CPCS 380 Operating System Principles

Midterm Exam Study Notes

Chapter 1

Introduction

Know the definition of an operating system
Familiar with the general system structure

1. Hardware
2. Operating Systems
3. Application Programs
4. Users

Computer startup (i.e. bootstrap programs)
Computer system organizations (CPUs, device controllers, bus memory cycles)
Interrupts and Interrupt Handling
I/O Structure

1. System calls
2. Device-status tables

Storage Structure

1. Storage Hierarchy
2. Caching

Direct Memory Access Structure

1. How DMA works

Computer-System Architecture

1. Symmetric Multiprocessing Architecture
2. Dual-Core Design
3. Clustered Systems

Operating System Structure

1. Operating-System Operations
2. Process Management
3. Memory Management
4. Storage Management
5. I/O Subsystem
6. Protection & Security

OS Services

1. User interface
2. Program execution
3. I/O operations
4. File-system manipulation
5. Communications
6. Error Detection
7. Resource allocation
8. Accounting, Protection and Security

Operating Systems Interface

1. CLI or GUI
    a. Shells (i.e. csh, bash, …)

System Calls/Programs

1. APIs
2. System Call Implementation

Chapter 2

Processes

Process Concept

1. Text section, data section, program counter, stack, heap
2. Process in memory
3. Process state
    a. New, running, waiting, …
4. Process Control Block
5. Threads of execution

Context-Switch

Operation on Processes

1. Creation, termination
2. Forking processes

Inter-process Communications

1. Shared memory
2. Message passing
3. Independent/Cooperating Processes
4. Indirect Communication
    a. Mailboxes
5. Synchronization
    a. Blocking/Non-blocking
6. Pipes, Sockets, Remote Procedures Calls (RPCs)

Threads & Concurrency

Multithreaded Architectures

1. Benefits
2. Multicore programming

Concurrency/Parallelism

User/Kernel Threads & Multithreading Models

1. Thread Libraries
2. Pthreads examples

Signal Handling

Thread Cancellation

Synchronization

Describe the Critical Section Problem.
Know the three requirements the critical-section problem must satisfy

1. Mutual Exclusion
2. Progress
3. Bounded Waiting

Peterson's Solution – understand the classic software-based solution to the critical-section problem (how to implement the solution)
Mutexes – difference between mutex/binary semaphore. How to implement a mutex

Define Critical Regions
What is an Atomic Transaction?

What is a semaphore? How are the used? How are they implemented?
Know/Describe some classic problems of synchronization

1. Bounded-Buffer Problem
2. Reader-Writers Problem

CPU Scheduling

Basic Concepts

1. CPU & I/O Bursts

CPU Scheduler

1. Preemptive/Non-preemptive
2. Dispatcher
3. Scheduling Criteria
4. Scheduling Algorithms
    a. FCFS
    b. SJF
    c. Priority
    d. RR
5. Multi-Processor Scheduling
    a. SMP/AMP
    b. Soft/Hard Affinity
    c. Load balancing
6. Real-Time CPU Scheduling
    a. Hard/Soft Real-Time systems
    b. Rate Monotonic (RM)
    c. Earliest Deadline First (EDF)

Deadlocks

What is deadlock?
What is a Resource Allocation Graph (RAG) – how is it used?
Know the conditions necessary for deadlock to hold and what they mean.

1. Mutual Exclusion
2. Hold and Wait
3. No preemption
4. Circular Wait

Know the methods for handling deadlock

1. Deadlock Prevention
2. Deadlock Avoidance
3. Deadlock Detection

Identify some the methods for recovering from deadlock

1. Process Termination
   a. Abort all processes
   b. Abort one process at a time
2. Resource Preemption
   a. Select a victim
   b. Rollback
   c. Starvation

Example Questions

Chapter 1 - Introduction

1. What are the three main purposes of an operating system?

2. Keeping in mind the various definitions of operating system, consider whether the operating system should include applications such as Web browsers and mail programs. Argue both that it should and that it should not, and support your answers.

3. How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) system?

4. Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

5. Distinguish between the client–server and peer-to-peer models of distributed systems.

6. What is the purpose of system calls?

7. What is the purpose of the command interpreter? Why is it usually separate from the kernel?

8. What is the purpose of system programs?

9. Why do some systems store the operating system in firmware, while others store it on disk?

Chapter 2 - Processes

1. The Sun UltraSPARC processor has multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

2. When a process creates a new process using the fork() operation, which of the following state is shared between the parent process and the child process?
   a. Stack
   b. Heap
   c. Shared Memory

3. Original versions of Apple's mobile iOS operating system provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

4. With respect to the RPC mechanisms, consider the "exactly once" semantic. Does the algorithm for implementing this semantic execute correctly even if the ACK message back to the client is lost due to a network problem? Describe the sequence of messages and discuss whether "exactly once" is still preserved.

Chapter 2- Threads

1. Provide three programming examples in which multithreading provides better performance than a single-threaded solution.

2. Describe the actions taken by a kernel to context-switch between kernel level threads.

3. What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

4. What resources are used when a thread is created? How do they differ from those used when a process is created?

Chapter 2 – Synchronization

1. What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

2. Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

3. Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

4. Illustrate how a binary semaphore can be used to implement mutual exclusion among *n* processes.

5. Describe what changes would be necessary to the producer and consumer processes so that a mutex lock could be used instead of a binary semaphore.

Chapter 2- CPU Scheduling

1. Explain the concept of a CPU–I/O burst cycle.

2. What role does the dispatcher play in CPU scheduling?

3. Explain the difference between response time and turnaround time. These times are both used to measure the effectiveness of scheduling schemes.

4. What effect does the size of the time quantum have on the performance of an RR algorithm?

Chapter 6 - Deadlocks

1. To handle deadlocks, operating systems most often _____.

   A) pretend that deadlocks never occur
   B) use protocols to prevent or avoid deadlocks
   C) detect and recover from deadlocks
   D) None of the above

2. Describe the four conditions that must hold simultaneously in a system if a deadlock is to occur.

3. What are the three general ways that a deadlock can be handled?

4. What is the difference between deadlock prevention and deadlock avoidance?