classmate
Date
Page

COMPLETE NOTES ON

Operating System...

Instagram : @ Curious_.programmer

Telegram : @ Curious_coder

Written by :-

Divya (CSE Student)

Index

Sr. no.	Chapters	Pg. no.
4.	File Systems and Storage Management 4.1. File Concepts and Operations. 4.2. File System Implementation 4.3. Disk management and RAID 4.4. File System Security and Access Control	15-18
5.	Input / Output Systems 5.1. I/O Devices and operations 5.2. I/O Buffering and caching. 5.3. Disk scheduling Algorithms 5.4. I/O Device Handling and interrupts	19-22
6.	Process Synchronization and Deadlocks. 6.1. Critical Section Problem 6.2. Semaphores and Mutexes 6.3. Deadlocks, Resource Allocation graph, Deadlock Prevention and Avoidance.	23-26

1. Introduction to Operating Systems

1.1. Definition of an operating System:

An operating system (OS) is a crucial software component that acts as an intermediary between computer hardware and software applications. It provides a consistent and user-friendly interface for users and applications to interact with the hardware. The OS manages hardware resources, executes and supervises applications, and ensures smooth coordination between various system components.

1.2. Function and Goals of an Operating System:

The primary functions of an operating system include:

1. Hardware Abstraction:
The OS hides the complex details of hardware from applications, enabling programmers to write software without needing to understand the intricacies of the underlying hardware.

2. Resource Management:
The OS allocates and manages hard-

ware resources such as CPU time, memory, storage and input/output devices. It ensures efficient utilization of resources while preventing conflicts.

3. Process Management:
The OS controls and schedules processes (~~re~~ running programs), allowing multitasking and ensuring fair access to CPU.

4. Memory Management:
The OS supervises memory allocation, tracking which parts of memory are in use and which are available. It handles tasks like memory allocation, deallocation and protection.

5. File System Management:
The OS manages files and directories, providing a hierarchial structure for data storage, access and organization.

6. Device Management:
The OS controls input and output devices, such as printers, disks, and network interfaces, to enable communication between applications and hardware.

1.3. Evolution of Operating systems:

Operating systems have evolved over several decades, from simple batch processing systems to complex, distributed environments.

Key stages in their evolution includes:

1. Batch operating Systems
2. Multiprogramming Systems
3. Time-sharing systems
4. Real-time Systems
5. Distributed Systems

1.4. Types of Operating Systems:

- **Batch Operating Systems:**

Handles scheduled jobs in batches without user interaction. Useful for repetitive tasks like payroll processing.

- **Multiprogramming Operating Systems:**

Manages multiple programs concurrently, optimizing CPU usage and reducing idle

time.

- **Time - Sharing Operating System:**

Supports multiple users interacting with the system simultaneously, offering quick response times and resource sharing.

- **Real-time Operating System:**

Designed for applications with strict timing requirements, ensuring tasks are completed within specific time limits.

- **Distributed Operating System:**

Manages resources across multiple interconnected computes enabling seamless communication and resource sharing.

2. Process Management

2.1. Process Concepts:

A process is an independent execution unit within an operating system. It represents a program in execution along with its associated resources such as memory, CPU time, open files and more.

Process management involves creating, executing, scheduling, and terminating processes.

1. Process State:

A process can be in different states such as "Running", "waiting", "Ready", or "terminated" based on its current activity and interactions with the system.

2. Process Control Block:

PCB is a data structure that holds essential information about a process, including its program counter, CPU registers, scheduling information, memory allocation, and more.

3. Context Switching:

When the operating system switches between processes, it performs a

context switch, saving the current process's state and loading the state of the next process to be executed.

Process Scheduling Algorithms:

Process Scheduling algorithms determine the order in which processes are executed by the CPU.

Common scheduling algorithm includes:

1. **First-Come, First-Served (FCFS):**
   Processes are executed in the order they arrive. Simple but may lead to longer average waiting times for processes.

2. **Shortest-Job first (SJF):**
   Processes with the shortest execution time are prioritied. Reduces average waiting time but can be difficult to predict execution times accurately.

3. **Round Robin (RR):**
   Each process is allocated a fixed time slice (quantum) of CPU time. After a quantum, the process is

moved to the end of the queue, allowing fair sharing of CPU time.

4. Priority Scheduling:
Processes are assigned priority levels, and the CPU is allocated to the highest priority queues. Can lead to priority inversion and starvation if not managed properly.

Interprocess Communication and Synchronisation:

Interprocess communication (IPC) refers to the method by which processes can do communication and share data with each other.

Synchronization mechanisms ensure that processes interact and coordinate without causing conflicts or inconsistencies. Key IPC mechanisms include:

1. Shared memory:
Processes can share a portion of memory to exchange data. Requires Synchronization mechanisms to avoid data corruption.

2. Message Passing:
Processes communicate by sending and receiving messages through the operating system. Can be either synchronous or asynchronous.

3. Semaphores:
Synchronization primitives that control access to resources. Used to prevent race conditions and ensure orderly execution.

Deadlocks and Prevention Mechanisms:

A deadlock occurs when two or more processes are unable to proceed because each is waiting for a resource held by another.

Prevention mechanisms aim to avoid deadlocks altogether.

Some strategies including:

1. Resource Allocation Graph:
A graphical representation of resources and processes that helps identify potential deadlocks.

2. Deadlock Prevention:
Ensuring that at least one of the necessary conditions for deadlock (mutual exclusion, ~~deadlo~~ hold and wait, no preemption, circular wait).

3. Deadlock Avoidance:
Using algorithms to determine whether resource allocation will lead to a deadlock, and only allocating resources if deadlock-free execution is gurant-eed.

4. Deadlock Detection and Recovery:
Periodically checking for deadlocks and taking corrective actions, such as killing processes or releasing resources.

3. Memory Management

3.1. Memory Hierarchy :

The memory hierarchy refers to the organisation of different types of memory in a computer system, ranging from fast but limited-capacity registers and cache to larger but slower main memory (RAM) and storage devices like hard drives and SSDs. The hierachy is designed to provide a balance between speed and capacity, allowing efficient data access for various tasks.

3.2 Virtual memory concepts :

Virtual memory is a memory management technique that extends the usable address space beyond the physical RAM capacity. It allows processes to access more memory than is physically available by using disk storage as an extension of main memory. This concept is based on the use of virtual addresses that are mapped to physical addresses by the operating system.

3.3. Paging and Segmentation :

Paging and segmentation are techniques

used in virtual memory systems:

1. Paging:

Memory is divided into fixed-size blocks called pages. Processes are divided into equally sized blocks called page frames. The mapping between virtual and physical addresses is maintained in a page table. Paging allow non-continguous allocation of memory and simplifies memory management.

2. Segmentation:

Memory is divided into segments, each representing a logical unit of a program, such as code, data, and stack. Each segment can have different sizes and attributes. Segmentation offers more flexibility but can lead to fragmentation.

3.4 Memory Allocation and Deallocation:

Memory allocation involves assigning portions of memory to processes when they are created, while deallocation involves reclaiming memory when processes are terminated or release memory they no longer need.

Two commonly memory allocation strategies are:

1. Contiguous Memory Allocation:

Processes are allocated continious blocks of memory. This can lead to fragmentation over time, both external (unused memory scattered) and internal (wasted memory within allocated blocks).

2. Dynamic Memory Allocation:

Allocates memory blocks are needed, which helps mitigate fragmentation ~~over time~~ issues. Techniques like pointers and memory allocation functions (e.g. 'malloc' in C) are used for dynamic allocation.

3.5 Page Replacement Algorithms:

Page replacement algorithms managed the limited physical memory by determining which pages should be swapped in and out of memory when needed.

Common algorithms include:

1. Least Recently used (LRU):

Evicts the page that has not been used for the longest time. Requires maintaining a history of page usage.

2. First-In-First-Out (FIFO):

Evicts the oldest page in memory. Simple but can suffer from the "Belady's anomaly" where increasing the number of page frames leads to worse performance.

3. Optimal:

Evicts the page that will not be used for the longest time in the future. Provides the best possible replacement but requires knowledge of future page accesses (which is not feasible in practice).

In summary, memory management involves organizing the computer's memory hierarchy, utilizing virtual memory to extend address space, allocating and deallocating memory for processes, and employing techniques like paging and segmentation for efficient

memory usage. Page replacement algorithms are crucial for managing memory in virtual memory systems to deciding which pages to swap in and out of physical memory.

4. File Systems and Storage Management

4.1. File Concepts and Operations:

A File System is a collection of related data that is stored on a storage device, such as a hard drive or SSD. Files can represent documents, programs, images, videos, and more. File operations include creating, reading, writing, updating and deleting files. The operating system provides an interface for users and applications to perform these operations.

4.2. File System Implementation:

File System implementation refers to the mechanisms used to organise and manage files on a storage device.

Key aspects include:

1. Directory Structure:

Directory (also known as folders) provide a hierarchial structure for organising files. Directories can contain files and subdirectories, creating a structured way to manage data.

2. File Allocation Methods:

Different methods are used to allocate space on storage devices for files. Common methods includes contingeuous allocation (storing files in contiguous blocks), linked allocation (using pointers to link blocks) and indexed allocation (using an index block to store pointers to data blocks).

4.3. Disk Management and Raid:

Disk management involves the organization and maintainance of physical storage devices (disks). RAID (Redundant Array of Independent disks) is a technology used to improve data storage performance, reliability, and fault tolerence by combining multiple disks into a single logical unit. Different RAID levels offer various combinations of performance and redundancy.

4.4. File System Security and Access Control:

File System security ensures that only authorized users and processess can

access and modify files. Access control mechanisms involve specifying permissions and restrictions for files and directories.

Key Security concepts include:

1. Authentication and Authorization:

Users must authenticate (prove their identity) to the system before being granted access. Authorization determines what actions users are allowed to perform files and directories based on their roles and permissions.

2. Access Control Lists (ACLs):

ACLs provide fine-grained control over file access by specifying permissions for individual users and groups.

3. File Ownership:

Each file is associated with an owner, who has certain privileges over the file. Ownership can be transferred and permissions can be adjusted.

4. Encryption:

File encryption protects data by converting it into an unreadable format. Only authorized users with the appropriate decrytion key can access the data.

5. Input/Output Systems

5.1. I/O Devices and Operations:

Input / Output (I/O) devices are hardware components that allow a computer to communicate with the external world. Examples include keyboards, mouse, displays, printers, disk drives, and network interfaces. I/O operations involves transferring data between these devices and the computer's memory.

There are two main types of I/O operations:

- blocking I/O (waits until the operation completes)

and

- non-blocking (I/O) (continues executing while waiting for I/O to complete)

5.2. I/O Buffering and Caching:

I/O buffering involves using intermediate memory areas (buffers) to temporarily store data during I/O operations. This helps reduce the impact of the speed difference between the CPU and I/O devices.

Caching involves storing frequently accessed data in a faster memory (cache) to speed up future access.

5.3. Disk Scheduling Algorithms:

Disk scheduling algorithms determine the order in which read and write requests are serviced on a disk drive.

Some Common algorithms include:

1. First-come, First-served (FCFS):

Processes requests in the order they arrive. Simple but can lead to poor performance due to the "SSTF" problem (Shortest Seek Time First).

2. Shortest Seek Time First (SSTF):

Services the request with the shortest distance between the current disk head position and the requested track. Can provide better response time compared to FCFS.

3. SCAN:

Moves the disk head in one direction, servicing requests along the way until it reaches the end of the disk. ~~head position~~ Then it reverses direction and repeats the process. Helps reduce the problem of starvation.

4. C-SCAN:

Similar to C-SCAN, but when reaching the end of the desk, the head immediately returns to the beginning of the disks. This reduces the time spent on seeking back.

5.4. I/O Device Handling and Interrupts:

I/O Device handling involves managing the communication between the CPU's and I/O devices. Interrupts are crucial for efficient I/O handling. When an I/O device has completed an operation or requires attention, it generates an interrupt, which interrupts the CPU's current execution.

The CPU then switches its attention to handle interrupt, often involves servicing the I/O operation or performing necessary actions to address the device's request.

In ~~sumo~~ summary, the Input/Output (I/O) subsystem of an operating system manages the interaction between the computer and external devices. I/O devices and operations facilitate data exchange, I/O buffering and caching optimize data transfer, disk scheduling algorithms ensure efficient disk access, and I/O device handling with interrupts enables efficient multitasking by allowing the CPU to respond to external events in a timely manner.

6. Process Synchronization and Deadlocks

6.1. Critical Section Problem:

The critical section problem refers to a scenario in which multiple processes or threads share a common resource (critical section) and must coordinate their access to avoid conflicts and ensure data integrity. The goal is to ensure that at any given time, only one process is executing within the ~~certain~~ critical section.

6.2. Semaphores and Mutexes:

Semaphores and mutexes are synchronization mechanisms used to control access to shared resources:

1. Semaphore:

A semaphore is an integer variable used to control access to a shared resource. It has two main operations: "wait" (decreamenting the semaphore) and "signal" (incrementing the semaphore). Semaphores can be used to implement various synchronization patterns.

2. Mutex (Mutual Exclusion Lock):

A mutex is a synchronization object that allow only one process/thread to access a resource at a time. It provides exclusive access and is often used to protect critical sections.

6.3. Deadlocks, Resource Allocation Graph, Deadlock Prevention and Avoidance:

A deadlock occurs when multiple processes are unable to proceed because each is waiting for a resource held by another.

Deadlock prevention aims to eliminate one or more of the four necessary conditions for deadlock:

1. Mutual Exclusion:

Preventing simultaneous access to resources.

2. Hold and Wait:

Requiring processes to request all needed resources at once.

3. No Preemption:

Allowing resources to be forcibly taken from processes.

4. Circular Wait:

Imposing a total order on resources and requiring processes to request resources in that order.

Deadlock avoidance involves using algorithms that dynamically allocate resources to processes in a way that prevents deadlocks. This may require predicting resource usage patterns and making allocations decisions accordingly.

6.4. Deadlock Detection and Recovery:

Deadlock detection involves periodically examining the resource allocation graph to determine if a deadlock has occured. If a deadlock is detected, is recovery actions can be taken.

1. Process Termination:

Killing some processes to release their held resources and break the ~~deadline~~ deadlock. Care must be taken to choose processes that ca be terminated safely.

2. Resource Preemption:

Forcibly reclaiming resources from one or more processes to resolve the deadlock. This can be complex and requires careful management.

7. CPU Scheduling

7.1. CPU scheduling Goals:

CPU scheduling is the process of determining which process / thread should run on the CPU at a given time. The primary goals of CPU scheduling are to maximize system throughput, minimize response time, ensure fairness, and efficiently utilize the CPU.

7.2. Scheduling Algorithms:

Scheduling Algorithms dictate how the operating system selects processes from the ready queue to execute on the CPU.

Different algorithms have various characteristics:

1. First-come, First-served (FCFS):

Executes processes in the order they arrive in the ready queue. Can lead to long waiting times for long processes, known as the "convoy effect."

2. Shortest Job First (SJF):

Selects the process with the smallest execution time next. Minimizes average waiting time but requires accurate estimates of process execution times.

3. Priority Scheduling:

Assigns priorities to processes, and the CPU is allocated to the highest priority ~~access~~ process. Can suffer from priority inversion and starvation if not managed carefully.

4. Round Robin (R.R):

Allocates each process a fixed time quantum, and then the CPU is switched to the next process in the queue. Provides fair sharing of CPU time but may not be optimal for certain workloads.

5. Multilevel Queue:

Divides the ready queue into multiple priority levels, with each level having its scheduling algorithm. Processes move between queues based on priority or scheduling policies.

7.3 Thread Scheduling and Multicore System:

In modern systems, the concept of threads is used to allow multiple execution flows within a single process. Thread scheduling involves managing thread execution on multiple CPU cores:

1. Cooperative Multithreading:

Threads yield control voluntarily, typically used in user-level threads libraries.

2. Preemptive Multithreading:

Threads are forcibly switched by the operating system, ensuring fair allocation of CPU time.

3. Multicore Systems:

Multiple CPU cores can execute threads concurrently. Thread scheduling aims to distribute threads across cores to maximize parallelism and system performance.

In summary, CPU scheduling is essential for managing the execution of processes and threads in an oper-

ating system. Different scheduling algorithms balance various goals such as throughput, response time, and fairness.

In modern systems, thread scheduling is important for utilizing multiple CPU cores effectively, allowing for ~~allowi~~ efficient multitasking and parallel execution.

8. Security and Protection

8.1 Security Threats and attacks:

Security threats and potential risks that can compromise the integrity, availability or confidentiality of computer systems and data.

Security attacks are intentional actions the exploit ~~ver~~ vulnerable / vulnerabilities to breach security.

Types of security threats and attacks include malware (viruses, worms, trojans), hacking, phishing, denial of services (DOS) attacks, data breaches, and social engineering.

8.2. Authentication and Authorization:

Authentication verifies the identity of users, processes, or systems trying to access resources. Common authentication methods include passwords, biometrics (fingerprint, facial recognition) and multi-factor authentication (requiring multiple proofs of identity).

Authorization determines what actions or resources a user, process, or system

is allowed to access based on their identity and privilages.

8.7

8.3. Access Control and Security Policies:

Access control involves restricting access to resources based on user roles, privileges, and policies.

Security policies define the rules and guidelines for securing a system. Mandatory access control (MAC) and discretionary access control (DAC) are models used to enforce access control. Role-based access control (RBAC) and attribute-based access control (ABAC) are common mechanisms for implementing access control policies.

8.4. Cryptography and Data Encryption:

Cryptography is the science of secure communication techniques, often involving the use of algorithms to encode and decode data.

Data encryption is a crucial technique to protect sensitive information during

storage and ~~int~~ transmission.

Symmetric encryption uses a single key for both encryption and decryption, while assymmetric encryption (public-key cryptography) uses a pair of keys:

a public key for encryption and a private key for decryption.

9. Distributed Systems

9.1. Characteristics of Distributed Systems:

Distributed systems are composed of multiple interconnected computers that work together as a unified system.

Key characteristics include:

1. Concurrent Processing:

Multiple processes can execute concurrently on different computers.

2. Transparency:

Users and applications perceive the distributed system as a single, coherent entity.

3. Scalability:

Distributed systems can be easily expanded by adding more computers.

4. Fault Tolerance:

Redundancy and fault recovery mechanisms enhance system reliability.

5. Heterogeneity:

Different hardware and software platforms may be used in a distributed system.

9.2. Communication Models:

Communication models define how processes in a distributed system exchange information.

Two Common models are:

1. Remote Procedure Calls (RPC):

Allows a process to execute a procedure or function on a remote computer as if it were a local call.

2. Sockets:

Provides a low-level communication interface between processes running on different computers, often used in networking applications.

9.3. Distributed File Systems:

Distributed file systems enable data sharing across multiple computers, offering a consistent and centralised way to access files stored on various machines.

Examples include NFS (Network File System) and DFS (Distributed File System).

9.4. Distributed Process Synchronization and Mutual Exclusion:

Synchronization and mutual exclusion are essential in distributed systems to ensure that processes cooperate and access shared resources correctly. Challenges arise due to network delays and the absence of a global clock. Mechanisms like distributed locks, timestamps, and logical clocks are used to manage synchronization and mutual exclusion.

In summary, distributed systems involves networks of interconnected computers working as a cohesive whole. They exhibit specific characteristics, utilize

communication models like RPC and sockets, implement distributed file systems for data sharing, and address challenging in distributed process synchronization and mutual exclusion to maintain data integrity and system coherence across the network.