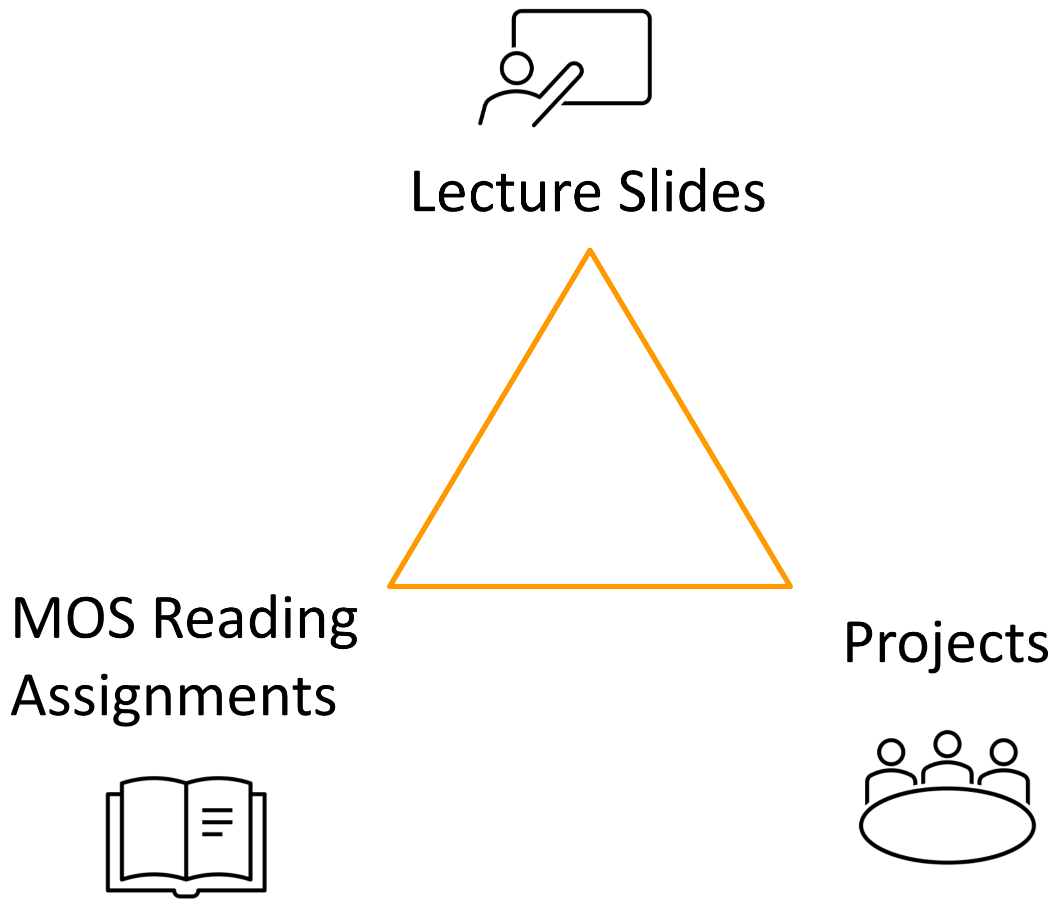
COS 318: Operating Systems

Review Session 1

Outline

- Elaborate on the scope of the final exam
- Practice 8 sample questions

The Scope of The Final Exam



Exam Scope: Lecture Slides

- Cover all lectures.

- Make sure you
  - understand concepts, and
  - reason through code snippets, algorithms, and examples

- In case you have difficulties with a topic and need to watch a lecture recording, feel free to email Dr. Shahrad.

Exam Scope: MOS Reading Assignments

- You were expected to complete those readings before lectures.
- Make sure you cover them fully for the final exam.
- Those three papers not assessed in the final exam.

You are expected to complete the readings **before** the corresponding lecture.

Date	Topic	Reading
8/31	Introduction	MOS 1.1-1.3
9/2	Overview	MOS 1.4-1.5
9/7	Protection and Virtual Memory	MOS 1.6-1.7, 3.1-3.3
9/9	Processes and Threads	MOS 2.1, 2.2.1-2.2.3
9/14	Threads Implementation	MOS 2.2.4-2.2.9
9/16	Synchronization: Mutual Exclusion	MOS 2.3.3, 2.3.6
9/21	Synchronization: Semaphores, Monitors, and Condition Variables	MOS 2.3.5, 2.3.7, Birrell's paper
9/23	CPU Scheduling	MOS 2.4
9/28	Message Passing	MOS 2.3.8, 8.2.1-8.2.4
9/30	Deadlock	MOS 6
10/5	Virtual Memory Address Translation	MOS 3.1-3.3
10/7	Virtual Memory Paging and Caching	MOS 3.4-3.6, 10.4, 11.5
10/14	I/O Devices and Drivers	MOS 5.1-5.3, 5.5-5.9
10/19	Storage Devices	MOS 5.4
10/21	Storage Devices (Cont.)	
10/26	File Structure	MOS 4.2, 4.3.1-4.3.3, 4.5.2-4.5.3
10/28	File Systems: Networked, Abstractions, and Protection	MOS 10.6.3-10.6.4, NetApp paper
11/2	File Caching and Reliability	MOS 4.1, 9.3.1-9.3.3
11/4	File Caching and Reliability (Cont.)	
11/9	Virtual Machine Monitors	MOS 7.3, 7.4.1, 7.6, 7.7, Virtual Machine Monitors paper
11/11	InterNetworking	

Exam Scope: Course Projects

- Only the first five projects: P1, P2, P3, P4, and P5

- If your teammates did all the work, you might have difficulties here.
  - Make sure to fully understand how projects worked.

- Read precept slides, watch precept recordings, and review design review questions.

Sample Question 1

- **What needs to be saved and restored on a context switch between two threads in the same process?**
  - We need to save the registers, stack pointer, and program counter into the thread control block (TCB) of the thread that is no longer running. Then, we need to reload the registers, stack pointer, and program counter from the TCB of the new thread.

- The answer above is brief, explicit, and accurate.

Sample Question 2

- **Which of the following typically require assistance from hardware to implement well? For those that do, circle them, and name the necessary hardware support and its purpose. For those that don't briefly explain why (one or two sentences at most).**
  - System call
  - Thread creation
  - Process context switch

Hardware assistance for system call?

- Requires system support
- In particular to set CPU's protection bit to run in the kernel mode

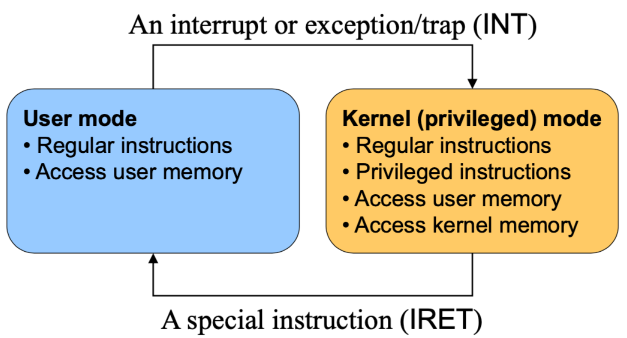
Some Protection Goals

- CPU
  - Allow kernel to take CPU away to prevent a user from using CPU forever
  - Users should not have this ability
- Memory
  - Prevent a user from accessing others' data
  - Prevent users from modifying kernel code and data structures
- I/O
  - Prevent users from performing "illegal" I/Os
- Difference between protection and security?

3

Architecture Support for CPU Protection

- **Privileged Mode**



4

Lecture 3

Hardware assistance for thread creation?

- Creating kernel threads require system calls and hence need a protection bit. Therefore need hardware support.
- User-level threads do not need assistance from hardware.

You might encounter such questions in the exam. Support your answer with a clear argument and consider cases not stated in the question.

HW assistance for process context switch?

- Timer interrupt used in preemptive scheduler

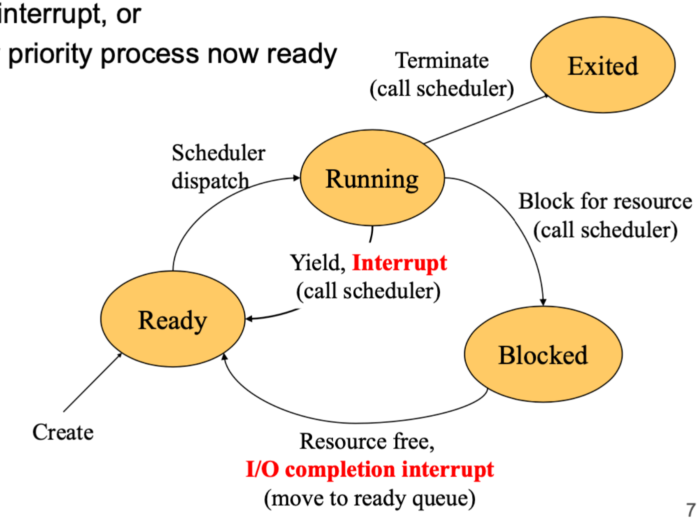
When to schedule?

1. New process created
   - `fork()` → child process created
   - Schedule parent or child (or both)
2. Process dies and returns exit status
   - Due to calling `exit()`, or fatal exception/signal
3. Blocked process
   - E.g. on I/O and semaphore
4. I/O interrupt
5. HW clock interrupt
   - E.g., with 250 Hz frequency
   - **Preemptive** scheduler uses this to replace running processes

5

Preemptive and Non-Preemptive Scheduling

- Preemption happens due to:
  1. Timer interrupt, or
  2. Higher priority process now ready



7

Sample Question 3

Consider a FAT-based (File Allocation Table) file system. Entries in the table are 16 bits wide. A user wants to install a disk with 131072 512-byte sectors.

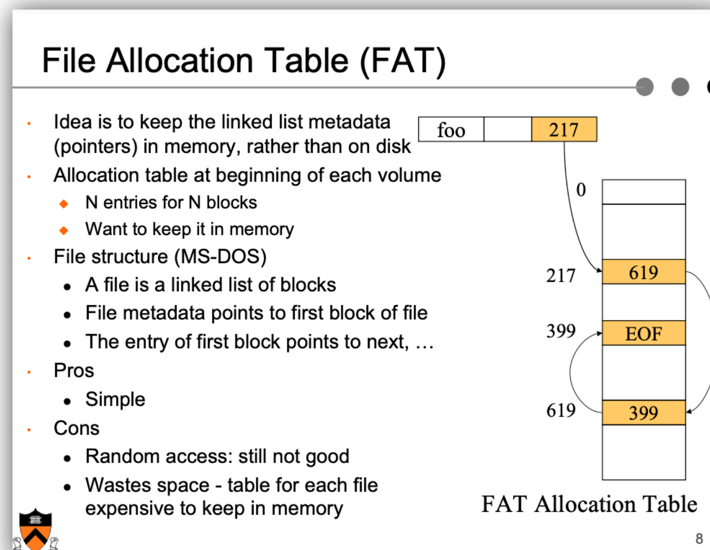
a. What is a potential problem?

b. Describe a solution to this problem and explain the trade-offs involved.

Attribution: from UC Berkeley CS162 Fall 2011 Final Exam

A. Answer:
   - Each entry is a disk sector address
   - 16 bits allows only 65,536 sectors

B. Solution:
   - Make each FAT entry access a logical sector that is 2 physical sectors
   - Trade-off:
     - Con: increased internal fragmentation
     - Pros: maintaining the size of the FAT, and backward compatibility



Sample Question 4

In contrast to a cooperative scheduler, a preemptive scheduler supports the following state transition:

(a) Ready → running

(b) Running → ready

(c) Ready → blocked

(d) Blocked → running

Attribution: from Rutgers CS 416 Spring 2011 Final Exam Review

- Answer: Running → ready

Be prepared to encounter multiple-choice questions in the exam.

Sample Question 5

Threads that are part of the same process share the same stack.

Threads that are part of the same process can access the same TLB entries.

With kernel-level threads, multiple threads from the same process can be scheduled on multiple CPUs simultaneously.

Attribution: from U Wisc-Madison CS 537 Fall 2016 Midterm Exam

Sample Question 5

Threads that are part of the same process share the same stack.

False – each thread has its own stack (specifically its own stack and frame pointer) although the stacks are placed in the same address space.

Threads that are part of the same process can access the same TLB entries.

With kernel-level threads, multiple threads from the same process can be scheduled on multiple CPUs simultaneously.

Attribution: from U Wisc-Madison CS 537 Fall 2016 Midterm Exam

Sample Question 5

Threads that are part of the same process share the same stack.

False – each thread has its own stack (specifically its own stack and frame pointer) although the stacks are placed in the same address space.

Threads that are part of the same process can access the same TLB entries.

True – since they share an address space, they have the same vpn->ppn translations and the same TLB entries are valid.

With kernel-level threads, multiple threads from the same process can be scheduled on multiple CPUs simultaneously.

Attribution: from U Wisc-Madison CS 537 Fall 2016 Midterm Exam

Sample Question 5

Threads that are part of the same process share the same stack.

False – each thread has its own stack (specifically its own stack and frame pointer) although the stacks are placed in the same address space.

Threads that are part of the same process can access the same TLB entries.

True – since they share an address space, they have the same vpn->ppn translations and the same TLB entries are valid.

With kernel-level threads, multiple threads from the same process can be scheduled on multiple CPUs simultaneously.

True – this is the benefit of kernel-level threads (true thread support from OS); we could not do this with user-level threads.

Attribution: from U Wisc-Madison CS 537 Fall 2016 Midterm Exam

Be prepared to encounter true/false questions in the exam.

Sample Question 6

if the semaphore operations *Wait* and *Signal* are not executed atomically, then mutual exclusion may be violated. Assume that *Wait* and *Signal* are implemented as below:

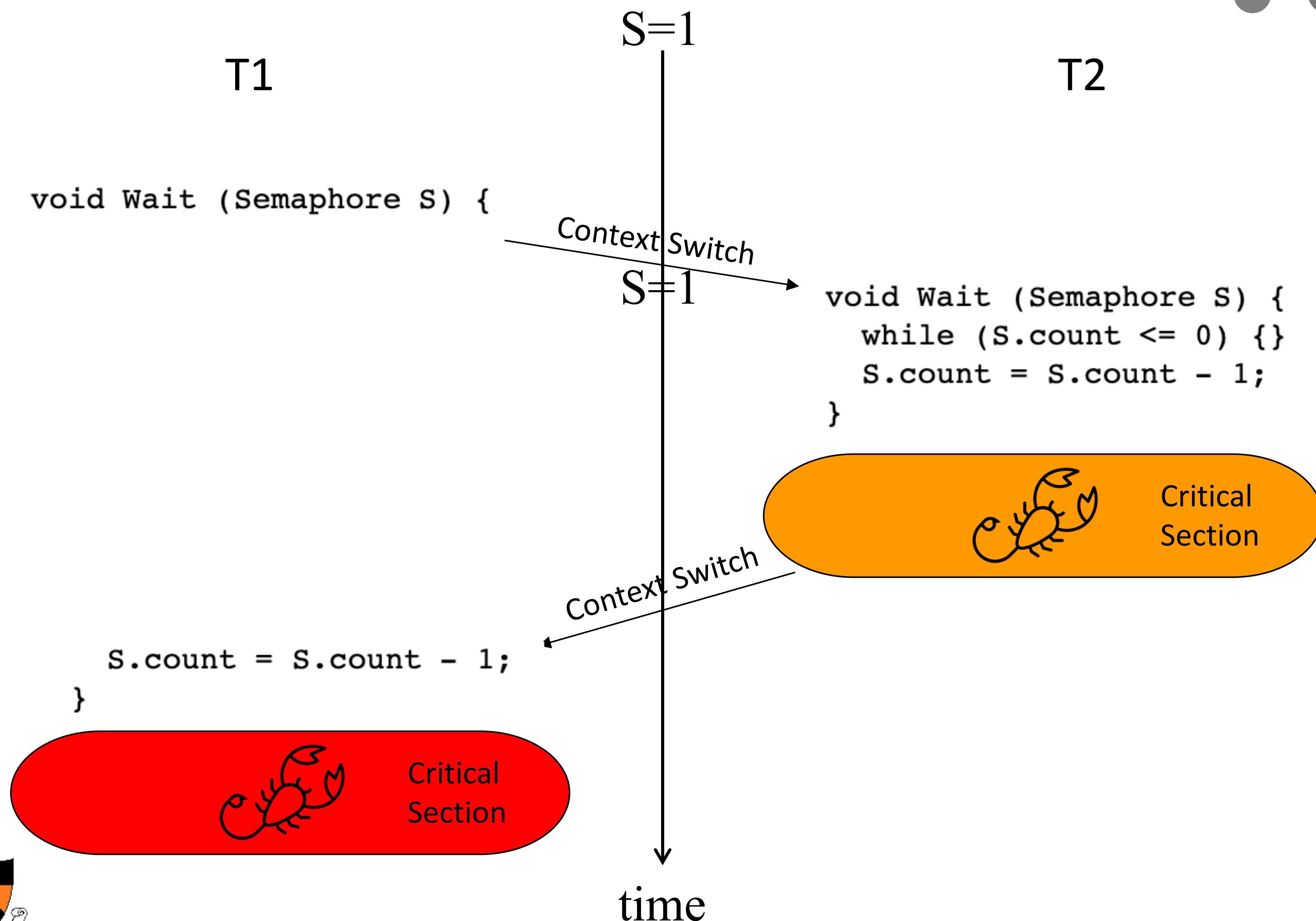
```
void Wait (Semaphore S) {
  while (S.count <= 0) {}
  S.count = S.count - 1;
}

void Signal (Semaphore S) {
  S.count = S.count + 1;
}
```

Describe a scenario of context switches where two threads, T1 and T2, can both enter a critical section guarded by a single mutex semaphore as a result of a lack of atomicity.

Attribution: from UCSD CSE 120 Fall 2016 Final Exam

Sample Question 6



Sample Question 7

Attribution: from U Toronto ECE 344 2018 Midterm Exam

(4) (9 marks) Which of the following operations will trigger the CPU to transition from user mode to kernel mode (i.e., it is an event)? If it is an event, further classify it as either exception or interrupt.

- A process divides an integer by zero

(A) Triggers an exception (B) Triggers an interrupt (C) Not an event

- A process accesses memory at address 0x00000000

(A) Triggers an exception (B) Triggers an interrupt (C) Not an event

- User presses key "a" on the keyboard when using shell

(A) Triggers an exception (B) Triggers an interrupt (C) Not an event

- A process executes test-and-set instruction

(A) Triggers an exception (B) Triggers an interrupt (C) Not an event

- The email client receives an email

(A) Triggers an exception (B) Triggers an interrupt (C) Not an event

Sample Question 8

FIFO

Attribution: from NYU CS 372 Spring 2010 Final Exam

6. [2 points] Consider a machine with 32 MB of RAM running an operating system with virtual memory and swapping. The OS's page replacement policy is: if, on a page fault, a process needs a new physical page in RAM, evict the page that has been in RAM in the *longest*, and write it to the disk if it is dirty. The machine owner notices that for some workloads, the operating system does a lot of disk writes, and the owner is unhappy about that. In response, the owner installs an extra 8 MB of RAM, and re-runs the workload.

Circle True or False for each item below:

True / False There are workloads for which the extra RAM will *decrease* the number of page faults.

True / False There are workloads for which the extra RAM will *have no effect on* the number of page faults.

True / False There are workloads for which the extra RAM will *increase* the number of page faults.

- Example: working set is 38MB
- Looped access to a 400MB working set, every access causes a PF.
- Belady's Anomaly

Belady’s Anomaly (FIFO case)

Access Sequence →

Fits 3 pages

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest				0	1	2	3	0	0	0	1	4	4
		PF	PF	PF	PF	PF	PF	PF			PF	PF	

9 Page Faults

Fits 4 pages

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	2	2	3	4	0	1	2	3
Oldest				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		PF	PF	PF	PF			PF	PF	PF	PF	PF	PF

10 Page Faults

Summary

- Cover all lectures.
- Study MOS reading assignments.
- Make sure you are fully familiar with course projects.

Best of luck with Project 5!