1 || hikers > 0)
    wait(cv);
  bikers++;
  release(lock);

  // go biking

  acquire(lock);
  bikers--;
  broadcast(cv);
  release(lock);
}
```

should we check the number of bikers first?

what if we used signal here instead?

Any other questions?