Operating system notes

1. Introduction to Operating Systems

- **Operating System (OS)**: A software that manages computer hardware and software resources and provides services for computer programs. It acts as an intermediary between users and the computer hardware.
- **Classification of Operating Systems**:
    - **Batch OS**: Executes batches of jobs with minimal user interaction (e.g., early mainframe systems).
    - **Interactive OS**: Allows users to interact with the system directly (e.g., Windows, Linux).

1. Multiprogramming (Multiple Programs Sharing the CPU):

- **What is it?** Multiprogramming is when multiple programs are in the computer's memory at the same time. The operating system switches between them to keep the CPU busy.
- **Why do we need it?** To make sure the CPU isn’t wasting time. When one program is waiting (like for input/output), the CPU can work on another program.
- **Example**: Think of it like watching a YouTube video while waiting for a file to download. If the video (program) pauses to buffer, you can still browse other sites (another program) in the meantime.

Key Point: Multiprogramming doesn't execute programs simultaneously; it just switches between them to keep the CPU from being idle.

---

2. Multitasking (Multiple Tasks Happening at Once):

- **What is it?** Multitasking means running multiple tasks (or programs) at the same time. The computer gives each task a little bit of time so it feels like they’re happening simultaneously.
- **How does it work?** The CPU switches between tasks very quickly, so we don't notice any delay. This is also known as "time-sharing."
- **Example**: You’re writing in a Word document while listening to music on a media player. The CPU switches between the Word application and the music player so fast, it seems like both are running at the same time.

Key Point: Multitasking creates an illusion that many tasks are happening together, but the CPU is just switching between them really quickly.

---

3. Multiprocessing (Multiple CPUs Working Together):

- **What is it?** Multiprocessing is when a computer has more than one CPU (or core), and they work together to run multiple programs or processes at the same time.
- **Why is it important?** It allows true parallel execution of tasks, meaning different tasks can actually run at the same time on different processors.
- **Example**: In modern laptops or smartphones, you often have multi-core processors. So one core might handle your game while another core is managing background apps like email.

Key Point: Multiprocessing means real parallelism – tasks are actually running at the same time on different processors.

How to Remember:

- **Multiprogramming** = Switching between programs to keep the CPU busy (like waiting for a file to download while watching a video).
- **Multitasking** = Rapidly switching between tasks, making it feel like they're happening at the same time (like typing and listening to music).
- **Multiprocessing** = Using multiple CPUs to do multiple things at the same time (like having two people work on separate parts of a task).

1. Real-Time Operating Systems (RTOS):

What is an RTOS?

- **Definition**: A Real-Time Operating System (RTOS) is designed to handle tasks with strict timing constraints. It ensures that critical tasks are executed within a certain time frame.
- **Purpose**: It's used in systems where timely processing is crucial, such as in embedded systems, medical devices, or industrial controls.

How Does It Work?

- **Deterministic Behavior**: An RTOS guarantees that certain operations will be completed within a predictable amount of time. This is crucial for applications where delays could lead to failures or dangerous situations.
- **Task Scheduling**: RTOS uses specialized scheduling algorithms (like priority-based scheduling) to ensure high-priority tasks get the CPU time they need as soon as possible.

Examples of RTOS:

- **VxWorks**, **RTEMS**, and **FreeRTOS** are popular real-time operating systems used in various critical applications.

Example Scenario:

- **Medical Devices**: An RTOS in a heart monitor needs to process data and trigger alarms in real-time to ensure patient safety. Any delay could have serious consequences.

2. Multithreading Operating Systems:

What is Multithreading?

- **Definition**: Multithreading refers to the ability of an operating system to manage multiple threads within a single process. A thread is a smaller unit of a process that can be executed independently.
- **Purpose**: It allows a program to perform multiple operations simultaneously, improving performance and responsiveness.

How Does It Work?

- **Threads**: Each thread in a process shares the same memory space but executes different tasks. Threads can run concurrently and are managed by the operating system.
- **Context Switching**: The OS handles switching between threads, so they appear to run simultaneously. This involves saving the state of one thread and loading the state of another.

Examples of Multithreading:

- **Web Browsers**: Modern browsers use multithreading to handle tasks like loading web pages, rendering graphics, and running scripts concurrently.
- **Games**: Games use multiple threads for rendering graphics, handling user input, and managing game logic simultaneously.

Example Scenario:

- **Word Processing**: While you're typing in a document, a spell checker might run in the background on a different thread, allowing you to continue working without interruption.

Key Points to Remember:

- **RTOS**:
  - Guarantees timely execution of tasks.
  - Used in systems where delays are unacceptable.
  - Example: Embedded systems in medical devices.

- **Multithreading**:
    - Allows multiple threads to run within a single process.
    - Improves performance by doing multiple tasks at once.
    - Example: Web browsers handling multiple tabs and processes.

1. System Protection

What is it?

- **Definition**: System protection ensures that the operating system safeguards its resources and data from unauthorized access or corruption.

How Does It Work?

- **User Authentication**: Checks if users are allowed to access the system (like logging in with a username and password).
- **Access Control**: Controls what actions users or programs can perform (like read/write permissions on files).
- **Isolation**: Keeps processes and data separate to prevent interference.

Example: Think of it like a secure building where only authorized personnel can enter certain rooms and everyone's activities are monitored.

---

2. System Calls

What is it?

- **Definition**: System calls are requests made by programs to the operating system to perform tasks that the program can't do directly (like accessing hardware or managing files).

How Does It Work?

- **Interface**: Provides a way for programs to interact with the OS.
- **Examples**: Reading a file, creating a new process, or sending data over the network.

Example: If you want to save a document, the program uses a system call to ask the OS to write the data to the disk.

---

3. Reentrant Kernels

What is it?

- **Definition**: A reentrant kernel can handle multiple tasks or requests without getting confused or corrupted by interruptions.

How Does It Work?

- **Safe Reentry**: The kernel's code can be interrupted and safely resumed, ensuring consistency.

Example: Imagine a chef who can handle multiple orders at once without mixing up the recipes.

---

4. Operating System Structure

Layered Structure:

- **Definition**: The OS is divided into layers, each with specific functions. Lower layers handle hardware, while upper layers provide services to users and applications.

Monolithic Systems:

- **Definition**: All OS functions run in a single large block of code that directly interacts with hardware.
- **Example**: Traditional Unix.

Microkernel Systems:

- **Definition**: Only essential services run in the kernel; other services run in user space.
- **Example**: Minix, QNX.

Example:

- **Layered Structure**: Like a multi-story building where each floor has different functions (e.g., utilities on the ground floor, offices above).
- **Monolithic**: A single large, complex machine where everything happens together.
- **Microkernel**: A small, focused machine with add-ons that handle additional tasks separately.

---

5. Operating System Components and Functions

Components:

- **Kernel**: Core of the OS that directly interacts with hardware.
- **Shell**: User interface for interacting with the OS.

- **File System**: Manages files and directories.

Functions:

- **Process Management**: Handles creating, scheduling, and terminating processes.
- **Memory Management**: Allocates and deallocates memory for processes.
- **Device Management**: Manages hardware devices and their interactions with the OS.

Example:

- **Kernel**: The engine of a car that drives the vehicle.
- **Shell**: The steering wheel and dashboard that allow you to control the car.
- **File System**: The trunk and compartments that store your belongings.

---

6. Processes

Process Concept:

- **Definition**: A process is a program in execution, including its code, data, and state.

Process States:

- **New**: Process is being created.
- **Ready**: Waiting for CPU time.
- **Running**: Currently executing.
- **Blocked**: Waiting for an event.
- **Terminated**: Finished execution.

Process Control Block (PCB):

- **Definition**: A data structure that stores information about a process (like its state, program counter, and CPU registers).

Process Scheduling Concepts:

- **Goal**: Decides the order in which processes get CPU time.
- **Scheduling Algorithms**:
    - **FCFS**: First process to arrive gets CPU first.
    - **SJF**: Shortest job gets CPU first.
    - **Round Robin**: Each process gets a fixed time slice in rotation.

Threads and Their Management:

- **Definition**: Threads are smaller units of a process that run concurrently.

- **Management**: Involves creating, scheduling, and synchronizing threads.

Example:

- **Processes**: Think of processes as different tasks in a to-do list, with each task having a state (waiting, doing, completed).
- **Threads**: Like sub-tasks within a single task that can be done simultaneously.

---

7. CPU Scheduling

Scheduling Concepts:

- **Definition**: Determines which process or thread gets to use the CPU and for how long.

Performance Criteria:

- **Metrics**: CPU utilization, throughput, turnaround time.

Scheduling Algorithms:

- **FCFS**: Jobs are processed in the order they arrive.
- **SJF**: Shorter jobs are completed before longer ones.
- **Round Robin**: Each job gets a time slice, rotating through them.

Multiprocessor Scheduling:

- **Definition**: Scheduling tasks across multiple CPUs to improve performance.

Example:

- **Scheduling**: Like managing a queue at a checkout counter, where you decide which customer (process) is served next.
- **Multiprocessor Scheduling**: Similar to having multiple cash registers open to handle more customers at once.

---

8. Process Synchronization

Principle of Concurrency:

- **Definition**: Multiple processes or threads run at the same time.

Concurrency Implementation:

- **Fork/Join**: Creates and manages concurrent processes.
- **Parbegin/Parend**: Marks the start and end of parallel tasks.

Inter-Process Communication (IPC):

- **Definition**: Methods for processes to communicate and synchronize.
- **Example**: Pipes, message queues.

Critical Section Problem:

- **Definition**: Ensures only one process accesses a shared resource at a time.
- **Solutions**:
    - **Dekker's Solution**: An algorithm ensuring mutual exclusion.
    - **Peterson's Solution**: Another mutual exclusion algorithm, simpler than Dekker's.
    - **Semaphores**: Tools for controlling access to shared resources.
    - **Synchronization Hardware**: Provides atomic operations (e.g., test-and-set).

Example:

- **Critical Section**: Like ensuring only one person can use a single shared printer at a time.

---

9. Classical Problems in Concurrency

Dining Philosophers Problem:

- **Definition**: Philosophers sitting at a table need chopsticks to eat and think. The challenge is to avoid deadlock and ensure fair access.

Readers/Writers Problem:

- **Definition**: Managing access to a shared resource where readers can access simultaneously, but writers require exclusive access.

Example:

- **Dining Philosophers**: Imagine a group of people who need to share a few utensils and have to avoid conflicts or starvation.
- **Readers/Writers**: Like a library where multiple readers can read at once, but only one person can write or update the catalog at a time.

---

10. Deadlock

System Model:

- **Definition**: Describes how processes and resources interact.

Deadlock Characterization:

- **Conditions**: Mutual Exclusion, Hold and Wait, No Preemption, Circular Wait.

Prevention:

- **Definition**: Techniques to ensure deadlock doesn’t occur by breaking one of the conditions.

Avoidance:

- **Definition**: Dynamically analyzing resource requests to avoid deadlock (e.g., Banker's Algorithm).

Detection:

- **Definition**: Identifying when a deadlock occurs using algorithms.

Recovery:

- **Definition**: Techniques to recover from deadlock, such as process termination or resource preemption.

Combined Approach:

- **Definition**: Using both prevention and avoidance strategies to manage deadlock.

Example:

- **Deadlock**: Imagine two processes each holding one resource and waiting for the other’s resource, causing a standstill.