

# CSE 120

# Operating Systems Principles

Spring 2025

Midterm Review

Amy Ousterhout

# Today's Outline

---

- Midterm logistics
- Midterm topics
- Practice problems

# Midterm Logistics

---

- Thursday May 1<sup>st</sup>, York 2722 (the usual lecture hall), 3:30-4:50 pm
- You may bring **one 8.5"x11" double-sided sheet of notes** to the exam
  - Handwritten or printed
- **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
- Multi-part problems
  - Including reading and writing code
  - See questions 4 and 5 in the sample midterm for examples

# Today's Outline

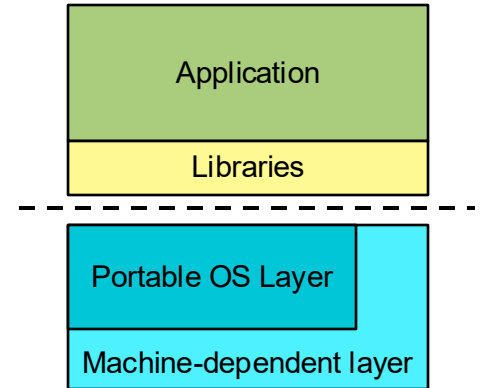
---

- 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?



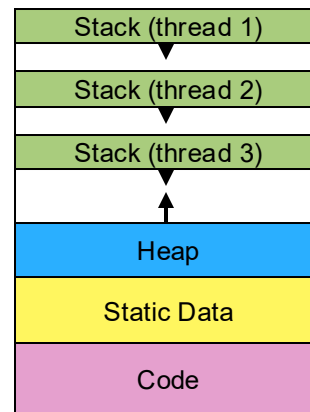
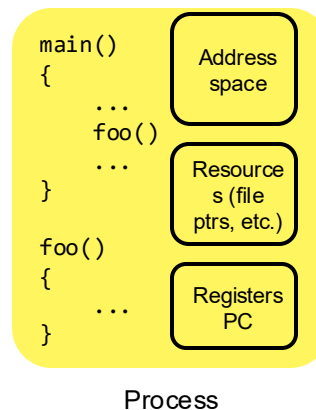
# Interactions with Apps and Hardware (2/2)

---

- 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?
- Running processes
  - What are process states?
  - What is the processing illusion?



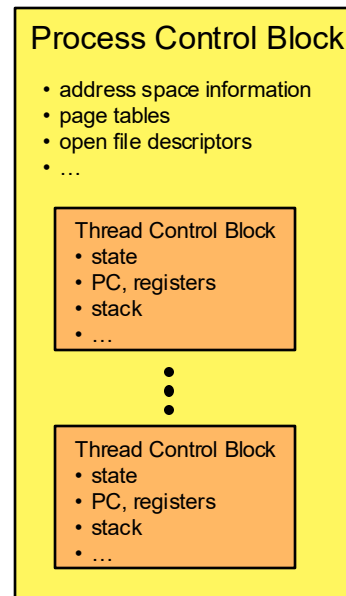
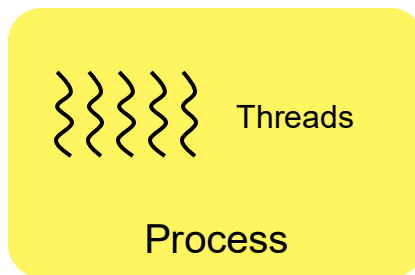
# Processes (2/2)

---

- 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?

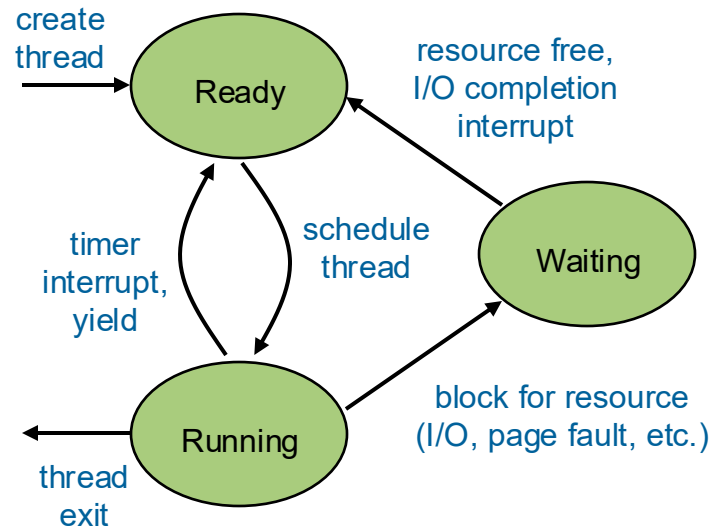
# Threads (1/2)

- 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



# Threads (2/2)

- User-level vs. kernel-level threads
  - What are the trade offs between them?
  - How are they managed?
  - What is M:N threading?
- Thread scheduling
  - What is a context switch?
  - What causes threads to change state?
  - 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

```
balance = get_balance(account);
```

```
balance = get_balance(account);
```

```
balance = balance - amount;
```

```
balance = balance - amount;
```

```
put_balance(account, balance);
```

```
put_balance(account, balance);
```

```
return balance;
```

```
return balance;
```

# 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)
```

Implementing a lock by disabling interrupts

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

} Interrupts disabled  
} Interrupts enabled  
} Interrupts disabled

# Semaphores

---

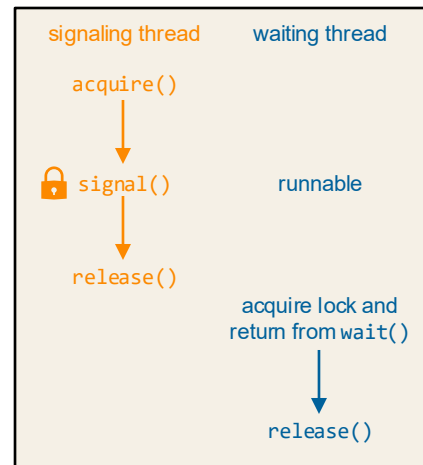
- What is a semaphore?
  - What does `wait/P` do?
  - What does `signal/V` do?
  - How does a semaphore differ from a lock?
  - What is the difference between a binary semaphore and a counting semaphore?
- When do threads block on semaphores? When are they woken up again?
- How can you use semaphores to solve synchronization problems?
  - How many?
  - How to initialize them?
  - Where to call `wait/signal`?
  - Where is the critical section?

```
wait(s) {  
    if (s <= 0)  
        sleep();  
    s--;  
}
```

```
signal(s) {  
    if (queued thread)  
        wakeup();  
    s++;  
}
```

# 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?
- What is the difference between Mesa and Hoare semantics?



# Monitors

---

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

```
Monitor producer_consumer {
    Condition not_full;
    Condition not_empty;

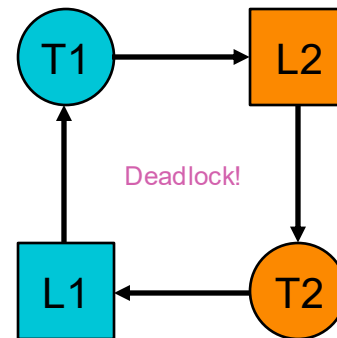
    void put_resource() {
        ...
        wait(not_full);
        ...
        signal(not_empty);
    }

    void get_resource() {
        ...
        wait(not_empty);
        ...
        signal(not_full);
    }
}
```

# 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



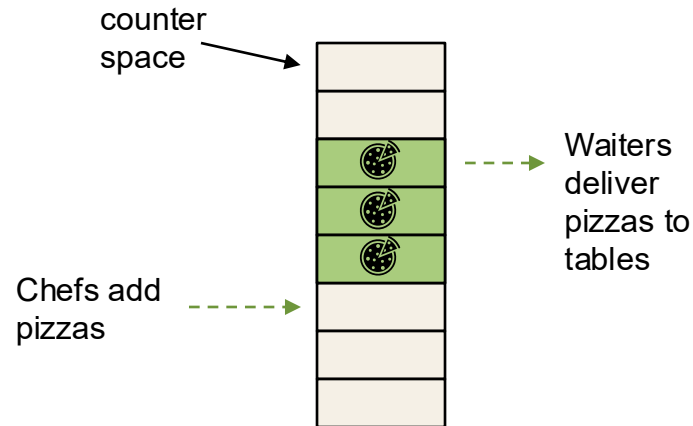
# Deadlock (2/2)

---

- 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

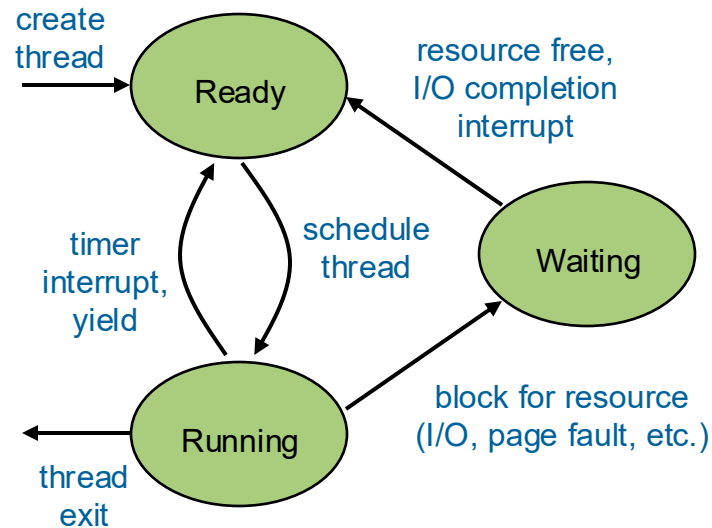
# Synchronization Problems

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



# Scheduling

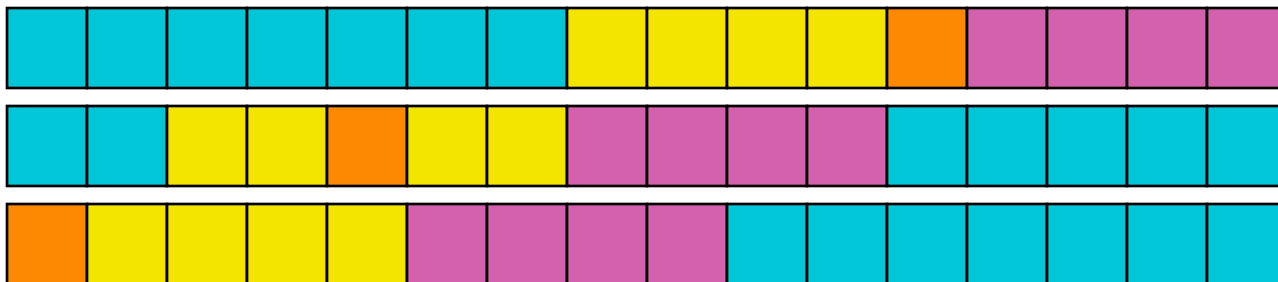
- When does scheduling occur?
- What are the possible states and what causes transitions between them?
- What are possible goals with scheduling?
  - Maximize CPU utilization
  - Maximize job throughput
  - Minimize turnaround time
  - Minimize response time
- What kinds of applications have each goal?
- 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
  - I'll give you 5-10 minutes to work on each one, then we'll discuss
  - Try them on your own first, then feel free to discuss with a neighbor
  - Feel free to work ahead (see slides on the course website)
  - I will walk around and answer questions

# Privileged Operations

---

- Which of the following operations should be privileged?
  - Executing I/O instructions (e.g., send a read operation to a disk) ✓
  - Invoking a system call ✗
  - Modifying memory protection information ✓
  - Destroying a process ✓
  - Reading from memory (e.g., with the MOV instruction) ✗
  - Disabling interrupts ✓
  - Calling `join()` in a user-level thread ✗

# 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;

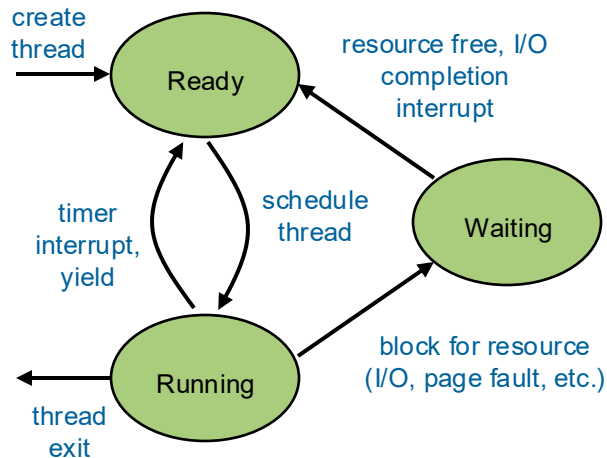
void AddToX() {
    for (i = 0; i < 10; i++)
        x++;
}

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

Anywhere from  
-10 to 10

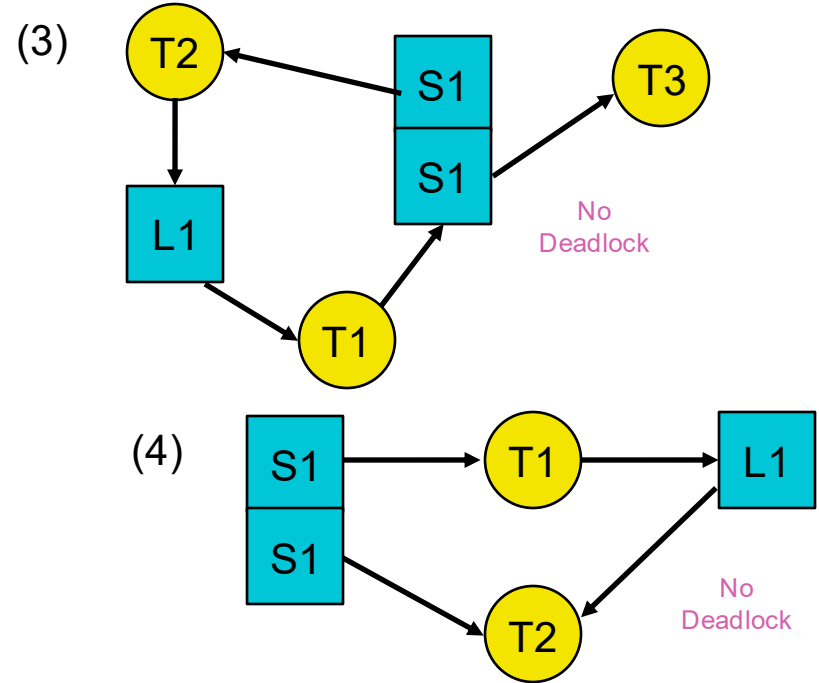
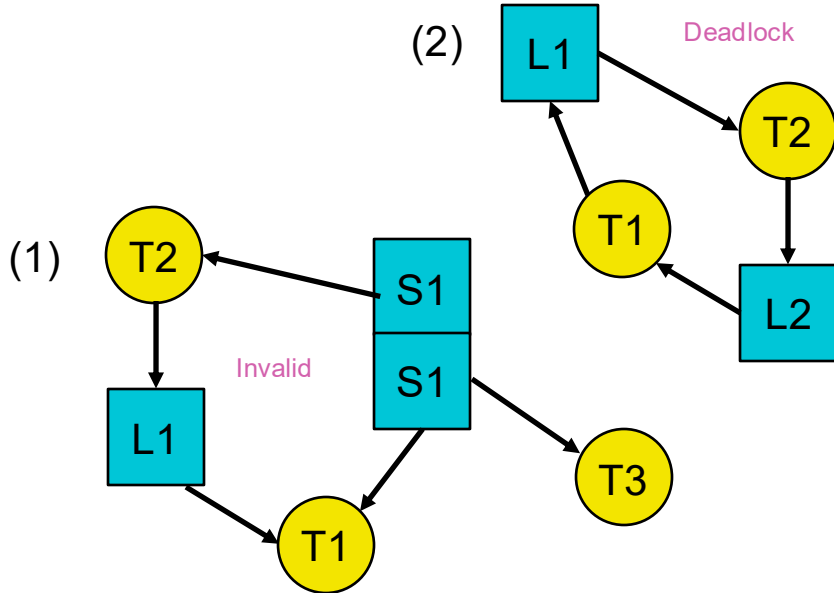
# Scheduling

- What can cause each of the following state transitions? If a transition is not possible, why is it not 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



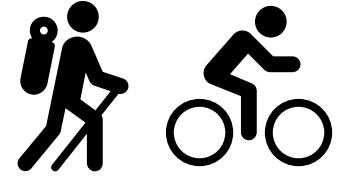
# Deadlock

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



# 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 one lock and one condition variable
  - Lock operations: acquire, release
  - Condition variable operations: sleep, wake, wakeAll



```
void hiker(void) {  
    ???  
  
    // go hiking  
  
    ???  
}
```

```
void biker(void) {  
    ???  
  
    // go biking  
  
    ???  
}
```

# 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?



# Make Me a Pizza

- Three chefs are working together to assemble many pizzas:
  - Chef 1 makes the crust and prints “crust i” when it makes crust i
  - Chef 2 adds sauce and toppings and prints “toppings i” for pizza i
  - Chef 3 cooks the pizza and prints “cooking i” for pizza i
- For each pizza i, Chef 2 can’t start working on it until Chef 1 is done with i, and similar for Chefs 2 and 3
- Use two semaphores to coordinate the execution of these chefs
- Write out code for three functions:



Example output:  
crust 1  
toppings 1  
crust 2  
cooking 1  
toppings 2  
cooking 2

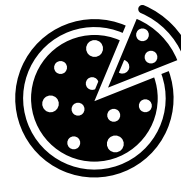
```
// make crusts
void chef1(n) {
    for i in 1 to n:
        ???
}
```

```
// add sauce/toppings
void chef2(n) {
    for i in 1 to n:
        ???
}
```

```
// cook pizzas
void chef3(n) {
    for i in 1 to n:
        ???
}
```

# Make Me a Pizza

- Approach: use one semaphore to represent pizzas that are ready for Chef 2 and one to represent pizzas that are ready for Chef 3



## Initialization

```
crust_done = new Semaphore(0);
toppings_done = new Semaphore(0);
```

- How many pizzas can be in progress at once?
- Should we acquire a lock before/after calling print?

## Chef 1

```
// make crusts
void chef1(n) {
  for i in 1 to n:
    print("crust ", i)
    crust_done.signal()
}
```

## Chef 2

```
// add sauce/toppings
void chef2(n) {
  for i in 1 to n:
    crust_done.wait()
    print("toppings ", i)
    toppings_done.signal()
}
```

## Chef 3

```
// cook pizzas
void chef3(n) {
  for i in 1 to n:
    toppings_done.wait()
    print("cooking ", i)
}
```

# Upcoming Tasks

---

- Homework 2
  - Due Friday 4/25 (today) at 11:59 pm
- Project 1
  - Due Tuesday 4/29 at 11:59 pm
  - Read the submission instructions carefully and submit at least once before the deadline
- Midterm
  - In class on 5/1
- Next week:
  - Tuesday 4/29: no class
  - Tuesday 4/29: virtual office hours
    - » 4:00-5:00 pm
  - Wednesday 4/30: no office hours
  - Thursday 5/1: office hours and midterm
  - See Piazza for details