OS is a system software that controls execution of programs and acts as an interface between applications and computer hardware

1️⃣ Process Management

Definition:

- **Process management** refers to how an **OS handles multiple processes efficiently** by ensuring proper execution, scheduling, and resource allocation.

Key Responsibilities of the OS:
✔ **Interleaving multiple processes** (running them without interference).
✔ **Allocating and protecting resources** (CPU, Memory, I/O).
✔ **Process synchronization** (avoiding conflicts in shared resources).
✔ **Managing process life cycles** (creation, execution, termination).

2️⃣ What is a Process?

💡 **Analogy: A process is like a worker in a factory. A program is the "task list" the worker follows, but a worker also needs tools (resources) and workspace (memory) to complete the task.**

✔ **A process is a program in execution.**
✔ **It has a current state and uses system resources (CPU, memory, files, I/O).**

Example:

- **If you open Google Chrome, each tab runs as a separate process to ensure independent execution.**
- **If one tab crashes, others remain unaffected because they are separate processes.**

Process = Program Code + Execution State + Required Resources

---

3️⃣ Process Elements

A process consists of:

1. **Program Code** (Possibly shared)
2. **Data** (Used by the process)
3. **Process Attributes** (Stored in PCB)

Each process consists of several elements stored in a **Process Control Block (PCB)**:

Process Element	Description
Process ID (PID)	Unique identifier for each process.
State	Running, Ready, Blocked, etc.
Program Counter (PC)	Address of next instruction to execute.
Memory Pointers	Pointers to code, data, and stack.
Registers	CPU registers holding temporary values.
I/O Info	Devices allocated to the process.
Accounting Info	CPU usage, time limits, user ID.

4️⃣ Process Control Block (PCB)

💡 **Analogy: PCB is like an identity card for a process. It stores all the details required for the process to function properly.**

✔ **PCB contains all essential process information.**
✔ **Allows OS to resume a process after interruption.**

🔹 Contents of a PCB

- **Process ID (PID)**
- **Process State (Running, Ready, Blocked, etc.)**
- **CPU Registers & Program Counter**
- **Memory Pointers**
- **I/O Status**
- **Scheduling Information**

Process Control Block

- Created and managed by the OS
- PCB contains sufficient information so that it is **possible to interrupt a running process and resume execution** as if the interrupt had not occurred
- Key tool that enables the OS to support multiple processes; multiprocessing
- → A process consists of Program Code and associated data plus PCB

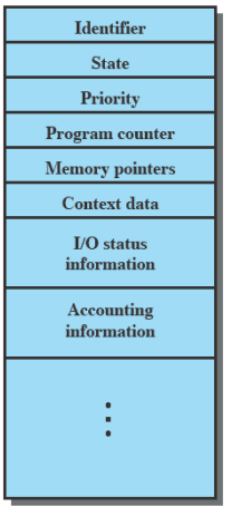


Figure 3.1 Simplified Process Control Block

5️⃣ Trace of a Process

💡 **Analogy: A trace is like a timeline of a process's life. It records every instruction executed by the process.**

✔ **The OS maintains traces to track process execution and scheduling.**
✔ **Two perspectives:**

- **Process View: The sequence of instructions executed.**
- **CPU View: The sequence of processes that the CPU executes.**

6️⃣ Process Execution (Memory & Dispatcher)

✔ A process **needs memory** to execute.
✔ The **dispatcher** is responsible for **switching processes on the CPU.**

Example:

- A game and a web browser are running together.
- The OS **switches between them** quickly, making it appear as if both are running simultaneously.
- The **dispatcher handles these switches efficiently.**

7️⃣ Process Creation

- **Events leading to process creation:**
  - **New job submission** (Batch Processing).
  - **New user logs in** (Interactive system).
  - **OS creates a process** for application needs (e.g., Printing).
  - **Parent process spawns child process.**

✔ **Steps of Process Creation:**
1️⃣ Assign a **unique Process ID (PID)**
2️⃣ Allocate **memory and resources**.
3️⃣ Create and initialize **PCB**.
4️⃣ Add the process to the **ready queue.** Initializes process state

8️⃣ Process Termination

- **Events leading to termination:**
  - **User logout or process completion**
  - **Time limit exceeded**
  - **Memory/resource limits exceeded**
  - **Invalid instructions (e.g., division by zero)**
  - **Parent process terminates child**

✔ **A process terminates when:**
1️⃣ **It completes execution (Normal Termination).**
2️⃣ **The user manually terminates it.**
3️⃣ **The OS forces termination due to an error.**

✔ **Common termination reasons:**

- **Illegal memory access**
- **Division by zero**
- **Infinite loops (CPU time limit exceeded)**

9 Two-State Process Model

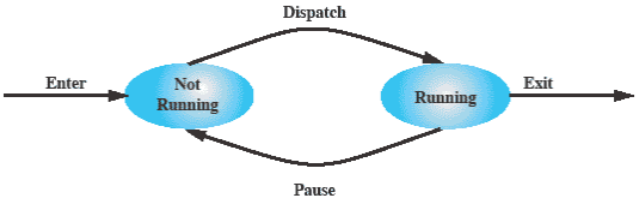
✔ **Two possible states:**

1 **Running: The process is using the CPU.**

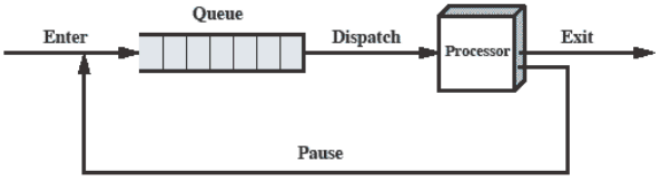
2 **Not Running: The process is waiting.**

✔ **State transition occurs when a process is interrupted or resumes execution.**

- When a new process is created the OS enters the process into the system in Not-running state
- PCB is created and OS knows that the process exists; waiting for opportunity to execute



Two-State Process Model: Queuing Diagram



(b) Queuing diagram

- Processes moved by the dispatcher to the CPU then back to the queue until the task is competed

🔟 Five-State Process Model

NEED:

- 2 state process model- inadequate
  - Would have been effective only if all processes were always ready to execute
- Some processes in not-running state are ready to execute, while others are blocked; waiting for completion of I/O operation
- **Dispatcher should scan the list looking for the process that is not blocked** and has been in the queue the longest

- Split the Not-Running state into 2 states: **Ready** and **Blocked**

Features:

✔ Five states:

State	Description
New	Process created but not admitted to memory.
Ready	Process waiting for CPU time.
Running	Currently executing.
Blocked	Waiting for an event (I/O, memory, etc.).
Exit	Process finished execution.

✔ The dispatcher decides when to switch processes.

Running- process that is currently being executed
Ready- process that is prepared to execute when given the opportunity
Blocked- process that cannot execute until some event occurs
New- process that has just been created but not yet admitted / loaded in main memory; its PCB is created
Exit- process that has been released from the pool of executable processes by the OS (halted or aborted)

Five-State Process Model

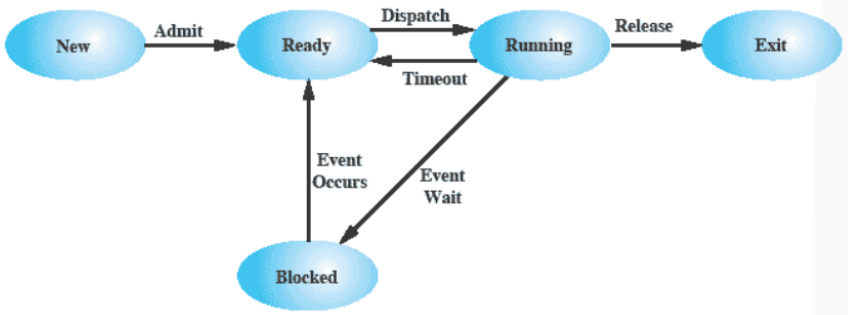


Figure 3.6 Five-State Process Model

State Transitions

- Ready → Running (Scheduler selects process)
- Running → Blocked (Waiting for I/O or event)
- Blocked → Ready (Event completes)
- Running → Exit (Process finishes execution)

1️⃣1️⃣ Suspended Processes

A process is suspended when it is temporarily removed from main memory (RAM) and stored in secondary storage (disk) by the operating system.

✔ The OS moves inactive processes from RAM to disk when there is memory shortage or to improve CPU efficiency.
✔ A suspended process is not immediately available for execution, even if the event it was waiting for is completed.

🔹 Why Do We Need Suspended Processes? (Purpose)

✔ CPU is much faster than I/O, so many processes are often waiting for an event (like reading a file).
✔ Instead of letting them take up valuable RAM, the OS suspends them to secondary storage.
✔ When memory is available again, the OS brings the process back into RAM and resumes execution.

🔹 Characteristics of Suspended Processes

✔ Suspended processes do not execute immediately.
✔ They may or may not be waiting for an event.
✔ They were placed in a suspended state by the OS or parent process.
✔ They must be explicitly resumed by the OS before execution.
✔ The occurrence of a blocking event does not mean automatic execution (it first needs to be loaded back into RAM).

🔹 States of Suspended Processes

Suspended processes can be in one of two possible states:

1️⃣ Blocked-Suspend

✔ The process was waiting for an event (e.g., file read operation).
✔ The OS moved it to disk because memory was needed.

✔ Example: A web browser downloading a large file is blocked waiting for the network. The OS suspends it to disk so that an active application can use memory.

2️⃣ Ready-Suspend

✔ The process is ready to execute but moved to disk due to memory constraints.

✔ Example: A video editing application paused by the user. It is ready to resume but remains on disk because the OS prioritizes another running process.

🔹 Suspended Process Transitions (State Diagram)

A process can transition **between different states** depending on resource availability.

vbnet

🔹 State Transitions of Suspended Processes

Transition	Reason
Blocked → Blocked-Suspend	No available RAM, process is swapped to disk.
Blocked-Suspend → Ready-Suspend	I/O event completes, but no free memory to bring it back.
Ready-Suspend → Ready	Memory becomes available, process is brought back to RAM.
Running → Ready-Suspend	Process is preempted and moved to disk to free memory.
New → Ready-Suspend	A newly created process is **immediately suspended** due to memory shortage.

📌1️⃣2️⃣ One Suspend State Model

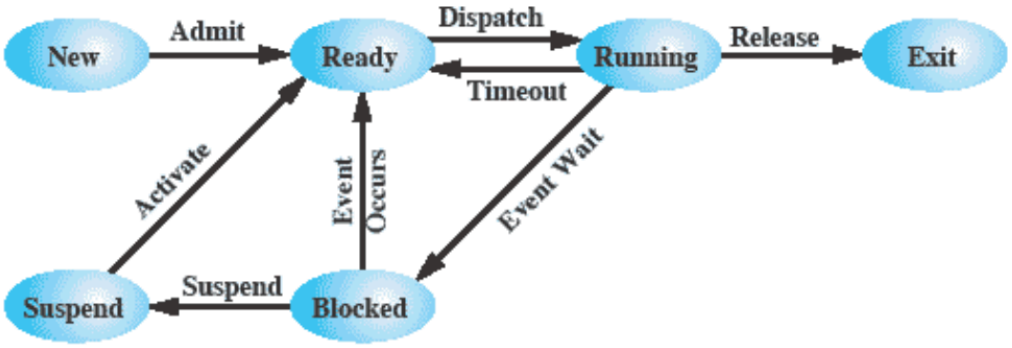
✔ In this model, there is only one suspend state:

- **Blocked-Suspend: A process is waiting for an event AND swapped out to disk.**

💡 Analogy: Think of a hospital waiting room with only one waiting section. Patients waiting for an appointment must wait in the same area, whether their appointment is soon or hours away.

✔ This model is simple, but not efficient for handling a large number of processes.

Process Model: One Suspend State



(a) With One Suspend State

📌1️⃣3️⃣ Two Suspend State Model

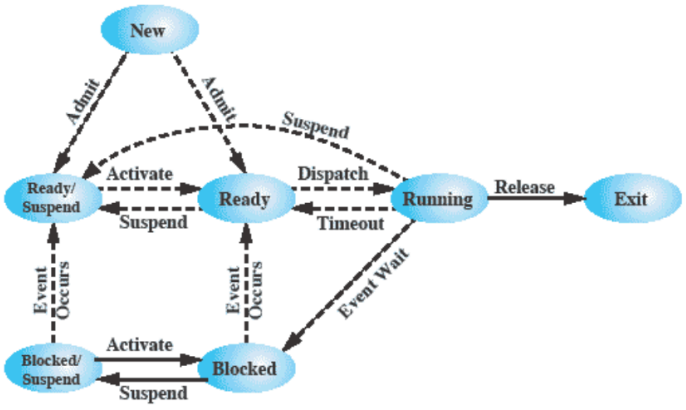
✔ To improve performance, OS uses two separate suspend states:
1️⃣ Blocked-Suspend (Waiting for an event and swapped to disk).
2️⃣ Ready-Suspend (Ready to run but swapped to disk).

💡 Analogy: Instead of one hospital waiting room, we have two sections:

- One for patients waiting for a doctor (Blocked-Suspend).
- One for patients ready for a check-up but waiting for an available room (Ready-Suspend).

✔ This reduces CPU idle time and improves memory utilization.

Process Model: Two Suspend State



(b) With Two Suspend States

📌 Reasons for Process Suspension

The OS suspends processes under certain conditions to manage memory and CPU usage efficiently.

Reason	Description	Example
Memory Shortage	**If RAM is full, low-priority processes are moved to disk (swapped out) to free memory.**	**If you run multiple heavy applications, like a game and video editing software, one may be suspended to free RAM.**
Parent Process Suspension	**If a parent process is suspended, its child processes are also suspended to maintain execution order.**	**If you pause Google Chrome, all its tabs (child processes) may be suspended.**
User Request	**A user can manually suspend a process to pause execution.**	**A user pauses a file download while waiting for better Wi-Fi.**
Resource Unavailability	**A process needs a resource (e.g., printer, network), but it is unavailable. Instead of blocking RAM, the process is suspended.**	**A document waiting for a busy printer is suspended until the printer is free.**
Timing Reasons	**Some processes are scheduled to run later, so they are suspended until their execution time arrives.**	**Windows Update runs at night, so it is suspended during the day.**
Deadlock Handling	**If a deadlock occurs, the OS suspends one or more processes to break the deadlock and free up resources.**	**If two processes are waiting for each other's resources, one is suspended.**

📌 Characteristics of Suspended Processes

Suspended processes behave differently from normal ready or blocked processes.

Characteristic	Explanation
Not immediately available for execution	A suspended process cannot execute immediately even if its event is completed. It must be loaded back into RAM first.
May or may not be waiting for an event	Some suspended processes are waiting for I/O (Blocked-Suspend), while others are ready to execute but waiting for memory (Ready-Suspend).
Process does not resume automatically	Just because an event is completed does not mean the process will run immediately. It must be explicitly resumed by the OS.
Suspended by OS, Parent, or User	Processes can be suspended by the operating system, parent process, or user action.
Process remains in storage (disk)	Unlike blocked processes, which are in RAM, suspended processes are stored in secondary memory (disk).
Swapping saves memory space	Suspension helps free up RAM and allocate it to active processes.
Process retains its state information	The OS stores all PCB details so that when the process is resumed, it can continue from where it left off.

📌1️⃣4️⃣ OS Control Structure

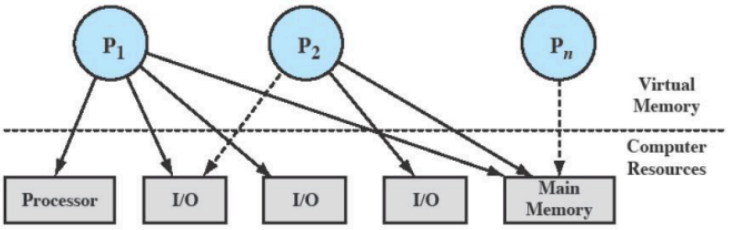
✔ The OS maintains several tables to track process and resource usage.
✔ The OS needs to maintain control over all processes and system resources.
✔ It stores and manages process-related information in different tables (data structures).

The OS Control Structure is a set of data structures and tables used by the Operating System (OS) to track and manage processes, memory, files, and I/O devices.

🔹 Why Do We Need an OS Control Structure?

✔ The OS needs a way to monitor all running processes and system resources.
✔ Without proper tracking, resource conflicts would occur, and the system could crash.
✔ The OS uses tables to store information about:
1️⃣ Processes (which programs are running).
2️⃣ Memory (which parts are free/allocated).
3️⃣ I/O Devices (which devices are in use).
4️⃣ Files (which files are open and their permissions).

- During the execution, every process needs access to certain resources e.g. processor, memory, I/O devices etc.
- OS manages use of system resources by processes



🔹 Types of OS Control Structures

The OS maintains four major tables to manage resources and processes efficiently.

1️⃣ Memory Table (Manages RAM & Virtual Memory)

✔ **The Memory Table keeps track of how memory is allocated and used.**
✔ **Used for address translation, process isolation, and swapping.**

Field	Description
Memory Block Address	**Tracks allocated and free memory locations.**
Process ID	**Identifies which process owns a memory block.**
Protection Information	**Specifies which processes can access a memory block.**
Paging/Segmentation Info	**Stores virtual memory details (page tables, segment tables).**

💡 **Example:**

- **When opening a new application, the OS checks the Memory Table to allocate memory.**
- **If RAM is full, it swaps inactive processes to disk to free space.**

2️⃣ Process Table (Tracks Running & Suspended Processes)

✔ **The Process Table stores important details about each process, such as:**

- **State (Ready, Running, Blocked, Suspended).**
- **Process ID (PID) and Parent Process ID (PPID).**
- **CPU Registers & Memory Allocation.**

Field	Description
Process ID (PID)	**Unique identifier for each process.**
Process State	**Ready, Running, Blocked, Suspended, etc.**
Program Counter (PC)	**Stores the address of the next instruction to execute.**
CPU Registers	**Stores temporary values for execution.**
Memory Pointers	**Tracks allocated memory regions.**

💡 **Example:**

- **When switching between apps (Alt + Tab in Windows), the OS uses the Process Table to pause one process and resume another.**

3️⃣ I/O Table (Manages Input & Output Devices)

✔ The I/O Table tracks which processes are using which devices.
✔ Prevents conflicts when multiple processes access the same device.

Field	Description
Device ID	Unique identifier for each I/O device.
Process ID (PID)	Tracks which process is using the device.
Device Status	Indicates whether the device is Busy, Idle, or Error.
Buffer Location	Tracks memory buffers used for I/O transfers.

💡 Example:

- If two processes try to print at the same time, the OS queues the requests using the I/O Table to avoid conflict.

4️⃣ File Table (Manages Open Files & Access Permissions)

✔ The File Table keeps track of:

- Which files are open.
- Where they are stored.
- Who can access them (permissions).

Field	Description
File Name	Identifies the file being accessed.
File Location	Stores the file's path on disk.
Access Permissions	Read, Write, Execute permissions for each user/process.
Process ID (PID)	Tracks which process is using the file.

💡 Example:

- When you open a Word document, the OS updates the File Table so that other programs can't modify it while it's in use.

📌 How the OS Control Structure Works Together

✔ The OS continuously updates these tables to manage multitasking efficiently.

1️⃣ A new process starts → The Process Table is updated.
2️⃣ The OS allocates memory → The Memory Table records the details.
3️⃣ If the process needs a file → The File Table is checked for access rights.
4️⃣ If the process requires a device (e.g., printer) → The I/O Table schedules access.
5️⃣ If RAM is full, the OS suspends a process and updates the tables.

📌 Why is the OS Control Structure Important?

✔ Ensures multitasking works smoothly.
✔ Prevents resource conflicts (e.g., two processes writing to the same file).
✔ Improves system performance by optimizing memory and CPU scheduling.

💡 Example: When using multiple apps on your phone, the OS:

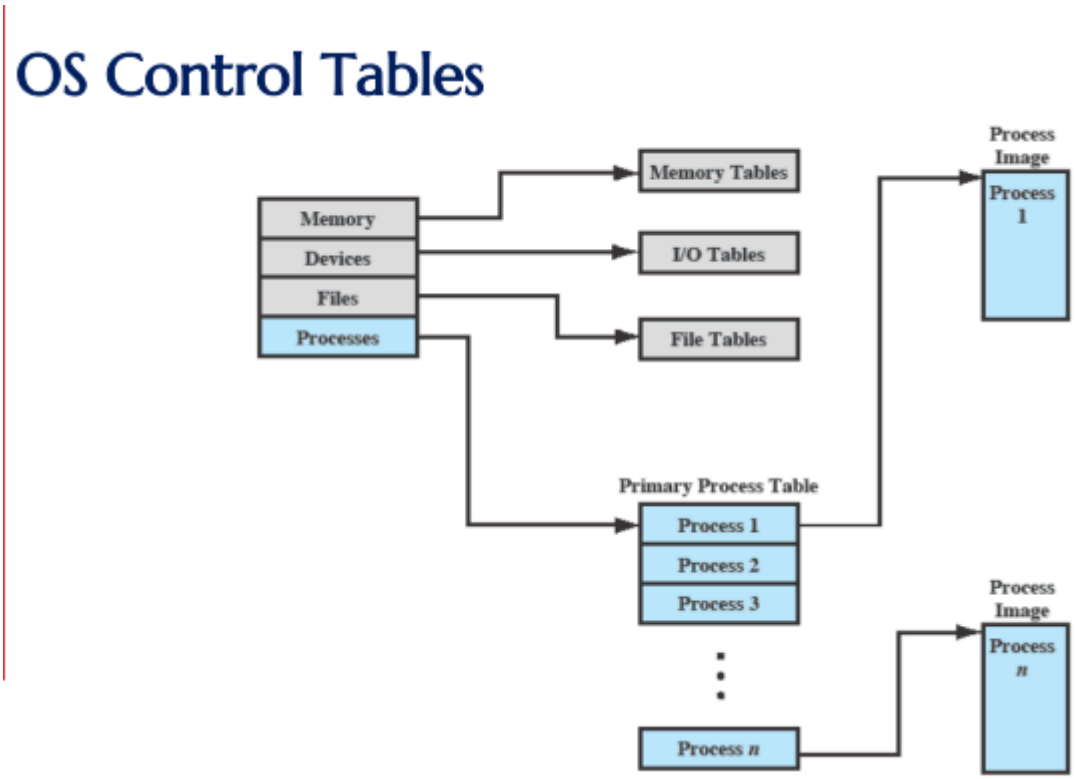
- Allocates memory for each app (Memory Table).
- Tracks active apps & background processes (Process Table).
- Manages file access (e.g., images, documents) (File Table).
- Controls network usage, camera, and storage access (I/O Table).

🔥 Final Summary

Table Name	Purpose	Real-World Example
Memory Table	Tracks memory allocation, page tables, and segment tables.	Ensures Chrome & Photoshop don't use the same RAM space.
Process Table	Tracks running, ready, and suspended processes.	Allows fast switching between apps in a mobile OS.
I/O Table	Manages device access and prevents conflicts.	Prevents two apps from using the microphone at the same time.
File Table	Tracks open files, locations, and permissions.	Prevents two programs from modifying a document at the same time.

🚀 Conclusion

✔ The OS Control Structure is the foundation of process management in any modern OS.
✔ It enables multitasking, memory allocation, and efficient device usage.
✔ Without it, the system would be unable to track active processes, memory usage, or file operations.



📌1️⃣5️⃣ Process Attributes

A process has three key attributes that help the OS manage and control execution.

🔹 1. Process Identification

✔ Every process is assigned a unique identifier (PID).
✔ The OS uses this to track and manage processes.

✔ Important Identifiers:
1️⃣ Process ID (PID) → Unique number assigned to each process.
2️⃣ Parent Process ID (PPID) → ID of the process that created it.
3️⃣ User ID (UID) → Identifies the user who owns the process.

💡 Example: In Linux, you can check running processes using:

bash

```
ps -aux
```

which shows PIDs of all active processes.

🔹 2. Processor State Information

✔ Stores the current status of a process, including:
1️⃣ CPU Registers (General & Special Registers)
2️⃣ Program Counter (PC) (Address of next instruction to execute)
3️⃣ Stack Pointer (SP) (Points to the top of the call stack)
4️⃣ Process Status Word (PSW) (Contains system flags, execution mode)

💡 Example: If a process is interrupted, the OS saves these values and restores them when the process resumes.

🔹 3. Process Control Information

✔ Includes information related to scheduling, memory allocation, and inter-process communication.

✔ Key Elements:

- **Scheduling Info (Priority, Waiting Time, Execution Time)**
- **Memory Management Info (Page Tables, Segment Tables)**
- **I/O Info (Devices assigned to process)**
- **Resource Ownership (Open files, shared memory)**

💡 **Example: If two processes communicate using shared memory, the OS tracks this in the PCB.**

📌1️⃣6️⃣ Role of Process Control Block (PCB)

The Process Control Block (PCB) is the most important OS data structure because it stores all the necessary information about a process.

✔ **The PCB enables process switching and multitasking.**
✔ **Each running process has a PCB, which the OS manages.**

🔹 Contents of a PCB

Category	Description
Process ID (PID)	**Unique identifier for the process**
Process State	**Running, Ready, Blocked, Suspended**
Program Counter (PC)	**Stores the next instruction address**
CPU Registers	**Stores temporary values for execution**
Memory Pointers	**Stores addresses of program code, stack, and heap**
I/O Status	**Tracks assigned I/O devices, files in use**
Scheduling Info	**Priority, waiting time, execution time**

✔ **The OS must protect the PCB because corrupting it can crash the system!**

💡 **Analogy: PCB is like a "Save Game" file in a video game, allowing the OS to resume a process exactly where it stopped.**

📌17 Modes of Execution

✔ **The CPU operates in two modes to protect the OS from user programs.**

🔹 1. User Mode

✔ **The process runs with limited privileges.**
✔ **Cannot access system hardware or modify OS data.**

💡 **Example: When you run a text editor, it runs in user mode and cannot directly access memory or CPU.**

🔹 2. Kernel Mode

✔ **The OS runs with full privileges and can:**

- **Directly access memory and CPU registers.**
- **Execute privileged instructions.**
- **Manage process scheduling and interrupts.**

💡 **Example: When you open a file, the OS switches to kernel mode to access the file system.**

📌18 Typical Functions of OS Kernel

✔ **The OS Kernel is the core part of the OS that manages hardware and software.**

🔹 Key Functions:

1️⃣ **Process Management → Scheduling, process creation, switching.**
2️⃣ **Memory Management → Allocation, paging, virtual memory.**
3️⃣ **I/O Management → Buffering, device allocation, drivers.**
4️⃣ **Interrupt Handling → Managing external events (keyboard input, system calls).**

💡 **Example: The kernel ensures processes don't interfere with each other by managing CPU time allocation.**

📌1️⃣9️⃣ Process Creation (Steps & Explanation)

✔ The OS follows these steps when creating a process:

🔹 Steps in Process Creation

1️⃣ Assign a unique Process ID (PID)
2️⃣ Allocate memory for the process
3️⃣ Initialize the Process Control Block (PCB)
4️⃣ Set up appropriate linkages (parent-child relationships)
5️⃣ Move the process to the Ready Queue

💡 Example: When you open an application, the OS creates a new process and moves it to the Ready state.

📌2️⃣0️⃣ Process Switching

✔ Process Switching happens when the CPU stops executing one process and starts another.

🔹 When Does Process Switching Occur?

1️⃣ Time Slice Expiry → Process used all its allocated CPU time.
2️⃣ Higher Priority Process Arrives → A more important process must execute.
3️⃣ I/O Request → Process moves to Blocked state.
4️⃣ Process Completes → Moves to Exit state.

💡 Example: The OS switches from a video player process to a web browser process when you switch applications.

📌2️⃣1️⃣ Mode Switching vs Process Switching

🔹 What Happens in Mode Switching?

1️⃣ The CPU **saves the current mode** (User Mode).
2️⃣ The CPU **switches to Kernel Mode**.
3️⃣ The OS **executes the system call, interrupt, or exception handler**.
4️⃣ The CPU **switches back to User Mode** and resumes the process.

✔ **No process change occurs**—only privilege level changes.

💡 **Example:** When a **game saves progress**, it temporarily switches to **Kernel Mode** to write data to a file, then goes back to **User Mode** to continue playing.

🔹 What is Process Switching?

✔ **Process switching occurs when the CPU stops executing one process and starts another.**
✔ This requires **saving the old process's state (context) and loading the new process's state.**
✔ Process switching occurs due to **scheduling decisions, I/O requests, or process termination.**

💡 **Analogy:**
Imagine you are **watching a movie** on your laptop, but your friend wants to **check an email**.

- You **pause the movie** (Save the process state).
- Your friend **opens their email app** (Load another process).
- Later, you **resume your movie from where you left off**.

📌 Mode Switching vs Process Switching (Comparison Table)

Feature	Mode Switching	Process Switching
Definition	CPU switches from **User Mode to Kernel Mode (or vice versa).**	CPU stops one process and switches to another.
Process Change?	❌ No	✅ Yes
Overhead (Time Cost)?	**Low** (Only changes execution mode)	**High** (Context switch, memory update required)
Triggered By?	**System Calls, Interrupts, Exceptions**	**CPU Scheduling, I/O Requests, Process Termination**
State Change?	❌ No (Process continues running)	✅ Yes (Process moves to Ready, Blocked, or Exit)
Registers/Context Save?	❌ No (Registers are not saved/restored)	✅ Yes (Process registers must be saved/restored)
Example	**Opening a file (System Call), Handling a keyboard event (Interrupt), Handling a division by zero (Exception)**	**Switching from YouTube to Chrome, Switching from a paused game to a video call**