An Operating System (OS) is responsible for managing hardware, providing an environment for applications, and ensuring efficient utilization of system resources. Two of the most fundamental concepts in OS are *processes* and *threads*. A **process** is an independent program in execution. It has its own memory space (code, data, stack, heap) and is represented in the system by a Process Control Block (PCB). Every process operates in isolation, which ensures stability but also increases overhead because switching between processes is expensive—each context switch requires saving and loading a separate memory space.

A **thread**, on the other hand, is a smaller execution unit inside a process. Multiple threads within the same process share memory such as global variables and heap, but each thread has its own stack and program counter. This shared memory makes threads lightweight and suitable for parallelism within the same application. For example, a browser may have separate threads for rendering, downloading, and handling user interactions.

OS uses **CPU scheduling** to determine which process or thread gets processor time. Scheduling becomes necessary because the CPU can execute only one instruction flow at a time (ignoring multi-core scenarios). The goal is to maximize CPU utilization, throughput, and responsiveness.

Common scheduling algorithms include:

1. **First-Come, First-Served (FCFS):**
   The simplest algorithm where the first process to arrive is the first to execute. It suffers from the "convoy effect," where a long job delays all others.

2. **Shortest Job First (SJF):**
   Prioritizes processes with the smallest execution time. It reduces waiting time but requires predicting process burst time.

3. **Round Robin (RR):**
   Each process gets a fixed time slice (quantum). It is ideal for time-sharing systems because no process can monopolize the CPU.

4. **Priority Scheduling:**
   Each process is assigned a priority. The CPU selects the highest-priority process. It may lead to starvation if low-priority processes never execute.

5. **Multilevel Queue Scheduling:**
   Processes are divided into multiple queues (e.g., system, interactive, batch), each with different scheduling rules.

Understanding processes, threads, and scheduling is crucial because they determine how efficiently an application and system perform. Multi-threading improves concurrency, while effective scheduling ensures fairness and maximizes CPU usage.