Group 1: Introduction & OS Basics (Lectures 1–4)

Lecture 1: Purpose & Definition of an Operating System (OS)

What is an Operating System (OS)?

- An **Operating System (OS)** is a software that acts as an interface between the user and the computer hardware.
- It manages computer hardware and software resources and provides services for computer programs.

Why do we need an OS?

- Without an OS, we would have to manage every single aspect of the computer (memory, processes, etc.) manually, which would be nearly impossible.
- The OS simplifies this task and makes the computer usable for applications and users.

Key Functions of an OS:

1. **Process Management:** The OS manages processes (running programs), schedules them, and ensures they have the resources they need to run.
2. **Memory Management:** The OS handles memory allocation, ensuring that each process gets enough memory without overlapping or causing errors.
3. **File System Management:** It organizes data on disk, keeps track of files, directories, and permissions.
4. **Device Management:** The OS communicates with hardware devices like printers, hard drives, and monitors.
5. **Security & User Management:** It ensures that unauthorized users don't access the system and controls permissions.

Lecture 2: Types of Operating Systems & Interrupts

Types of Operating Systems:

1. **Batch OS:**
    - **Example:** Early mainframe computers.
    - No interaction with the user, tasks are grouped into batches and processed one at a time.

2. **Time-Sharing OS:**
    - **Example:** UNIX, Linux, Windows.
    - Allows multiple users to share system resources at the same time.
    - The OS rapidly switches between users to give the illusion of simultaneous access.
3. **Real-Time OS:**
    - **Example:** Embedded systems, robotics, industrial systems.
    - Guarantees the completion of tasks within a fixed time frame.

Interrupts:

- **Interrupts** are signals that inform the OS that it needs immediate attention, often due to an external event.
- **Types of Interrupts:**
    - **Hardware Interrupts:** Generated by hardware devices (e.g., keyboard press).
    - **Software Interrupts:** Generated by software programs (e.g., system calls).

Why Interrupts Matter:

- They allow the OS to handle urgent tasks without waiting for the current process to finish, making the system more responsive.

Lecture 3: OS Components, Services, and Structures

Core Components of an OS:

1. **Kernel:**
    - The **core** part of the OS. It manages the system's resources (CPU, memory, devices).
    - Runs in **privileged mode** with full control over hardware.
2. **Shell:**
    - An interface that allows users to interact with the OS using commands (like the command prompt or terminal).
    - It processes commands and sends them to the kernel for execution.
3. **System Calls:**
    - **System calls** are functions that programs use to request services from the OS (e.g., opening a file, creating a process).

- These calls allow the program to interact with the underlying hardware in a safe and controlled way.

OS Structures:

- **Monolithic Structure:** Everything (kernel, file system, device drivers) is in a single large block of code.
- **Microkernel Structure:** The OS is broken down into smaller modules, with only essential services running in the kernel. This makes the OS more modular and secure.

Lecture 4: UNIX/Linux Basics

UNIX/Linux Directory Structure:

- UNIX and Linux are similar operating systems with a hierarchical file system.

1. **Root ("/"):** The top level of the directory structure.
2. **Home ("~/"):** The directory where user files are stored.
3. **Bin ("/bin"):** Contains essential system binaries (commands).
4. **Etc ("/etc"):** Stores configuration files for system-wide settings.
5. **Var ("/var"):** Contains variable data, like logs and mail.

Basic UNIX/Linux Commands:

1. **ls:** Lists files in a directory.
2. **cd:** Changes the current directory.
3. **pwd:** Prints the current working directory.
4. **mkdir:** Creates a new directory.
5. **rmdir:** Removes an empty directory.
6. **man:** Displays manual pages for commands (e.g., man ls).

How to Use Them:

- Open a terminal (command line interface).
- Type these commands to interact with the system.

Review & Key Focus Areas:

- **What is an OS?** (Definition, Purpose)

- **Types of OS:** Batch, Time-Sharing, Real-Time
- **Interrupts:** Importance of interrupts, types, and how they work
- **OS Components:** Kernel, Shell, System Calls
- **UNIX/Linux Basics:** Basic file system structure, commands

---

Quick Recap for Exam:

- The **OS** is the backbone of your computer, providing essential services like managing processes, memory, files, and hardware.
- **Interrupts** are signals to the OS to handle urgent tasks, making the system more efficient.
- **UNIX/Linux** are operating systems with powerful command-line tools to interact with the system. Learn the basic commands to navigate and manage files.

Group 2: Process Management

Lecture 1: What is a Process?

Definition of a Process:

- A **process** is a **program** in execution. It is the instance of a program that is being executed by the computer's CPU. A process is dynamic, meaning it is a program that is currently running.

Process Components:

1. **Program Counter (PC):** Keeps track of the next instruction to be executed.
2. **Stack:** Holds the function calls, local variables, and the return address.
3. **Data Section:** Stores global variables and dynamically allocated memory.
4. **Heap:** A memory region used for dynamic memory allocation.
5. **Process Control Block (PCB):** Contains important information about the process, such as its state, priority, program counter, and registers.

Process State:

- A process goes through several states during its lifecycle:
    1. **New:** The process is being created.
    2. **Ready:** The process is waiting to be assigned to the CPU.

3. **Running:** The process is being executed by the CPU.
4. **Waiting (Blocked):** The process is waiting for some event (e.g., I/O operation) to complete.
5. **Terminated:** The process has finished execution.

Lecture 2: Process Scheduling & Scheduling Algorithms

What is Process Scheduling?

- **Process scheduling** is the mechanism by which the OS decides which process gets to use the CPU at any given time. The OS uses **scheduling algorithms** to allocate CPU time to processes.

Common Scheduling Algorithms:

1. **First-Come, First-Served (FCFS):**
    - Processes are executed in the order they arrive.
    - Simple but can lead to poor performance if a long process arrives first.
2. **Shortest Job Next (SJN):**
    - The process with the shortest next CPU burst is selected next.
    - Reduces waiting time but requires knowing the length of the next CPU burst, which is hard to predict.
3. **Round Robin (RR):**
    - Each process gets a small, fixed time slice (quantum) to run, after which it is interrupted and put back in the ready queue.
    - Good for time-sharing systems.
4. **Priority Scheduling:**
    - Processes are assigned priorities, and the one with the highest priority gets to run first.
    - Can lead to starvation (low-priority processes may never run).

Lecture 3: Process Synchronization & Deadlocks

What is Process Synchronization?

- **Process synchronization** is a mechanism to ensure that multiple processes or threads can operate concurrently without interfering with each other.
- It is crucial when processes share resources like memory, files, or I/O devices.

Critical Section Problem:

- When multiple processes access shared resources, we need to ensure that no two processes enter the **critical section** at the same time, causing data corruption.

Solutions to Synchronization:

- **Locks:** A way to enforce mutual exclusion. A process must acquire a lock before entering the critical section.
- **Semaphores:** Special variables used to control access to shared resources. They can be binary (0 or 1) or counting (more than 1).
- **Monitors:** High-level synchronization mechanism that automatically manages mutual exclusion.

Deadlock:

- A **deadlock** is a situation where two or more processes are blocked indefinitely because they are each waiting for the other to release a resource.

Conditions for Deadlock to Occur:

1. **Mutual Exclusion:** A resource can only be assigned to one process at a time.
2. **Hold and Wait:** A process holding one resource is waiting for another.
3. **No Preemption:** Resources cannot be forcibly removed from processes.
4. **Circular Wait:** A set of processes are waiting for each other in a circular chain.

Deadlock Prevention & Avoidance:

- **Deadlock Prevention:** Modify the system to eliminate one of the necessary conditions (e.g., disallow circular wait).
- **Deadlock Avoidance:** Use algorithms like **Banker's Algorithm** to decide whether a process can safely proceed based on available resources.

Lecture 4: Interprocess Communication (IPC)

What is IPC?

- **Interprocess Communication (IPC)** is a set of techniques that allows processes to communicate and share data.

Types of IPC:

1. **Shared Memory:**
    - Processes share a region of memory for communication.
    - Fast but requires synchronization to avoid conflicts.

2. **Message Passing:**
    - Processes communicate by sending and receiving messages.
    - More flexible but slower than shared memory.

IPC Mechanisms:

1. **Pipes:** A unidirectional communication channel used to transfer data between processes.
2. **Sockets:** Communication between processes on the same or different systems over a network.
3. **Signals:** A way for processes to send notifications to other processes (e.g., terminating a process).

Review & Key Focus Areas:

- **What is a Process?** (Lifecycle, Components)
- **Scheduling Algorithms:** FCFS, SJN, Round Robin, Priority Scheduling
- **Process Synchronization:** Critical Section, Semaphores, Monitors
- **Deadlock:** Conditions, Prevention, Avoidance
- **IPC:** Shared Memory, Message Passing, Pipes, Sockets

Quick Recap for Exam:

- A **process** is a running program, and its state can change through various stages (New, Ready, Running, Waiting, Terminated).
- **Scheduling algorithms** decide which process gets CPU time: FCFS, SJN, RR, and Priority Scheduling.
- **Synchronization** ensures that processes don't interfere with each other when accessing shared resources.
- **Deadlock** is when processes get stuck waiting for each other—prevention and avoidance strategies are key.
- **IPC** is essential for processes to communicate, either through shared memory or message passing.

Group 3: Memory Management

Lecture 1: Introduction to Memory Management

What is Memory Management?

- **Memory management** refers to the process of efficiently managing the computer's memory, ensuring that every program or process has enough memory to run properly and that memory is allocated and deallocated when needed.

Types of Memory:

1. **Primary Memory (RAM):**
   - This is the main memory that is directly accessed by the CPU. It is volatile (loses data when the power is off).
2. **Secondary Memory (Disk Storage):**
   - This is non-volatile memory, where data is stored permanently (e.g., hard drives, SSDs).
3. **Cache Memory:**
   - This is a small, fast memory that stores frequently used data to speed up processing.

Memory Management in OS:

- The OS is responsible for managing memory, ensuring that processes have their own memory space, and preventing them from interfering with each other. The key goals are:
  1. **Fairness:** Allocating memory to processes fairly.
  2. **Efficiency:** Making sure memory is used optimally.
  3. **Isolation:** Ensuring processes don't overwrite each other's memory.

Lecture 2: Contiguous Memory Allocation

What is Contiguous Memory Allocation?

- In **contiguous memory allocation**, each process is allocated a single, contiguous block of memory.

Advantages:

- **Simplicity:** Easy to implement and manage.
- **Performance:** Direct access to the memory without complex management.

Disadvantages:

- **Fragmentation:** Over time, free memory can be fragmented (small holes), making it hard to allocate large blocks.
- **Fixed Size:** Processes are limited to fixed-size blocks of memory.

Example:

- If there are 10MB free memory blocks and a process needs 5MB, it will be allocated the 5MB, but once it finishes, the free block might be fragmented and not usable for larger processes.

Lecture 3: Paging and Virtual Memory

What is Paging?

- **Paging** is a memory management scheme that eliminates the need for contiguous memory allocation by dividing physical memory into small, fixed-size blocks called **pages**.

How Paging Works:

- The process is divided into **pages**, and physical memory is divided into **page frames** of the same size.
- The **Page Table** is used to map logical pages to physical page frames.

Advantages of Paging:

1. **No External Fragmentation:** Since processes can be placed in any available page frame, it eliminates external fragmentation.
2. **Efficient Memory Use:** More processes can fit into memory.

Virtual Memory:

- **Virtual memory** allows processes to use more memory than what is physically available by swapping data between physical memory (RAM) and disk storage (using **page swapping**).
- **Swap Space**: A section of disk used to store pages when they are not in use.

Lecture 4: Segmentation

What is Segmentation?

- **Segmentation** is similar to paging but instead of dividing memory into fixed-size pages, it divides memory into variable-sized **segments** based on logical divisions like code, data, and stack.

How Segmentation Works:

- Each segment has a **segment number** and an **offset** (location within the segment).
- The **Segment Table** maps each segment to its physical location in memory.

Advantages of Segmentation:

1. **Logical Structure:** Segmentation allows the memory to be divided logically, making it easier for users to understand (e.g., code, stack, data).
2. **Dynamic Allocation:** The size of segments can vary, reducing waste compared to fixed-size pages.

Disadvantages:

1. **Fragmentation:** Internal fragmentation can occur when segments leave unused gaps.
2. **Complexity:** Managing segments can be more complicated than paging.

Review & Key Focus Areas:

1. **Memory Management:**
    - OS's responsibility to manage memory efficiently, ensuring processes have the memory they need without interfering with each other.
2. **Contiguous Allocation:**
    - Simple but can lead to fragmentation and inefficiency.
3. **Paging:**
    - Divides memory into pages and page frames to avoid fragmentation, uses a page table for mapping.
4. **Virtual Memory:**
    - Uses disk storage to extend memory beyond physical limits, swapping pages in and out.
5. **Segmentation:**
    - Divides memory into segments based on logical divisions, offers more flexibility than paging but can lead to fragmentation.

Quick Recap for Exam:

- **Memory Management** ensures efficient allocation of memory to processes.
- **Contiguous Allocation** is easy but suffers from fragmentation.
- **Paging** eliminates external fragmentation and allows better memory usage.
- **Virtual Memory** lets you use more memory than available by swapping data between RAM and disk.
- **Segmentation** divides memory based on logical sections but can be more complex to manage.

Group 4: Process Management

Lecture 1: Process Concept

What is a Process?

- A **process** is a program in execution. It consists of the program code, its current activity, and the resources needed for its execution (such as memory, CPU, and I/O devices).

Process vs Program:

- A **program** is a passive entity (just a set of instructions) while a **process** is an active entity (program in execution).

States of a Process:

1. **New:** The process is being created.
2. **Ready:** The process is waiting to be assigned to the CPU.
3. **Running:** The process is being executed by the CPU.
4. **Waiting (Blocked):** The process is waiting for an event (like I/O).
5. **Terminated:** The process has finished execution.

Process Control Block (PCB):

- The PCB is a data structure that contains information about the process, such as:
  - Process ID (PID)
  - Process state
  - CPU register values
  - Memory management information
  - I/O status

Lecture 2: Process Scheduling

What is Process Scheduling?

- **Process Scheduling** is the method used by the OS to manage which process gets to use the CPU and for how long.

Types of Schedulers:

1. **Long-Term Scheduler:** Decides which processes are admitted to the system.

2. **Short-Term Scheduler (CPU Scheduler):** Decides which process gets the CPU next.
3. **Medium-Term Scheduler:** Moves processes between the ready queue and disk to balance load.

Scheduling Algorithms:

1. **First-Come, First-Served (FCFS):**
    - Processes are executed in the order they arrive.
    - Simple but can lead to poor performance (long processes delay others).
2. **Shortest Job Next (SJN):**
    - The process with the shortest burst time is executed next.
    - Optimizes waiting time but can lead to starvation (long processes may never get executed).
3. **Round Robin (RR):**
    - Each process gets a fixed time slice (quantum) to run.
    - Fairer and prevents starvation but can cause high turnaround time.
4. **Priority Scheduling:**
    - Processes are assigned priority, and the process with the highest priority is executed first.
    - Starvation is possible if low-priority processes are always waiting.

Lecture 3: Inter-Process Communication (IPC)

What is Inter-Process Communication?

- **IPC** is the mechanism that allows processes to communicate and synchronize with each other. It is essential for coordinating multiple processes running concurrently.

IPC Mechanisms:

1. **Message Passing:**
    - Processes exchange messages to communicate.
    - Can be **synchronous** (sender waits for the receiver's response) or **asynchronous** (sender doesn't wait).
2. **Shared Memory:**
    - Processes communicate by reading and writing to shared memory.
    - Requires synchronization to prevent data corruption.

Synchronization:

- Ensures that concurrent processes do not interfere with each other while accessing shared resources.

Lecture 4: Deadlock

What is Deadlock?

- **Deadlock** is a situation where two or more processes are blocked forever, waiting for each other to release resources.

Conditions for Deadlock:

1. **Mutual Exclusion:** At least one resource must be held in a non-shareable mode.
2. **Hold and Wait:** A process holding one resource is waiting to acquire additional resources held by other processes.
3. **No Preemption:** Resources cannot be preempted; they must be released voluntarily by the process holding them.
4. **Circular Wait:** A set of processes is waiting for resources in a circular chain.

Deadlock Prevention:

- Prevent one of the four necessary conditions from happening.
    1. **Eliminate Mutual Exclusion:** This is difficult, as some resources (like printers) cannot be shared.
    2. **Eliminate Hold and Wait:** Processes must request all resources at once.
    3. **Eliminate No Preemption:** Allow resources to be taken from a process if necessary.
    4. **Eliminate Circular Wait:** Enforce a partial order of resource requests.

Deadlock Detection:

- The OS can periodically check for deadlocks and take corrective actions (like terminating processes or rolling back operations).

Deadlock Avoidance:

- The OS uses algorithms (like the **Banker's Algorithm**) to decide whether resource allocation can be made without causing a deadlock.

Review & Key Focus Areas:

1. **Process Concept:**

    - A process is a program in execution, and the OS manages its states and resources.
2. **Process Scheduling:**
    - The OS uses scheduling algorithms like FCFS, SJN, RR, and Priority Scheduling to manage CPU time.
3. **Inter-Process Communication (IPC):**
    - IPC mechanisms like message passing and shared memory allow processes to communicate and synchronize.
4. **Deadlock:**
    - Deadlock occurs when processes are stuck in a circular waiting situation, and deadlock prevention, detection, and avoidance mechanisms are employed to handle it.

Quick Recap for Exam:

- **Process:** Program in execution, with different states (New, Ready, Running, Waiting, Terminated).
- **Scheduling Algorithms:**
    - FCFS: First come, first served.
    - SJN: Shortest job first.
    - RR: Round Robin with time slices.
    - Priority: Processes with higher priority run first.
- **IPC:** Used for communication between processes, using message passing or shared memory.
- **Deadlock:** Occurs when processes are blocked indefinitely; can be prevented, detected, or avoided by the OS.

Group 5: Memory Management

Lecture 1: Memory Management Concept

What is Memory Management?

- **Memory management** refers to the OS's method of managing the computer's memory. It involves controlling and coordinating the allocation and deallocation of memory resources to different processes, ensuring efficient use of memory.

Memory Hierarchy:

- The **memory hierarchy** describes different types of memory based on speed and size:
    - **Registers** (fastest, smallest)
    - **Cache Memory**
    - **Main Memory (RAM)**
    - **Secondary Storage (Disk)**

Main Tasks in Memory Management:

1. **Allocation of Memory:** Assign memory to processes and ensure efficient memory use.
2. **Tracking Memory Usage:** Keep track of memory usage and ensure that no process exceeds its allocated space.
3. **Deallocation of Memory:** Free memory once a process terminates.
4. **Protection and Sharing:** Prevent processes from accessing unauthorized memory areas.

Lecture 2: Contiguous Memory Allocation

What is Contiguous Memory Allocation?

- In **contiguous memory allocation**, each process is allocated a single contiguous block of memory in RAM.

Advantages:

- Simple and easy to implement.
- Efficient for small systems where fragmentation is not a concern.

Disadvantages:

- **External Fragmentation:** As processes are allocated and deallocated memory, free memory spaces are scattered throughout, causing fragmentation.
- **Internal Fragmentation:** Some allocated memory might not be used by the process, wasting memory within allocated blocks.

Partitioning Techniques:

1. **Fixed Partitioning:**
    - The memory is divided into fixed-size partitions, and each process is assigned to one partition.
    - Simple, but can lead to wasted space (if a process is smaller than a partition).
2. **Dynamic Partitioning:**

    - Partitions are created dynamically based on the process's memory needs.
    - Reduces internal fragmentation but can cause external fragmentation.

Lecture 3: Paging

What is Paging?

- **Paging** is a memory management scheme that eliminates the problem of contiguous memory allocation by dividing both physical memory (RAM) and logical memory (process memory) into small fixed-size blocks, called **pages** and **frames** respectively.

How Paging Works:

1. The OS divides the physical memory into **fixed-size blocks** called **frames**.
2. The process's logical memory is divided into **pages**, and each page is mapped to a frame in physical memory.

Advantages of Paging:

- **No External Fragmentation:** Since pages can be scattered in memory, there's no wasted space.
- **Efficient Use of Memory:** The OS can use the available memory more efficiently by assigning only the required number of frames to each process.

Page Table:

- The **page table** is a data structure used to store the mapping of pages to frames.

Lecture 4: Segmentation

What is Segmentation?

- **Segmentation** is a memory management technique where a process is divided into **variable-sized segments** such as code, data, and stack.

How Segmentation Works:

1. The process is divided into segments of different sizes (e.g., code, stack, heap).
2. Each segment is assigned to a different area in memory, and the OS keeps track of the segment's base address and size.

Advantages of Segmentation:

- **Logical Segmentation:** Each segment corresponds to a logical unit of a program (e.g., code, stack, data).
- **Easy to share:** Segments can be shared between processes (e.g., shared libraries).

Disadvantages of Segmentation:

- **External Fragmentation:** Over time, the memory may become fragmented as segments are loaded and unloaded.
- **Complexity:** Managing segments can be more complex than paging.

Lecture 5: Virtual Memory

What is Virtual Memory?

- **Virtual memory** is a memory management technique that gives an "idealized abstraction" of the storage resources that are actually available on a given machine. It allows processes to behave as if they have their own contiguous block of memory, regardless of the actual physical memory layout.

How Virtual Memory Works:

- Virtual memory uses both **paging** and **segmentation** to provide processes with the illusion of having more memory than physically available.

Components of Virtual Memory:

1. **Swap Space:** When physical memory is full, the OS moves less-used pages to the disk (swap space) and brings them back into memory when needed.
2. **Page Table:** Maps virtual addresses to physical memory locations.

Advantages of Virtual Memory:

- **More Memory:** Processes can use more memory than is physically available.
- **Isolation:** Each process is isolated from others, preventing them from accessing each other’s memory.

Disadvantages of Virtual Memory:

- **Page Faults:** Occur when a process accesses a page that is not in physical memory. The OS must retrieve the page from disk, which is slow.
- **Overhead:** The use of disk space and the overhead of managing paging can slow down the system.

Review & Key Focus Areas:

1. **Memory Management:** Controls allocation, tracking, and deallocation of memory to processes, ensuring efficient usage.
2. **Contiguous Memory Allocation:** Memory is allocated in contiguous blocks, with issues of fragmentation.

3. **Paging:** Eliminates fragmentation by using fixed-size pages and frames, with a page table to map logical to physical memory.
4. **Segmentation:** Divides a process into variable-sized segments (e.g., code, data, stack), with benefits and problems similar to paging.
5. **Virtual Memory:** Extends the available memory using disk storage, allowing processes to access more memory than physically available.

Quick Recap for Exam:

- **Memory Management:** Controls and allocates memory to processes.
- **Contiguous Allocation:** Processes are allocated contiguous memory blocks.
- **Paging:** Divides memory into fixed-size pages and eliminates fragmentation.
- **Segmentation:** Divides a process into segments, but may cause fragmentation.
- **Virtual Memory:** Uses disk space to extend physical memory, allowing processes to use more memory.

Group 6: Process Scheduling

Lecture 1: What is Process Scheduling?

Definition of Process Scheduling:

- **Process scheduling** refers to the method by which the OS decides which process will run at any given time. This is crucial for multi-tasking, where the CPU must switch between various processes.

Types of Processes:

1. **CPU-bound processes:** Processes that require heavy CPU usage (e.g., calculations, data processing).
2. **I/O-bound processes:** Processes that require a lot of input/output operations (e.g., file handling, network communication).

Scheduling Objectives:

1. **Fairness:** Ensuring that all processes get a fair share of CPU time.
2. **Efficiency:** Maximizing CPU utilization.
3. **Response Time:** Minimizing the time from when a request is made until the system responds.

4. **Turnaround Time:** Minimizing the total time a process takes to complete, including waiting and execution time.

Lecture 2: Scheduling Algorithms

Overview:

Scheduling algorithms determine the order in which processes are executed by the CPU. Here are the most common ones:

1. **First-Come-First-Served (FCFS):**
    - **Description:** The first process to arrive gets executed first.
    - **Advantages:** Simple to implement.
    - **Disadvantages:** Can lead to **convoy effect** (long processes cause short processes to wait for a long time).
2. **Shortest Job First (SJF):**
    - **Description:** The process with the shortest burst time is selected for execution.
    - **Preemptive version: Shortest Remaining Time First (SRTF).**
    - **Advantages:** Minimizes average waiting time.
    - **Disadvantages:** Difficult to predict burst time, especially in a real-time system.
3. **Round Robin (RR):**
    - **Description:** Each process is given a small unit of time (quantum) in a cyclic order.
    - **Advantages:** Simple and fairly good for time-sharing systems.
    - **Disadvantages:** If the time quantum is too large, it becomes FCFS; too small, it leads to overhead.
4. **Priority Scheduling:**
    - **Description:** Each process is assigned a priority, and the process with the highest priority is executed first.
    - **Preemptive version: Preemptive Priority Scheduling.**
    - **Advantages:** Can be customized for different types of tasks (e.g., critical tasks get higher priority).
    - **Disadvantages:** Can lead to **starvation** of lower-priority processes (they never get CPU time).
5. **Multilevel Queue Scheduling:**

  - **Description:** Processes are divided into different queues based on priority or other factors, with each queue having its own scheduling algorithm.
  - **Advantages:** Allows the system to handle different types of processes efficiently.
  - **Disadvantages:** Can be complex to manage.

Lecture 3: Multilevel Feedback Queue Scheduling

What is Multilevel Feedback Queue Scheduling?

- **Multilevel Feedback Queue (MFQ)** is an advanced scheduling algorithm that combines multiple queues with dynamic priorities.
  - **Multiple Queues:** Each queue has a different priority, and processes are moved between queues based on their behavior.
  - **Dynamic Adjustment:** Processes that use more CPU time are moved to lower-priority queues, while processes that frequently perform I/O operations move to higher-priority queues.

Advantages:

- Balances CPU-bound and I/O-bound processes.
- Helps in improving overall system performance.
- Reduces the risk of starvation by adjusting priorities dynamically.

Lecture 4: Context Switching

What is Context Switching?

- **Context switching** is the process of saving the state of a currently running process and loading the state of the next process to execute. This is done by the **scheduler** to ensure that multiple processes can share the CPU.

Steps in Context Switching:

1. **Save the current process state:** The OS saves the CPU registers, program counter, and stack pointer of the currently running process.
2. **Update process state:** The OS updates the state of the current process (e.g., switching it to the "waiting" or "ready" state).
3. **Load the next process state:** The OS loads the state of the next process to execute, including the program counter, stack pointer, etc.

Impact of Context Switching:

- **Overhead:** Frequent context switching leads to a significant overhead because the CPU must save and restore process states.
- **Efficiency:** While context switching enables multitasking, it must be managed efficiently to avoid excessive overhead.

Lecture 5: Scheduling in Real-Time Systems

What is Real-Time Scheduling?

- **Real-time scheduling** refers to scheduling algorithms designed for real-time systems where tasks have strict timing constraints.

Types of Real-Time Scheduling:

1. **Hard Real-Time Scheduling:**
    - Tasks must be completed within a strict deadline. Missing the deadline can result in catastrophic consequences (e.g., in embedded systems like pacemakers).
2. **Soft Real-Time Scheduling:**
    - Tasks have deadlines, but missing them occasionally may be tolerable. It is often used in multimedia systems or telecommunications.

Real-Time Scheduling Algorithms:

1. **Rate Monotonic Scheduling (RMS):**
    - Assigns priorities based on the period of tasks; shorter tasks get higher priorities.
    - **Preemptive.**
2. **Earliest Deadline First (EDF):**
    - Assigns priorities based on the nearest deadline.
    - **Dynamic priority.**

Challenges in Real-Time Scheduling:

- Ensuring that tasks meet their deadlines, especially in systems with limited resources.
- Minimizing task starvation and ensuring fairness.

Review & Key Focus Areas:

1. **Process Scheduling:** Refers to how the OS allocates CPU time to processes.
2. **Scheduling Algorithms:**

   - **FCFS, SJF, RR, Priority Scheduling:** Each with its own pros and cons.
   - **Multilevel Queue and MFQ:** Used for handling different types of tasks effectively.
3. **Context Switching:** The mechanism of saving and loading process states for multitasking.
4. **Real-Time Scheduling:** Special scheduling for systems with time constraints, using algorithms like RMS and EDF.

Quick Recap for Exam:

- **Process Scheduling** determines the order of process execution.
- **FCFS:** Processes are scheduled in the order they arrive.
- **SJF:** Shortest job first, minimizing average waiting time.
- **Round Robin:** Time-sharing system with fixed time slices.
- **Priority Scheduling:** Processes assigned priorities.
- **Multilevel Feedback Queue:** Dynamic adjustment based on process behavior.
- **Context Switching:** Saves and restores process states.
- **Real-Time Scheduling:** Ensures that tasks meet strict timing requirements.

QUICK RECAP

Group 1: Introduction & OS Basics

1. **What is OS?** – Interface between user and hardware.
2. **Types of OS** – Batch, Time-sharing, Real-time, etc.
3. **OS Components** – Kernel, Shell, System Calls, Services.
4. **UNIX/Linux Basics** – Commands like ls, pwd, cd, etc.

⚙️ **Group 2: Process Management Basics**

5. **What is a Process?** – Running program.
6. **Process States** – New, Ready, Running, Waiting, Terminated.

7. **Process Control Block (PCB)** – Data structure for process info.
8. **Context Switching & Threads** – Switching processes, intro to threads.

🔄 **Group 3: Synchronization**

9. **Process Sync Intro** – Need for managing concurrent processes.
10. **Critical Section Problem** – Problem and conditions.
11. **Peterson's Solution & Mutex** – Software/hardware solutions.
12. **Semaphores & Deadlocks** – Signaling mechanisms, intro to deadlocks.

❌ **Group 4: Deadlocks**

13. **Deadlock Basics** – Conditions & examples.
14. **Deadlock Prevention** – Breaking deadlock conditions.
15. **Deadlock Avoidance** – Banker's Algorithm.
16. **Deadlock Detection & Recovery** – Wait-for graph, recovery.

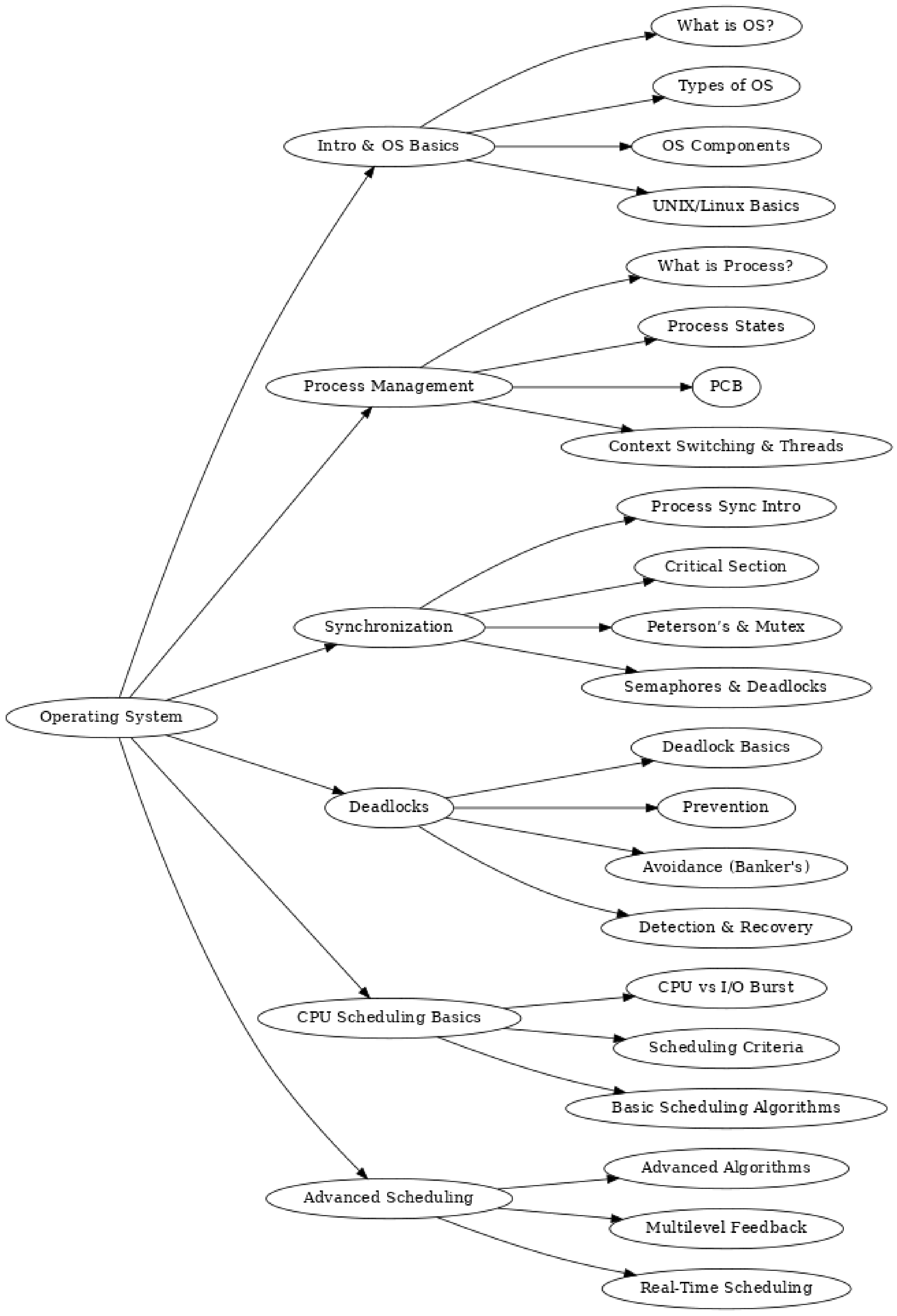
🧠 **Group 5: CPU Scheduling Basics**

17. **CPU vs I/O Burst** – Process execution characteristics.
18. **Scheduling Criteria** – Efficiency metrics (turnaround, response time, etc.).
19. **Types of Scheduling Algorithms** – FCFS, SJF, RR, Priority.

🧮 **Group 6: Advanced Scheduling**

20. **Advanced Algorithms** – Multilevel Queue Scheduling.
21. **Multilevel Feedback Queue** – Dynamic priorities.
22. **Real-Time Scheduling** – Hard vs Soft real-time, EDF.

MIND MAP


Operating System
Intro & OS Basics
What is OS?
Types of OS
OS Components
UNIX/Linux Basics
Process Management
What is Process?
Process States
PCB
Context Switching & Threads
Synchronization
Process Sync Intro
Critical Section
Peterson's & Mutex
Semaphores & Deadlocks
Deadlocks
Deadlock Basics
Prevention
Avoidance (Banker's)
Detection & Recovery
CPU Scheduling Basics
CPU vs I/O Burst
Scheduling Criteria
Basic Scheduling Algorithms
Advanced Scheduling
Advanced Algorithms
Multilevel Feedback
Real-Time Scheduling