Operating Systems Section 001, Final 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.

Deadlock, Chapter 8

- Describe what a deadlock is and why it's undesirable.
- List and define the four necessary conditions for deadlock.
- Identify possibilities for deadlock in multi-threaded code with synchronization.
- Describe resource allocation graph notation and draw a graph resulting from an initial set of claims and a sequence of requests and assignments.
- Compare and contrast techniques based on deadlock detection, deadlock avoidance and deadlock prevention.
- Describe the four deadlock prevention techniques, each associated with a different necessary condition.
- Describe how various deadlock prevention techniques may or may not be appropriate for particular situations.
- Given a resource allocation graph with one or more instances per resource, determine whether the system is deadlocked.
- Identify processes that may be terminated in order to eliminate a deadlock.
- Define safe, unsafe and deadlocked states and classify resource allocation graphs accordingly.
- Summarize the Banker's algorithm and determine whether particular resource requests can be granted under this algorithm.

Memory, Chapter 9

- Describe address binding techniques (including advantages/disadvantages) for compile time, load time and execution time.
- Describe static linking, dynamic linking, dynamic loading and shared libraries and advantages of these techniques.
- Define swapping, identify the hardware involved and explain how and when it would be performed.

- Define contiguous allocation and common techniques for performing dynamic storage allocation (e.g., first-fit, best-fit, worst-fit). Describe the 50-percent rule.
- Explain the distinction between logical and physical addresses and the purpose of address translation.
- Define external and internal fragmentation and recognize what allocation practices can result in these conditions.
- Describe and perform address translation via a base / relocation register.
- Describe the organization and operation of a single-level and a multi-level paged memory system.
- Draw conclusions about the size of logical address space and page tables based on partial information (e.g., if a logical address is 2^{20} bytes and a page is 8 KB, how many entries will a single-level page table contain?)
- Define the valid, read-only, modified (dirty) and reference (used) bits, describe how they are stored and updated, and report how they relate to particular examples of address translation.
- Describe how the operating system populates page tables for user processes and how these can be built to implement shared memory, copy-on-write and a logical address space that grows as needed.
- Describe the need for a TLB, what it does and derive effective memory access time based on parameters for single- and multi-level paging.
- Describe how problems associated with a very large single-level page table can arise, and how a page-table length register, multi-level paging, hashed page tables and inverted page tables alleviate this problem.
- Describe structure of the page tables and the process of address translation for Intel's 32-bit paged memory and AMD 64 paged memory.
- Construct relevant structures and perform address translation for small examples of hashed and inverted page table.
- Perform address translation by hand using small examples of a single-level or a multi-level page table, a hashed page table or an inverted page table.

Virtual Memory, Chapter 10

- Describe demand paging, what it does, how it's implemented. Identify and describe the sequence of steps that normally occur when the OS responds to a page fault.
- Describe pure demand paging as a mechanism for demand loading.
- Compute an effective memory access time based on a page fault rate, memory access time and page fault overhead.
- Describe the FIFO, Optimal, and LRU page replacement algorithms and simulate these algorithms for a particular reference string. Describe their advantages and disadvantages.
- Describe Belady's anomaly, what a stack algorithm is and how stack algorithms avoid the anomaly.
- Describe and compare local and global page replacement policies.
- Describe the phenomenon of thrashing, what causes it and what can be done to remedy the situation.

- Describe the working set model and localities, and how different data structures and implementation techniques affect locality of reference.
- Describe the operation of the Linux 2.4 page replacement algorithm, including the concepts of page aging, page cleaning, minor page faults and active and inactive lists.

GPU Programming (CUDA Slides and Lecture)

- Describe the parallel execution model offered by CUDA, including the strengths of the GPU vs the CPU, the meaning of a kernel, grids and blocks.
- Describe the memory regions available to host and device threads, including host memory, device global memory, and shared memory
- You won't need to be able to write perfect CUDA code on the exam, but you should be able to explain, in general, what the different constructs mean (e.g., __device__) and what the few calls we looked at are for (e.g., cudaMalloc).

Distributed Systems, Chapter 19

- Describe a distributed system and the various advantages a distributed system can have over a centralized system
- Describe sources of unreliability a distributed system must deal with
- Describe and compare a network operating system and a distributed operating system
- Describe data, computation and process migration and how they are useful
- Describe the client / server model of a distributed application and various models for sharing responsibility between client and server
- Identify the roles of the application, transport, network, data link and physical networking layers. Describe the jobs and relationship between Internet Protocol (IP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)
- Describe, in general, what the Socket API is for and how it's used by a client and a server (you don't need to know the exact sequence of socket calls and their arguments)
- Describe what happens in a Remote Procedure Call (RPC) and how RPC differs from a local procedure call

File Systems, Chapters 13 and 14

- Describe attributes normally maintained for each file.
- Describe the function of inodes and the fields you'd expect to see in one.
- Describe basic operations on files.
- Describe the representation of a directory.
- Describe file access methods, including sequential access and direct access.
- Describe different file allocation approaches, including contiguous, linked, indexed allocation, and explain the advantages and disadvantages of each method.
- Perform mapping from logical address to physical address using contiguous, linked, indexed allocation.
- Describe file-allocation table (FAT).

- Explain multi-level indexing used in indexed allocation, calculate the largest file size it can support.

Storage Systems, Chapter 11

- Describe the geometry of a magnetic disk and how the logical address of a disk block is related.
- Describe the components of magnetic disk access time and how they compare for typical magnetic disks.
- Describe the operation of disk scheduling algorithms, FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK, their relative advantages and disadvantages, and their potential for starvation.
- Given a collection of disk requests, determine the behavior and total seek time for each of these scheduling algorithms.
- Describe the basic operation of flash NAND storage used to implement solid-state drives and compare it to magnetic disk.
- Describe how failure of individual disk sectors can be handled by the hardware via sector slipping and sector sparing.
- Describe how RAID uses multiple disks to obtain higher performance and greater reliability.
- Simulate RAID systems using Hamming code or parity to maintain redundant data or recover lost data.
- Compare the various raid levels in terms of their ability to provide greater storage capacity, greater reliability, improve disk access performance for a single process and parallelize disk access for multiple processes.

Protection and Security, Chapters 16 and 17

- Define and compare the topics of protection and security.
- List and describe typical access rights and how these are notated in the access matrix