Operating Systems Section 001, Midterm Exam Study Guide

Examples and Concepts from Quizzes, Programming Exercises, and Homework Assignments

OS and Architecture Overview, Chapter 1 and Chapter 12

- Explain the advantages and disadvantages of the operating system as a software layer separating user programs from the hardware.
- Describe the role of system programs, how the operating system provides an interface to end users.
- Describe the general technique of caching, temporal and spatial locality, and how write-through and write-back policies work to keep the backing store consistent with the cache.
- Describe the different types of storage available and the relative size/performance tradeoffs.
- Explain the operation of hardware components and mechanisms like device controllers, DMA, interrupts, interrupt vectors and traps and how/why these are essential to the implementation of a modern operating system.
- Explain the need for dual-mode operation, the difference between user- and kernel-mode and how transitions between these modes occur.
- Contrast the role and execution environment of a user process vs. the OS kernel.
- Contrast the system call mechanism with an ordinary subroutine call and explain how a system call is implemented.
- List and explain the different types of protection (e.g., memory, CPU, I/O) an operating system is expected to provide to its processes, and how these are achieved with the help of hardware.
- Describe how base and limit registers function as a simple mechanism for memory protection.

Operating System Structures, Chapter 2

- Describe the jobs and responsibilities of a modern operating system and the types of services normally provided to user processes.
- List and describe steps performed during system boot.
- Describe the major components of a modern operating system, including the kernel and system programs.
- Describe different approaches to operating system organization, in particular, microkernel vs. monolithic kernel, and the relative advantages/disadvantages.

Processes, Chapter 3

- Define the term, Process, and describe the memory layout of a (single-threaded) process.
- Describe the basic steps of process creation.
- Describe the parent-child relationship among processes and how this yields a process tree.
- Implement simple programs using POSIX API for processes, explain the behavior of these system calls, and explain the behavior of example programs using them (including fork(), wait(), _exit() and exec*()).
- Describe the most important steps in a context switch starting from timer interrupts and how process context is maintained by the operating system.
- Describe the function and contents of the PCB.
- Explain the meanings of various process states (e.g., new, ready) and how transitions among these states may occur because of events inside and outside the process.
- Describe how and why PCBs may move among OS scheduling queues.
- Describe the signal mechanism available in Unix.
- Compare and contrast IPC mechanisms based on message passing with POSIX anonymous pipes, message queues, and shared memory, and compare direct and indirect communications.
- Based on homework assignments, exercises, and in-class examples, explain the operation of Unix IPC mechanisms: pipes, message queues and shared memory.

Threads, Chapter 4

- Describe how the traditional model of a (single-threaded) process can be generalized to include multiple threads.
- Explain how memory is laid out and how thread context is maintained in a multi-threaded process.
- Explain why threads can be an important and valuable resource in designing an application for a modern computer system.
- Explain differences in implementation and tradeoffs for user-level and kernel-level implementations of threads.
- Describe the blocking behavior of a process with user-level threads.
- Implement simple programs and program fragments using the Pthreads API, and explain the behavior of code fragments using this API (including pthread_create(), pthread_join()).
- Use argument and return value passing in the Pthreads API and explain the behavior of code fragments using this facility.
- Speculate about possible executions of multiple threads and their interactions through shared variables.
- Describe the different, general models for mapping process threads to kernel threads (e.g., many-to-one, one-to-one) the relevant tradeoffs.
- Describe asynchronous and deferred models of thread cancellation and the consequences of using one or the other.

CPU Scheduling, Chapter 5

- List specific reasons for a process leaving the running state, and how these are related to preemption by the CPU scheduler.
- Explain the meaning and importance of each of the five CPU scheduling criteria (CPU utilization, throughput, etc.) and determine values for these based on a CPU schedule.
- Explain why typical operating systems can't satisfy these requirements.
- Explain the necessity of burst length prediction in SJF and SRTF.
- Define starvation, recognize scenarios when it can occur and how aging may be used as a countermeasure.
- Explain the behavior and advantages/disadvantages of various CPU scheduling algorithms.
- Given a collection of processes arriving in the ready queue, develop a schedule corresponding to one of the CPU scheduling algorithms.
- For a particular schedule, compute waiting time, average waiting time, turnaround time and response time.
- Describe how and why multiple scheduling behaviors may be combined in a multi-level feedback queue to simultaneously accommodate processes with a variety of behaviors.
- Describe types of processor affinity and types of thread scheduling based on contention scope.

Synchronization, Chapter 6 and 7

- Describe the critical section problem using appropriate terms and the requirements for a solution (e.g., bounded waiting, progress).
- Identify deficiencies in flawed critical section solutions and describe them in terms of critical section requirements.
- Implement Peterson's solution, describe the advantages and disadvantages of this solution.
- Describe the behavior of semaphores and their relevant operations, acquire() and release(). Make judgments about the possible behavior of a multi-threaded program using these mechanisms.
- Use semaphores to solve simple synchronization problems (like the ones on the in-class exercises and the examples) using our semaphore pseudocode (e.g., sem s = 1; acquire(s); release(s);).
- Describe the bounded buffer problem and implement a bounded buffer solution using semaphores.
- Describe a typical implementation of a counting semaphore in the kernel.
- Implement critical section solutions based on atomic test-and-set or compare-and-swap CPU instructions.
- Describe the behavior of counting and binary semaphores.
- Speculate about possible executions of multiple threads and their interactions through shared variables.

- Use POSIX semaphores, describe the relevant types (sem_t) and functions (sem_init(), sem_destroy(), sem_wait() and sem_post()).
- Describe and use the Pthreads mutex APIs.
- Describe and use the Pthreads condition-variable APIs.
- Implement thread_join with condition variables and understand why each element is necessary.
- Implement a bounded buffer solution using mutex and condition variables. Describe why each element in the bounded-buffer solution is necessary and describe execution interleaving to show buggy solutions.
- Describe the difference between signal and broadcast operations on condition variables.
- Describe the classic dining-philosopher problem and potential deadlocks in solving the problem.
- Describe the classic readers-writers problem.