Operating Systems Fundamentals - Complete Study Notes

Table of Contents

Introduction to Operating Systems

What is an Operating System?

An operating system (OS) is a software program that acts as an intermediary between computer hardware and user applications. It manages computer resources and provides services to applications.

Main Functions of an OS

1. **Resource Management**: CPU, memory, storage, I/O devices
2. **Process Management**: Creating, scheduling, and terminating processes
3. **Memory Management**: Allocating and deallocating memory
4. **File Management**: Organizing and accessing files
5. **Security**: Protecting system resources and user data
6. **User Interface**: Providing command-line or graphical interfaces

Types of Operating Systems

Based on User Interface

- **Command Line Interface (CLI)**: Text-based interaction (Unix, DOS)
- **Graphical User Interface (GUI)**: Visual interaction (Windows, macOS, Linux Desktop)

Based on Processing

- **Batch Processing**: Jobs processed in batches without user interaction
- **Time-Sharing**: Multiple users share system resources simultaneously
- **Real-Time**: Immediate response to input (embedded systems, control systems)
- **Distributed**: Multiple computers work together as one system

Based on Users

- **Single-User**: Only one user at a time (early personal computers)
- **Multi-User**: Multiple users can use the system simultaneously (Unix, Linux)

OS Services

1. **Program Execution**: Loading and running programs
2. **I/O Operations**: Reading from and writing to devices
3. **File System Manipulation**: Creating, deleting, reading, writing files
4. **Communications**: Inter-process communication, networking
5. **Error Detection**: Hardware and software error handling
6. **Resource Allocation**: Managing CPU, memory, files, I/O devices
7. **Accounting**: Tracking resource usage
8. **Protection and Security**: Controlling access to resources

---

System Structure

Computer System Components

1. **Hardware**: CPU, memory, I/O devices, storage
2. **Operating System**: Manages hardware resources
3. **Application Programs**: Word processors, compilers, web browsers
4. **Users**: People, machines, other computers

System Calls

System calls are the interface between user programs and the OS kernel. They provide access to OS services.

Types of System Calls

1. **Process Control**: fork(), exec(), wait(), exit()
2. **File Management**: open(), read(), write(), close()

3. **Device Management**: request(), release(), read(), write()
4. **Information Maintenance**: getpid(), alarm(), sleep()
5. **Communication**: pipe(), msgget(), msgrcv(), msgsnd()

System Call Implementation

```
User Program → System Call Interface → OS Kernel → Hardware
```

OS Structure Models

Monolithic Structure

- All OS components in single address space
- **Advantages**: Fast communication, efficient
- **Disadvantages**: Complex, difficult to maintain, less secure
- **Examples**: Early Unix, MS-DOS

Layered Structure

- OS divided into layers, each built on lower layers
- **Advantages**: Modular, easier to debug
- **Disadvantages**: Performance overhead, difficult to define layers
- **Example**: THE operating system

Microkernel Structure

- Minimal kernel with basic services
- Other services run as user-level processes
- **Advantages**: Extensible, reliable, secure
- **Disadvantages**: Performance overhead due to message passing
- **Examples**: Mach, L4, QNX

Modular Structure

- Kernel has core components plus loadable modules
- **Advantages**: Flexible, efficient
- **Disadvantages**: More complex than monolithic
- **Examples**: Modern Linux, Solaris

Process Management

Process Concept

A process is a program in execution. It includes:

- **Program Code** (text section)
- **Program Counter** (current activity)
- **Stack** (temporary data - parameters, return addresses, local variables)
- **Data Section** (global variables)
- **Heap** (dynamically allocated memory)

Process States

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

Process Control Block (PCB)

Contains information about a process:

- **Process State**: Current state of the process
- **Program Counter**: Address of next instruction
- **CPU Registers**: Contents of processor registers
- **CPU Scheduling Information**: Priority, scheduling queue pointers
- **Memory Management Information**: Memory allocated to process
- **Accounting Information**: CPU used, clock time elapsed
- **I/O Status Information**: I/O devices allocated, open files

Process Operations

Process Creation

- **Parent Process** creates **Child Process**
- **fork()** system call creates new process
- **exec()** system call replaces process image

c

```c
// Example of process creation
pid_t pid = fork();
if (pid == 0) {
    // Child process
    exec("new_program");
} else if (pid > 0) {
    // Parent process
    wait(NULL);
}
```

Process Termination

- **exit()** system call terminates process
- **wait()** system call waits for child termination
- **Zombie Process**: Terminated but not yet reaped by parent
- **Orphan Process**: Parent terminated before child

Inter-Process Communication (IPC)

Shared Memory

- Processes share a region of memory
- **Advantages**: Fast communication
- **Disadvantages**: Synchronization needed

Message Passing

- Processes communicate through messages
- **Direct Communication**: Processes name each other explicitly
- **Indirect Communication**: Messages sent through mailboxes/ports

IPC Mechanisms

1. **Pipes**: Unidirectional communication channel
2. **Named Pipes (FIFOs)**: Bidirectional, persistent
3. **Message Queues**: Structured message passing
4. **Shared Memory**: Fast, requires synchronization
5. **Semaphores**: Synchronization primitive
6. **Sockets**: Network communication

CPU Scheduling

Basic Concepts

CPU scheduling determines which process runs when multiple processes are ready to execute.

Scheduling Criteria

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

Scheduling Algorithms

First-Come, First-Served (FCFS)

- Processes served in order of arrival
- **Advantages**: Simple, fair
- **Disadvantages**: Poor performance, convoy effect

Shortest Job First (SJF)

- Process with shortest CPU burst runs first
- **Advantages**: Optimal for average waiting time
- **Disadvantages**: Difficult to predict burst time, starvation

Round Robin (RR)

- Each process gets fixed time quantum
- **Advantages**: Fair, good response time
- **Disadvantages**: Context switching overhead

Priority Scheduling

- Process with highest priority runs first
- **Advantages**: Important processes get preference
- **Disadvantages**: Starvation of low-priority processes

Multilevel Queue Scheduling

- Ready queue divided into separate queues
- Each queue has its own scheduling algorithm
- **Example**: System processes, interactive processes, batch processes

Multilevel Feedback Queue

- Processes can move between queues
- **Advantages**: Flexible, responsive
- **Disadvantages**: Complex to implement

Scheduling in Multi-Processor Systems

1. **Asymmetric Multiprocessing**: One processor does scheduling
2. **Symmetric Multiprocessing**: Each processor schedules itself
3. **Processor Affinity**: Process runs on same processor
4. **Load Balancing**: Distribute processes evenly

Process Synchronization

Critical Section Problem

Multiple processes accessing shared data concurrently can lead to data inconsistency.

Solution Requirements

1. **Mutual Exclusion**: Only one process in critical section
2. **Progress**: Selection of next process cannot be postponed indefinitely
3. **Bounded Waiting**: Limit on waiting time

Synchronization Tools

Mutex Locks

- **Mutual Exclusion** lock
- **acquire()** and **release()** operations
- **Spinlock**: Busy waiting

```c
acquire() {
    while (!available); // busy wait
    available = false;
}

release() {
    available = true;
}
```

Semaphores

- Integer variable with atomic operations
- **wait()** (P) and **signal()** (V) operations
- **Binary Semaphore**: 0 or 1 (similar to mutex)
- **Counting Semaphore**: Can have value > 1

```c
wait(S) {
    while (S <= 0); // busy wait
    S--;
}

signal(S) {
    S++;
}
```

Monitors

- High-level synchronization construct
- Only one process can be active within monitor
- **Condition Variables**: wait() and signal()

Classical Synchronization Problems

Producer-Consumer Problem

- **Producer** creates data, **Consumer** uses data
- **Bounded Buffer**: Fixed-size buffer
- **Solution**: Use semaphores for synchronization

Readers-Writers Problem

- Multiple readers can access simultaneously
- Only one writer can access at a time
- **Solution**: Use reader count and mutex

Dining Philosophers Problem

- Five philosophers, five chopsticks
- Potential for deadlock
- **Solution**: Limit number of philosophers, ordering

Deadlocks

Deadlock Definition

A set of blocked processes, each holding a resource and waiting for another resource held by another process in the set.

Necessary Conditions for Deadlock

1. **Mutual Exclusion**: Only one process can use resource at a time
2. **Hold and Wait**: Process holds resource while waiting for another
3. **No Preemption**: Resources cannot be forcibly taken
4. **Circular Wait**: Circular chain of processes waiting for resources

Deadlock Prevention

Ensure at least one necessary condition cannot hold:

Mutual Exclusion

- Make resources sharable (not always possible)

Hold and Wait

- Require process to request all resources at once
- Release all resources before requesting new ones

No Preemption

- Allow preemption of resources
- Process releases resources if cannot get all needed

Circular Wait

- Impose ordering on resource types
- Process can only request resources in increasing order

Deadlock Avoidance

System has additional information about resource requests.

Safe State

- System can allocate resources to each process in some order
- Avoid unsafe states

Banker's Algorithm

- Checks if granting request leaves system in safe state
- Uses **Allocation**, **Max**, and **Available** matrices

Deadlock Detection

Allow deadlocks to occur, then detect and recover.

Detection Algorithm

- Wait-for graph for single instance resources
- Reduction algorithm for multiple instances

Recovery from Deadlock

1. **Process Termination**: Kill processes until deadlock broken
2. **Resource Preemption**: Take resources from processes

Memory Management

Basic Concepts

Memory management involves keeping track of which parts of memory are in use and allocating/deallocating memory space.

Memory Management Strategies

Contiguous Memory Allocation

- Each process occupies contiguous memory block

Fixed Partitioning

- Memory divided into fixed-size partitions
- **Internal Fragmentation**: Unused memory within partition

Dynamic Partitioning

- Partitions created dynamically
- **External Fragmentation**: Unused memory between partitions

Allocation Algorithms

1. **First Fit**: Allocate first hole large enough
2. **Best Fit**: Allocate smallest hole large enough
3. **Worst Fit**: Allocate largest hole

Segmentation

- Program divided into segments (code, data, stack)
- Each segment can be different size
- **Segment Table**: Maps segment to physical memory

Paging

- Physical memory divided into fixed-size **frames**
- Logical memory divided into same-size **pages**
- **Page Table**: Maps pages to frames

Page Table Structure

- **Page Number**: Index into page table
- **Offset**: Location within page
- **Frame Number**: Physical memory frame

Translation Lookaside Buffer (TLB)

- Cache for page table entries
- Speeds up address translation

Memory Protection

- **Base and Limit Registers**: Define legal address range
- **Hardware Protection**: MMU checks every memory access
- **Software Protection**: OS manages memory allocation

Virtual Memory

Concept

Virtual memory allows execution of processes not completely in memory, enabling:

- Programs larger than physical memory
- More programs in memory simultaneously
- Less I/O needed for loading/swapping

Demand Paging

- Pages loaded only when needed
- **Page Fault**: Reference to page not in memory
- **Lazy Loading**: Load page only when accessed

Page Fault Handling

1. Reference to invalid page causes trap
2. OS checks if reference is valid
3. Find free frame (or make one free)
4. Read page from storage into frame
5. Update page table
6. Restart instruction

Page Replacement Algorithms

First-In-First-Out (FIFO)

- Replace oldest page
- **Belady's Anomaly**: More frames may cause more page faults

Optimal Algorithm

- Replace page that will not be used for longest time
- Theoretical optimal, not practical

Least Recently Used (LRU)

- Replace page not used for longest time
- **Implementation**: Counter, stack, or approximation

LRU Approximation

- **Additional Reference Bits**: Track page usage
- **Second Chance**: Give page second chance if referenced

Allocation of Frames

- **Equal Allocation**: Each process gets same number of frames
- **Proportional Allocation**: Frames allocated based on process size
- **Global vs Local Replacement**: Process can replace pages from all processes or only its own

Thrashing

- Process spends more time paging than executing
- **Causes**: Too many processes, insufficient memory
- **Solutions**: Decrease multiprogramming, increase memory

Working Set Model

- **Working Set**: Set of pages used by process in time window
- **Principle of Locality**: Programs tend to use same pages repeatedly

---

File Systems

File Concept

A file is a collection of related information recorded on secondary storage.

File Attributes

- **Name**: Human-readable identifier
- **Identifier**: Unique number identifying file
- **Type**: Information about file format
- **Location**: Pointer to file location on device
- **Size**: Current file size
- **Protection**: Access permissions
- **Time, Date, and User ID**: Creation, modification, last use

File Operations

1. **Creating**: Allocate space, create directory entry
2. **Writing**: Write data to file
3. **Reading**: Read data from file
4. **Repositioning**: Change file position pointer
5. **Deleting**: Remove directory entry, free space
6. **Truncating**: Reset file to zero length

File Types

- **Regular Files**: Contain user data
- **Directory Files**: Contain information about other files
- **Special Files**: Represent devices, pipes, etc.

File Access Methods

Sequential Access

- Information accessed in order
- **Advantages**: Simple, efficient for sequential processing
- **Disadvantages**: Slow for random access

Direct Access (Random Access)

- Jump directly to any record
- **Advantages**: Fast access to any record
- **Disadvantages**: More complex

Indexed Access

- Index contains pointers to records
- **Advantages**: Fast search, efficient for large files
- **Disadvantages**: Extra space for index

Directory Structure

Single-Level Directory

- All files in one directory
- **Problems**: Naming conflicts, no organization

Two-Level Directory

- Separate directory for each user
- **Advantages**: Solves naming problem
- **Disadvantages**: No grouping capability

Tree-Structured Directory

- Hierarchical directory structure
- **Advantages**: Efficient searching, grouping
- **Disadvantages**: More complex

Acyclic-Graph Directory

- Shared subdirectories and files
- **Links**: Hard links, symbolic links
- **Problems**: Dangling pointers when files deleted

File System Implementation

File Allocation Methods

Contiguous Allocation

- Each file occupies contiguous blocks
- **Advantages**: Simple, fast access

- **Disadvantages**: External fragmentation, size limitations

Linked Allocation

- Each file is linked list of blocks
- **Advantages**: No external fragmentation
- **Disadvantages**: Slow random access, space overhead

Indexed Allocation

- Index block contains pointers to file blocks
- **Advantages**: Fast random access, no external fragmentation
- **Disadvantages**: Space overhead for index

Free Space Management

1. **Bit Vector**: Each bit represents block status
2. **Linked List**: Free blocks linked together
3. **Grouping**: Store addresses of free blocks
4. **Counting**: Store address and count of free blocks

File System Structure

- **Boot Control Block**: Information to boot OS
- **Volume Control Block**: Volume information
- **Directory Structure**: Organization of files
- **File Control Block**: Information about each file

I/O Systems

I/O Hardware

I/O Devices

- **Block Devices**: Store information in fixed-size blocks (disks)
- **Character Devices**: Accept/produce stream of characters (keyboards, mice)

Device Controllers

- Hardware component that controls I/O device
- Contains device driver interface
- **Device Registers**: Command, status, data

Memory-Mapped I/O

- Device registers mapped to memory addresses
- CPU uses normal memory instructions

Interrupts

- Device signals CPU when operation complete
- **Interrupt Vector**: Table of interrupt handler addresses
- **Interrupt Handler**: OS routine to handle interrupt

I/O Software Layers

Interrupt Handlers

- Handle hardware interrupts
- Save context, determine cause, call appropriate handler

Device Drivers

- Device-specific code
- Translate OS commands to hardware commands
- Handle device-specific error conditions

Device-Independent I/O Software

- Uniform interface for device drivers
- Buffering, error reporting, allocating/deallocating devices

User-Level I/O Software

- System calls, library procedures
- Spooling systems

I/O Scheduling

- **FCFS**: First-Come, First-Served
- **SSTF**: Shortest Seek Time First
- **SCAN**: Elevator algorithm
- **C-SCAN**: Circular SCAN
- **LOOK**: SCAN without going to ends

Buffering

- **Single Buffer**: One buffer in system space

- **Double Buffer**: Two buffers allow overlap
- **Circular Buffer**: Ring of buffers
- **Buffer Pool**: Set of buffers available to processes

Security and Protection

Security vs Protection

- **Protection**: Mechanisms to control access to resources
- **Security**: Defense against attacks from inside and outside

Authentication

Verifying identity of user or process.

Methods

1. **Something you know**: Password, PIN
2. **Something you have**: Token, smart card
3. **Something you are**: Biometrics

Password Security

- **Strong passwords**: Long, complex, unique
- **Password policies**: Expiration, history, complexity
- **Multi-factor authentication**: Combine methods

Access Control

Determining what authenticated users can do.

Access Control Matrix

- **Subjects**: Users, processes
- **Objects**: Files, devices, processes
- **Access Rights**: Read, write, execute, delete

Implementation

1. **Access Control Lists (ACL)**: List of permissions per object
2. **Capability Lists**: List of permissions per subject
3. **Role-Based Access Control (RBAC)**: Permissions based on roles

Security Attacks

Types of Attacks

1. **Malware**: Virus, worm, trojan horse, spyware
2. **Social Engineering**: Phishing, pretexting
3. **Denial of Service (DoS)**: Overwhelm system resources
4. **Man-in-the-Middle**: Intercept communications
5. **Buffer Overflow**: Exploit program vulnerabilities

Defense Mechanisms

1. **Firewalls**: Filter network traffic
2. **Intrusion Detection**: Monitor for suspicious activity
3. **Encryption**: Protect data confidentiality
4. **Digital Signatures**: Verify authenticity
5. **Sandboxing**: Isolate untrusted programs

Distributed Systems

Distributed System Characteristics

- **Resource Sharing**: Share hardware, software, data
- **Openness**: Extensible, interoperable
- **Concurrency**: Multiple processes executing simultaneously
- **Scalability**: System grows with demand
- **Fault Tolerance**: Continue operation despite failures
- **Transparency**: Hide distribution from users

Network Operating Systems

- Users aware of multiple machines
- Each machine runs own OS
- **Examples**: Windows networking, NFS

Distributed Operating Systems

- Users not aware of multiple machines
- System appears as single machine
- **Examples**: Amoeba, Plan 9

Distributed File Systems

- Files stored on multiple machines
- **Network File System (NFS)**: Client-server model
- **Andrew File System (AFS)**: Whole-file caching
- **Distributed File System (DFS)**: Replication, caching

Distributed Coordination

Distributed Mutual Exclusion

- **Centralized Algorithm**: Single coordinator
- **Distributed Algorithm**: All nodes participate
- **Token Ring**: Token passed around ring

Distributed Deadlock Detection

- **Wait-for graphs** distributed across nodes
- **Phantom deadlocks**: False deadlocks due to message delays

Leader Election

- **Bully Algorithm**: Highest ID wins
- **Ring Algorithm**: Token passed around ring

Distributed Systems Challenges

1. **Network Failures**: Partitions, message loss
2. **Partial Failures**: Some components fail
3. **Concurrency Control**: Coordinating distributed transactions
4. **Consistency**: Maintaining data consistency
5. **Security**: Authentication, authorization across network

---

Key Takeaways for Fundamental Understanding

Essential Concepts to Master

1. **Process Management**: Understand process lifecycle, scheduling, and IPC
2. **Memory Management**: Know how virtual memory and paging work
3. **File Systems**: Understand file organization and access methods
4. **Synchronization**: Master critical sections, semaphores, and deadlocks
5. **I/O Systems**: Know how OS manages hardware devices
6. **Security**: Understand authentication, authorization, and protection

Common Exam Topics

- Process scheduling algorithms and their trade-offs
- Memory management techniques (paging, segmentation)
- Deadlock prevention, avoidance, and detection
- File allocation methods and directory structures
- Synchronization mechanisms and classical problems
- Virtual memory concepts and page replacement algorithms

Practical Applications

- Understanding how your programs interact with the OS
- Optimizing application performance based on OS behavior
- Debugging system-level issues
- Designing efficient concurrent applications
- Understanding security implications of system design

Study Tips

1. **Practice with examples**: Work through scheduling, memory allocation problems
2. **Understand trade-offs**: Every design decision has pros and cons
3. **Learn by doing**: Use command-line tools, write simple programs
4. **Connect concepts**: See how different components interact
5. **Stay current**: OS concepts evolve with new hardware and requirements

This foundation will prepare you for advanced topics in operating systems and help you understand how modern systems work under the hood.