**Operating Systems: Comprehensive Study Guide**

**Table of Contents**

1. [Introduction to Operating Systems](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#introduction-to-operating-systems)
2. [Process Management](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#process-management)
3. [CPU Scheduling](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#cpu-scheduling)
4. [Concurrency and Synchronization](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#concurrency-and-synchronization)
5. [Deadlocks](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#deadlocks)
6. [Memory Management](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#memory-management)
7. [Virtual Memory](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#virtual-memory)
8. [File Systems](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#file-systems)
9. [Distributed Operating Systems](https://claude.ai/chat/c0a4072f-3339-475f-9b47-acef17abf4e0#distributed-operating-systems)

**Introduction to Operating Systems**

**Definition and Purpose**

An **operating system (OS)** is system software that manages computer hardware, software resources, and provides common services for computer programs. It acts as an intermediary between users and the computer hardware.

**Core Functions of an OS**

1. **Process Management**: Creating, scheduling, and terminating processes
2. **Memory Management**: Allocating and deallocating memory space
3. **File Management**: Creating, deleting, and accessing files
4. **Device Management**: Managing I/O devices and their drivers
5. **Security and Protection**: Controlling system access and protecting resources

**Types of Operating Systems**

1. **Batch OS**: Executes jobs without user interaction
2. **Time-Sharing OS**: Multiple users share computing resources simultaneously
3. **Distributed OS**: Manages a group of distinct computers that appear as a single system
4. **Network OS**: Manages computers and devices across a network
5. **Real-Time OS**: Guarantees processing within specific time constraints
   * **Hard Real-Time**: Missing a deadline is a total system failure
   * **Soft Real-Time**: Missing a deadline results in degraded quality

**OS Structures**

1. **Monolithic Structure**: OS runs as a single program in kernel mode
2. **Layered Approach**: OS divided into hierarchical layers
3. **Microkernel**: Minimal kernel with basic features; other services run as user processes
4. **Modular Approach**: Core kernel with dynamically loadable modules

**System Calls**

**System calls** are the programming interface to the services provided by the OS.

**Types of System Calls:**

1. **Process Control**: create, terminate, wait, allocate/free memory
2. **File Management**: create, delete, open, close, read, write
3. **Device Management**: request/release device, read/write to device
4. **Information Maintenance**: get/set time, system data
5. **Communication**: create/delete communication connection, send/receive messages

**OS Boot Process**

1. **Power-On Self-Test (POST)**: Basic hardware check
2. **Bootstrap Loader**: Loads the kernel into memory
3. **Kernel Initialization**: Sets up memory, loads device drivers
4. **System Processes Initialization**: Starts system daemons and services
5. **User Login**: Presents login interface

**Process Management**

**Process Concept**

A **process** is an instance of a program in execution. It is the unit of work in a modern time-sharing system.

**Components of a Process**

1. **Program Counter**: Contains the address of the next instruction
2. **Process State**: Running, ready, waiting, new, terminated
3. **CPU Registers**: Accumulators, index registers, stack pointers
4. **Memory Limits**: Memory allocated to the process
5. **List of Open Files**: Files currently accessed by the process

**Process Control Block (PCB)**

The **PCB** is a data structure containing information associated with each process:

* Process ID (PID)
* Process state
* Program counter
* CPU registers
* CPU scheduling information
* Memory management information
* Accounting information
* I/O status information

**Process States**

1. **New**: Process is being created
2. **Ready**: Process is waiting to be assigned to a processor
3. **Running**: Instructions are being executed
4. **Waiting/Blocked**: Process is waiting for some event to occur
5. **Terminated**: Process has finished execution

**Process Operations**

1. **Process Creation**
   * Parent process creates children processes
   * Resource sharing options:
     + Parent and children share all resources
     + Children share subset of parent's resources
     + Parent and children share no resources
   * Execution options:
     + Parent and children execute concurrently
     + Parent waits until children terminate
2. **Process Termination**
   * Causes:
     + Normal exit
     + Error exit
     + Fatal error
     + Killed by another process

**Inter-Process Communication (IPC)**

Mechanisms for processes to communicate and synchronize their actions:

1. **Shared Memory**
   * Processes share a region of memory
   * Fast, but requires synchronization
   * Example: Producer-Consumer problem
2. **Message Passing**
   * Processes exchange messages
   * Can be implemented in various ways:
     + Direct/Indirect communication
     + Synchronous/Asynchronous communication
     + Automatic/Explicit buffering

**Threads**

A **thread** is a basic unit of CPU utilization, consisting of:

* Thread ID
* Program counter
* Register set
* Stack

**Types of Threads**

1. **User-Level Threads**: Managed by user-level thread library
   * Fast thread switching
   * Can run on any OS
   * If one thread blocks, all threads block
2. **Kernel-Level Threads**: Managed by the OS kernel
   * Slow thread switching
   * If one thread blocks, others can run
   * OS-dependent
3. **Hybrid Approaches**: Combines user and kernel threads
   * Example: Many-to-Many model

**Thread Implementation Issues**

* Thread creation and management
* Thread cancellation
* Signal handling
* Thread-local storage
* Scheduler activations

**CPU Scheduling**

**Basic Concepts**

**CPU scheduling** determines which process runs on the CPU when there are multiple ready processes.

**Scheduling Criteria**

1. **CPU Utilization**: Keep the CPU as busy as possible
2. **Throughput**: Number of processes completed per time unit
3. **Turnaround Time**: Time from submission to completion
4. **Waiting Time**: Time spent in the ready queue
5. **Response Time**: Time from submission until first response

**Scheduling Algorithms**

1. **First-Come, First-Served (FCFS)**
   * Non-preemptive
   * Simple but can cause convoy effect
   * Average waiting time can be long
2. **Shortest Job First (SJF)**
   * Can be preemptive or non-preemptive
   * Optimal for average waiting time
   * Challenge: Predicting job length
3. **Priority Scheduling**
   * Each process assigned a priority
   * Can be preemptive or non-preemptive
   * Issue: Starvation (indefinite blocking)
   * Solution: Aging (increasing priority over time)
4. **Round Robin (RR)**
   * Preemptive
   * Time quantum assigned to each process
   * After time quantum expires, process is moved to end of ready queue
   * Performance depends on size of time quantum
5. **Multilevel Queue Scheduling**
   * Ready queue divided into separate queues
   * Processes permanently assigned to one queue
   * Each queue has its own scheduling algorithm
   * Must schedule between queues
6. **Multilevel Feedback Queue**
   * Similar to multilevel queue, but allows processes to move between queues
   * Adapts to changing behavior of processes
   * Most general, but most complex

**Scheduling in Multi-Processor Systems**

1. **Symmetric Multiprocessing (SMP)**
   * Each processor self-schedules
   * Common ready queue or private ready queues
2. **Asymmetric Multiprocessing**
   * One processor handles scheduling decisions
   * Others execute user code
3. **Processor Affinity**
   * Process tends to run on same processor
   * Soft affinity vs. hard affinity
4. **Load Balancing**
   * Push migration: Periodically check load
   * Pull migration: Idle processors take processes
5. **Multicore Processors**
   * Multiple processors on a single chip
   * Memory stall: waiting for memory access
   * Hardware threads: Multiple threads per core

**Concurrency and Synchronization**

**The Critical Section Problem**

A **critical section** is a segment of code where shared resources are accessed.

**Requirements for a Solution**

1. **Mutual Exclusion**: Only one process in critical section at a time
2. **Progress**: If no process is in critical section, one should be allowed to enter
3. **Bounded Waiting**: Limited number of times other processes enter their critical sections after a process has requested entry

**Synchronization Tools**

1. **Mutex Locks**
   * Simplest synchronization tool
   * Has two operations:
     + acquire(): Enter critical section
     + release(): Exit critical section
   * Implemented with busy waiting (spinlock) or sleep/wakeup
2. **Semaphores**
   * Integer variable accessed through wait() and signal() operations
   * Types:
     + **Binary semaphore**: Values 0 or 1 only
     + **Counting semaphore**: Can have arbitrary domain
   * Can solve various synchronization problems
3. **Monitors**
   * High-level synchronization construct
   * Abstract data type with operations controlled for mutual exclusion
   * Condition variables for synchronization:
     + wait(): Process joins queue of waiting processes
     + signal(): Wakes up one waiting process

**Classic Synchronization Problems**

1. **Bounded-Buffer (Producer-Consumer) Problem**
   * Fixed-size buffer, shared between producers and consumers
   * Need synchronization for:
     + Mutual exclusion when accessing the buffer
     + Managing full/empty conditions
2. **Readers-Writers Problem**
   * Multiple readers can read simultaneously
   * Writers need exclusive access
   * Variations:
     + First readers-writers problem: No reader waits unless a writer has access
     + Second readers-writers problem: Writers get priority
3. **Dining Philosophers Problem**
   * Five philosophers sitting at a table
   * Need two chopsticks to eat
   * Challenge: Avoid deadlock and starvation

**Race Conditions**

A **race condition** occurs when multiple processes access and manipulate shared data concurrently, and the final result depends on the order of execution.

**Livelock and Starvation**

* **Livelock**: Processes change states in response to each other, but make no progress
* **Starvation**: Process never gets the resources it needs

**Deadlocks**

**Deadlock Characterization**

A **deadlock** is a situation where a set of processes are blocked because each is holding resources and waiting to acquire resources held by another process.

**Necessary Conditions for Deadlock**

1. **Mutual Exclusion**: At least one resource must be non-sharable
2. **Hold and Wait**: Process holds resources while waiting for others
3. **No Preemption**: Resources cannot be forcibly taken from processes
4. **Circular Wait**: Circular chain of processes waiting for resources

**Deadlock Handling Strategies**

1. **Prevention**
   * Ensure at least one of the four necessary conditions cannot hold
   * Methods:
     + Eliminate mutual exclusion: Not always possible
     + Eliminate hold and wait: Request all resources at once
     + Allow preemption: Take resources away if needed
     + Eliminate circular wait: Impose resource ordering
2. **Avoidance**
   * Requires advance information about resource usage
   * **Safe State**: System can allocate resources to each process in some order
   * **Resource Allocation Graph**: Model of resource allocation
   * **Banker's Algorithm**: Checks if request leads to safe state
3. **Detection**
   * Allow deadlocks to occur
   * Periodically check for deadlocks
   * Methods:
     + Resource allocation graph
     + Matrix-based algorithm (similar to Banker's)
4. **Recovery**
   * When deadlock is detected, break it by:
     + Process termination (kill one or all deadlocked processes)
     + Resource preemption (take resources from processes)

**Memory Management**

**Basic Memory Management**

**Memory management** involves keeping track of which parts of memory are being used and allocating/deallocating memory space.

**Address Binding**

Process of mapping program addresses to physical memory addresses:

1. **Compile time**: Absolute code, must know where process will reside
2. **Load time**: Relocatable code, binding delayed until load time
3. **Execution time**: Binding delayed until run time, requires special hardware

**Logical vs. Physical Address Space**

* **Logical address**: Generated by CPU, also called virtual address
* **Physical address**: Address seen by memory unit
* **Memory Management Unit (MMU)**: Hardware device that maps virtual to physical addresses

**Memory Allocation Techniques**

1. **Contiguous Allocation**
   * Each process is allocated a single continuous block of memory
   * Types:
     + **Fixed partitioning**: Memory divided into fixed-sized partitions
     + **Dynamic partitioning**: Partitions of variable size
2. **Fragmentation**
   * **External fragmentation**: Total memory space exists but non-contiguous
   * **Internal fragmentation**: Allocated memory larger than requested memory
   * Solutions:
     + **Compaction**: Shuffle memory to place all free memory together
     + **Paging**: Non-contiguous allocation

**Paging**

A memory management scheme that eliminates external fragmentation:

* Physical memory divided into fixed-sized blocks called **frames**
* Logical memory divided into blocks of same size called **pages**
* Page table maps logical pages to physical frames
* Advantages:
  + No external fragmentation
  + Simple allocation
* Disadvantages:
  + Internal fragmentation
  + Complex address translation

**Page Table Implementation**

1. **Dedicated registers**: Fast but limited number of entries
2. **Memory-based page table**: Flexible but slower
3. **Translation lookaside buffer (TLB)**: Fast associative memory cache for page tables

**Segmentation**

Memory management scheme reflecting user's view of memory:

* Program divided into segments (main, procedure, function, etc.)
* Logical address consists of <segment-number, offset>
* Segment table maps segments to physical addresses
* Advantages:
  + Supports sharing and protection
  + No internal fragmentation
* Disadvantages:
  + External fragmentation
  + Complex memory allocation

**Segmentation with Paging**

Combines advantages of both:

* Segments are divided into pages
* Only the pages of a segment are loaded into memory
* Segment table points to page tables
* Page tables point to frames

**Virtual Memory**

**Concept of Virtual Memory**

**Virtual memory** is a technique that allows the execution of processes that may not be completely in memory, effectively separating logical memory from physical memory.

**Benefits**

1. Allows running programs larger than physical memory
2. Enables higher degree of multiprogramming
3. Less I/O for loading and swapping processes

**Demand Paging**

* Bring pages into memory only when needed
* Uses a **page fault** mechanism:
  1. Reference to page triggers page fault
  2. OS locates page on disk
  3. OS brings page into free frame
  4. OS updates page tables
  5. Restart instruction

**Page Replacement Algorithms**

Determine which page to replace when a new page needs to be brought in:

1. **First-In-First-Out (FIFO)**
   * Replace oldest page
   * Simple but not always efficient
   * Suffers from **Belady's anomaly**: More frames can lead to more page faults
2. **Optimal Algorithm**
   * Replace page that will not be used for longest period
   * Lowest possible page fault rate
   * Not implementable (requires future knowledge)
   * Used as theoretical benchmark
3. **Least Recently Used (LRU)**
   * Replace page that has not been used for longest time
   * Good approximation of optimal
   * Implementation challenges:
     + Counter implementation
     + Stack implementation
4. **Approximation Algorithms**
   * **Reference bit**: Set when page is referenced
   * **Additional reference bits**: Record history over time
   * **Second chance (Clock) algorithm**: FIFO with reference bit

**Frame Allocation**

How to allocate frames among various processes:

1. **Equal allocation**: Each process gets equal share
2. **Proportional allocation**: Allocation based on process size
3. **Priority allocation**: Allocation based on process priority

**Thrashing**

**Thrashing** occurs when a process spends more time paging than executing.

Causes:

* Process doesn't have enough frames
* High degree of multiprogramming

Solutions:

* **Working Set Model**: Track working set of pages
* **Page Fault Frequency**: Adjust allocation based on fault rate

**Memory-Mapped Files**

Map file I/O into virtual memory operations:

* File mapped to virtual memory region
* Page faults load data from file
* Dirty pages written back to file
* Simplifies file I/O programming
* Allows sharing of files between processes

**File Systems**

**File Concept**

A **file** is a named collection of related information stored on secondary storage.

**File Attributes**

* Name
* Type
* Location
* Size
* Protection
* Time (created, last access, last modification)

**File Operations**

* Create
* Write
* Read
* Reposition (seek)
* Delete
* Truncate

**File Access Methods**

1. **Sequential Access**: Records processed in order
2. **Direct Access**: Records can be read in any order
3. **Indexed Access**: Index provides pointers to various blocks

**Directory Structure**

Organizes files for easy retrieval:

1. **Single-Level Directory**: All files in same directory
2. **Two-Level Directory**: Separate directory for each user
3. **Tree-Structured Directory**: Hierarchical file system
4. **Acyclic-Graph Directory**: Allows shared subdirectories/files
5. **General Graph Directory**: Allows cycles

**File System Implementation**

**Allocation Methods**

1. **Contiguous Allocation**
   * Each file occupies set of contiguous blocks
   * Simple, good performance for sequential access
   * External fragmentation, difficult to grow files
2. **Linked Allocation**
   * Each file is a linked list of blocks
   * No external fragmentation
   * Poor random access performance
   * Reliability issues (lost pointers)
   * Variant: **File Allocation Table (FAT)**
3. **Indexed Allocation**
   * Index block contains pointers to data blocks
   * Supports direct access
   * No external fragmentation
   * Large files require multiple index blocks
   * Variants: linked scheme, multilevel index, combined scheme

**Free Space Management**

1. **Bit vector**: Each block represented by one bit
2. **Linked list**: Link all free blocks
3. **Grouping**: First free block contains addresses of next n free blocks
4. **Counting**: Keep address of first free block and count of contiguous free blocks

**File System Reliability**

* **Consistency checking**: File system check (fsck/chkdsk)
* **Backup and restore**: Regular saving of file system
* **Journaling**: Log-based transaction recovery

**Disk Structure and Scheduling**

* **Disk sectors**: Basic unit of disk I/O
* **Disk scheduling algorithms**:
  + FCFS (First-Come, First-Served)
  + SSTF (Shortest Seek Time First)
  + SCAN (elevator algorithm)
  + C-SCAN (circular SCAN)
  + LOOK and C-LOOK

**Distributed Operating Systems**

**Concepts and Design Issues**

A **distributed operating system** manages a collection of independent computers that appears to users as a single system.

**Characteristics**

* Resource sharing
* Computation speedup
* Reliability
* Communication

**Design Issues**

* Transparency
* Flexibility
* Reliability
* Performance
* Scalability

**Communication in Distributed Systems**

1. **Remote Procedure Call (RPC)**
   * Client-server model for distributed computing
   * Looks like local procedure call to programmer
   * Issues: parameter passing, binding, failure handling
2. **Message Passing**
   * Processes communicate by exchanging messages
   * Synchronous vs. asynchronous
   * Direct vs. indirect communication

**Distributed File Systems**

1. **Design Issues**
   * Naming and transparency
   * Remote file access
   * Caching strategies
   * Replication
2. **Examples**
   * Network File System (NFS)
   * Andrew File System (AFS)
   * Google File System (GFS)

**Distributed Synchronization**

1. **Clock Synchronization**
   * Physical clocks vs. logical clocks
   * Lamport's timestamps
   * Vector clocks
2. **Mutual Exclusion**
   * Centralized algorithm
   * Distributed algorithm
   * Token-passing algorithms
3. **Election Algorithms**
   * Bully algorithm
   * Ring algorithm

**Distributed Deadlock**

1. **Detection**
   * Centralized approach
   * Distributed approach
2. **Prevention and Avoidance**
   * Global resource ordering
   * Deadlock prevention methods adapted for distributed systems

**Consistency and Replication**

1. **Data-Centric Consistency Models**
   * Strict consistency
   * Sequential consistency
   * Causal consistency
   * Eventual consistency
2. **Client-Centric Consistency Models**
   * Monotonic reads/writes
   * Read/write your writes

**Fault Tolerance**

1. **Failure Models**
   * Crash failures
   * Omission failures
   * Timing failures
   * Byzantine failures
2. **Techniques**
   * Redundancy
   * Process resilience
   * Reliable group communication
   * Consensus protocols

**Additional Topics and Advanced Concepts**

**Virtualization**

1. **Types of Hypervisors**
   * Type 1 (bare-metal): Runs directly on hardware
   * Type 2 (hosted): Runs on host OS
2. **Virtualization Techniques**
   * Full virtualization
   * Paravirtualization
   * Hardware-assisted virtualization

**Cloud Computing**

1. **Service Models**
   * Infrastructure as a Service (IaaS)
   * Platform as a Service (PaaS)
   * Software as a Service (SaaS)
2. **Deployment Models**
   * Public cloud
   * Private cloud
   * Hybrid cloud
   * Community cloud

**Mobile Operating Systems**

1. **Characteristics**
   * Power management
   * Limited resources
   * Connectivity management
   * Security model
2. **Examples**
   * Android
   * iOS
   * HarmonyOS

**Security in Operating Systems**

1. **Protection Mechanisms**
   * Access control
   * Capabilities
   * Security policies
2. **Authentication**
   * Passwords
   * Biometrics
   * Multi-factor authentication
3. **Threats and Attacks**
   * Malware
   * Privilege escalation
   * Denial of service

**Real-Time Systems**

1. **Characteristics**
   * Predictability
   * Reliability
   * Performance
2. **Scheduling**
   * Rate-monotonic scheduling
   * Earliest deadline first
   * Priority inversion problem

**Study Strategies and Exam Tips**

**Understanding Core Concepts**

* Focus on understanding the principles behind each topic rather than memorizing details
* Draw diagrams to visualize abstract concepts
* Implement simple versions of algorithms to understand their operation

**Practice Problems**

* Work through scheduling, page replacement, and deadlock detection examples
* Draw process state diagrams and transitions
* Trace through synchronization problems

**Connecting Topics**

* Understand how topics interconnect (e.g., how process scheduling affects memory management)
* Compare and contrast different approaches (e.g., paging vs. segmentation)
* Consider trade-offs in different design choices

**Exam Approach**

* Read questions carefully to identify the specific concept being tested
* Start with a clear definition of key terms
* Support answers with examples or diagrams when appropriate
* Justify your answers by explaining advantages and disadvantages

**Glossary of Key OS Terms**

* **Context Switch**: Process of saving the state of a process so that it can be restored and execution resumed from the same point later
* **Daemon**: Background process that runs without user intervention
* **Dispatch**: Function that gives control of the CPU to the process selected by the scheduler
* **Fragmentation**: Wasted space in memory or disk storage
* **Kernel**: Core component of an operating system
* **Multiprogramming**: Having multiple programs loaded into memory and ready for execution
* **Page Fault**: Interrupt that occurs when a program requests a page not currently mapped to physical memory
* **Preemption**: Temporarily interrupting a task being carried out by a computer system
* **Quantum**: Maximum amount of time a process can run before being preempted
* **Semaphore**: Variable used to control access to a common resource by multiple processes
* **Shell**: User interface for access to an operating system's services
* **Spooling**: Simultaneous Peripheral Operations On-Line, buffering data for peripheral devices
* **Swapping**: Process of moving a process from main memory to disk and vice versa
* **Thrashing**: State where excessive paging operations are performed
* **Time-sharing**: Allows multiple users to interact with a computer system at the same time