**Operating System**

An **operating system** is an interface between the user and hardware, responsible for process execution, resource allocation, CPU management, file management, and providing a convenient and efficient environment for executing programs.

Functions of Operating System: Memory Management, Processor Management, User Interfaces, File Management, Device Management, Security.

Some names of operating systems: MS Windows, Chrome OS, macOS, Android, and Ubuntu.

**Types of Operating Systems**

**1. Batch Operating System**

Jobs with similar requirements are grouped into batches and processed sequentially without user interaction.

2. Multiprogramming OS

**Multiprogramming OS** runs multiple programs simultaneously by managing system resources efficiently. It switches between programs so the CPU is always busy, improving performance and reducing idle time.

**3. Multiprocessing Operating System**

A multiprocessing operating system uses multiple CPUs to perform tasks simultaneously. Each CPU works on different tasks at the same time, which enhances performance and reliability.

**4. Multi-Tasking Operating**

A **Multi-Tasking Operating System** is a logical extension of a multiprogramming system that allows multiple tasks or processes to run concurrently on a single CPU. It achieves this by rapidly switching between tasks, giving the illusion that all are running simultaneously.

**5. Multithreading OS**

Multithreading is a feature that allows a single process to have multiple threads of execution running concurrently. Each thread can perform a different task, but they share the same resources such as memory, which makes communication between threads easier.

**6. Real-Time Operating System (RTOS)**

A Real-Time Operating System ensures tasks are completed within a fixed time frame, crucial for time-sensitive applications like medical devices and automotive systems.

**User and Kernal Space in OS**

**User Space** is the memory area where user applications run, separate from the core of the system.

**Kernel Space** is where the operating system's core functions operate, managing hardware and system resources like CPU, memory, and devices

**Functions of OS**

* **Process Management**: Manages the creation, scheduling, and termination of processes, ensuring efficient CPU utilization.
* **Memory Management**: Allocates and manages system memory (RAM) for processes, and handles memory allocation and deallocation.
* **File System Management**: Organizes, stores, retrieves, and manages data on storage devices, like hard drives.
* **Device Management**: Manages communication between hardware devices (I/O devices) and processes, handling input and output.
* **Security and Access Control**: Protects data and resources by controlling user access and permissions.
* **User Interface**: Provides an interface (CLI or GUI) for user interaction with the system.

**Types of Kernels:**

* **Monolithic Kernel**: A single large kernel that includes all necessary services, such as device drivers and file system management, running in kernel space.
* **Microkernel**: This kernel design runs only the most essential services in kernel space, like inter-process communication and basic scheduling, while other services run in user space..
* **Hybrid Kernel**: A combination of monolithic and microkernel designs. It runs essential services in kernel space while allowing some components to operate in user space.
* **Nanokernel**: A **nanokernel** is a minimal kernel that provides only basic hardware-level functions, like CPU and memory management, while delegating other services to user-level processes for simplicity and flexibility.

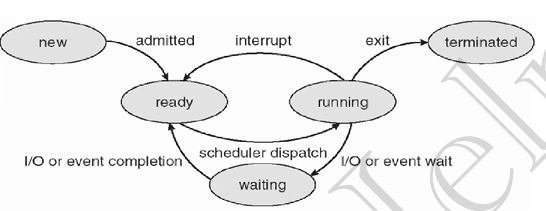
A program is a set of instructions written in a programming language that a computer can execute to perform a specific task.

Process: A program under execution, indicated by the program counter (PC), represented by a Process Control Block (PCB).

**Process States**

As a process executes, it changes states. Each process may be in one of the following states:

* **New**: The OS is about to convert the program into a process; the process is being created.
* **Running**: Instructions are being executed; the CPU is allocated.
* **Waiting**: The process is waiting for I/O operations to complete.
* **Ready**: The process is in memory, waiting to be assigned to a processor.
* **Terminated**: The process has finished execution; the PCB entry is removed from the process table.



**Process Control Block (PCB)**  
The Process Control Block (PCB) is a data structure used by the operating system to store information about a process.

**Key Attributes** **are** **Process ID (PID), Process State, Program Counter, CPU Registers, Memory Management Info, I/O Status, Scheduling Info, Accounting Info**

**Functions**:

* Manages process execution and state.
* Facilitates context switching by saving and loading process states.
* Helps in resource management and allocation.

**Thread**

A **thread** is the smallest unit of execution within a process. It is a lightweight component that shares the same memory and resources as other threads in the same process.

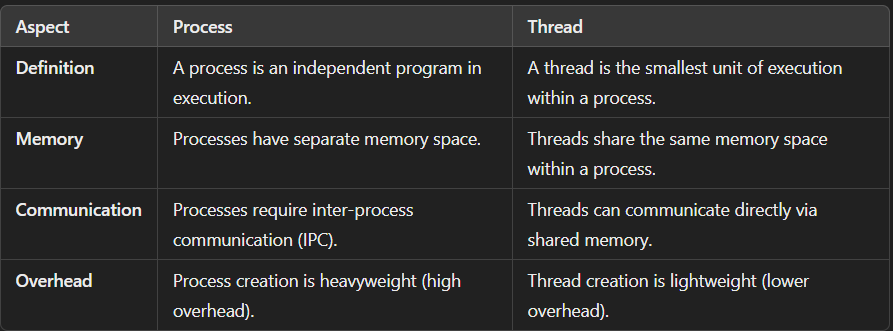
**Characteristics**:

* + Each thread has its own program counter, register set, and stack.
  + Threads share resources (code, data, files, signals) with other threads in the same process.

**Note**: A new thread (child process) can be created using the fork() system call, with n calls generating 2n−12^n - 12n−1 child processes.

**Types of Threads**:

* **User Threads**: Implemented by users.
* **Kernel Threads**: Implemented by the operating system.



**CPU Scheduling**

**Non-Preemptive scheduling**

Once CPU has been allocated to a process, the process keeps the CPU until it releases CPU either by terminating or by switching to wait-state.

**Preemptive scheduling**

CPU is taken away from a process after time quantum expires along with terminating or switching to wait-state.

**Process Scheduling**:

1. **Arrival Time**: When the process enters the ready queue.
2. **Completion Time**: When the process finishes execution.
3. **Burst Time**: Time required for CPU execution.
4. **Turn Around Time**: Time from arrival to completion (Completion Time - Arrival Time).
5. **Waiting Time (WT)**: Time difference between turnaround time and burst time (Waiting Time = Turnaround Time - Burst Time).

Scheduling algorithms:

1. **First Come First Serve (FCFS)**: Schedules processes based on arrival times.
2. **Shortest Job First (SJF)**: Schedules processes with the shortest burst time first.
3. **Shortest Remaining Time First (SRTF)**: Preemptive SJF that schedules based on the shortest remaining time.
4. **Round Robin (RR)**: Each process receives a fixed time slice in a cyclic manner.
5. **Priority Based Scheduling (Non-Preemptive)**: Schedules processes by priority, with ties resolved by arrival time.
6. **Highest Response Ratio Next (HRRN)**: Schedules based on the highest response ratio, preventing starvation (Response Ratio = (Waiting Time + Burst Time) / Burst Time).
7. **Multilevel Queue Scheduling (MLQ)**: Processes are placed in different priority queues; higher priority queues are scheduled first.
8. **Multilevel Feedback Queue (MLFQ)**: Allows processes to move between queues based on CPU usage characteristics



**DeadLock**  
A situation where a set of processes are blocked because each process is holding a resource and waiting for another resource held by some other process.

**Necessary Conditions for Deadlock**

For a deadlock to occur, the following four conditions must be present simultaneously in a system:

* **Mutual Exclusion:** At least one resource must be held in a non-sharable mode.
* **Hold and Wait:** Processes holding resources can request additional resources without releasing their current resources.
* **No Preemption:** Resources cannot be forcibly taken from a process; they must be voluntarily released by the holding process.
* **Circular Wait:** There exists a set of processes {P1, P2, ..., Pn} such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3, and so on, with Pn waiting for a resource held by P1, forming a circular chain.

**Handling Deadlocks**

To manage deadlocks, we can use the following approaches:

* **Deadlock Prevention:** Modify one of the four necessary conditions to ensure that deadlocks cannot occur: **Eliminate Mutual Exclusion, Eliminate Hold and Wait, Allow Preemption, Eliminate Circular Wait**
* **Deadlock Avoidance:** Dynamically evaluate resource requests to ensure that the system remains in a safe state. Use algorithms like the **Banker’s Algorithm**.
* **Deadlock Detection and Recovery:**
  1. **Detection:** Periodically check for deadlocks by constructing a wait-for graph and looking for cycles.
  2. **Recovery:** When a deadlock is detected, take actions such as:**Terminating one or more processes or Preempting resources**
* **Ignore Deadlocks (Ostrich Algorithm):** simply ignoring them.

**Critical Section Problem**

The **Critical Section Problem** arises when multiple processes need access to shared resources. To prevent conflicts, only one process can be in the critical section (where the shared resource is accessed) at a time.

**Synchronization Tools:**

1. **Semaphore**: A protected variable used to lock resources; indicates resource status. Binary Semaphore, Counting Semaphore. **wait(), signal().**
2. **Mutex**: Provides mutual exclusion; only one thread can access a resource at a time, ensuring synchronization between producers and consumers.

**Banker’s Algorithm**  
The Banker’s Algorithm is a deadlock avoidance algorithm used in operating systems to allocate resources safely by determining whether granting a resource request will keep the system in a safe state.

Main Features of Banker’s Algorithmare **deadlock avoidance, safe state verification, resource allocation tracking**

**Contiguous Memory Management**

In this method, a process is allocated a continuous block of memory. It includes two approaches:

**Fixed Partitioning**

In the fixed partition scheme, memory is divided into fixed number of partitions.

**Limitations**:

* **Internal fragmentation**: Unused space within partitions if the process is smaller than the partition.
* **External fragmentation**: Free space between partitions can’t be utilized effectively.
* **Low multiprogramming**: Fixed partitioning restricts the number of processes, limiting system efficiency.

**Dynamic** **Partitioning:**

Partition sizes are allocated dynamically at runtime based on process size.

**Advantages**:

* **No internal fragmentation**: Memory is allocated exactly as per process size.
* **No limit on process size**: Larger processes can be accommodated without fixed partition restrictions.
* **Higher multiprogramming**: More processes can be loaded as partition sizes adjust dynamically.

**Limitation**: External fragmentation can still occur, requiring compaction or other solutions.

**Placement Algorithm:**

* **First Fit:**Allocates the first available block that is large enough for the process.
* **Best Fit:** Allocates the smallest available block that is large enough, minimizing wasted space.
* **Worst Fit:**Allocates the largest available block, potentially leaving the most room for future processes**.**

**Fragmentation:**  
Fragmentation refers to the inefficient use of memory space in a computer system, leading to wasted memory. It occurs in two main forms:

* **Internal Fragmentation:**  
  This happens when allocated memory blocks contain unused space.
* **External Fragmentation:**  
  This occurs when free memory is divided into small, non-contiguous blocks over time, making it difficult to allocate larger contiguous memory requests.

**Logical Address:**

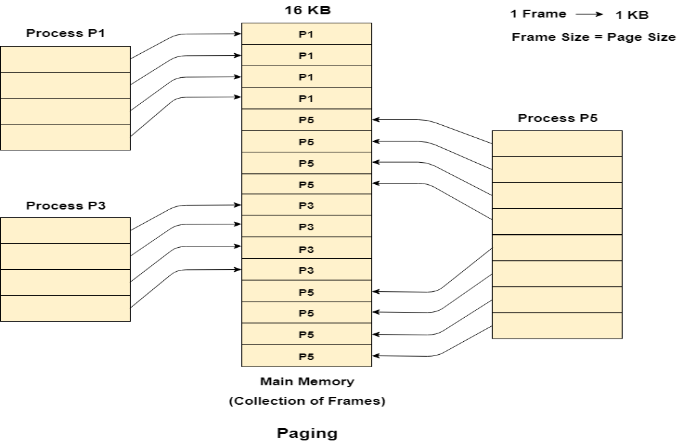
Also called a **virtual address**, it is generated by the CPU during program execution.

**Physical Address:**

This is the actual location in **main memory (RAM)** where data is stored.

**Paging**

Paging is a memory management technique used by operating systems to avoid the issue of **external fragmentation** that occurs in dynamic partitioning. It allows the physical address space of a process to be **non-contiguous**, which makes memory allocation more flexible.



**1. Memory Division**:

* Physical memory is divided into fixed-size blocks called **frames**.
* Logical memory (process address space) is divided into **pages** of the same size.
* A **page table** maps pages to frames, allowing processes to be split and stored across available frames.

**2. Performance**:

* Paging can be slower due to the extra step of looking up the page table. To speed this up, systems use a **Translation Lookaside Buffer (TLB)**, which caches recent page-to-frame mappings for faster access.

**Advantages:**

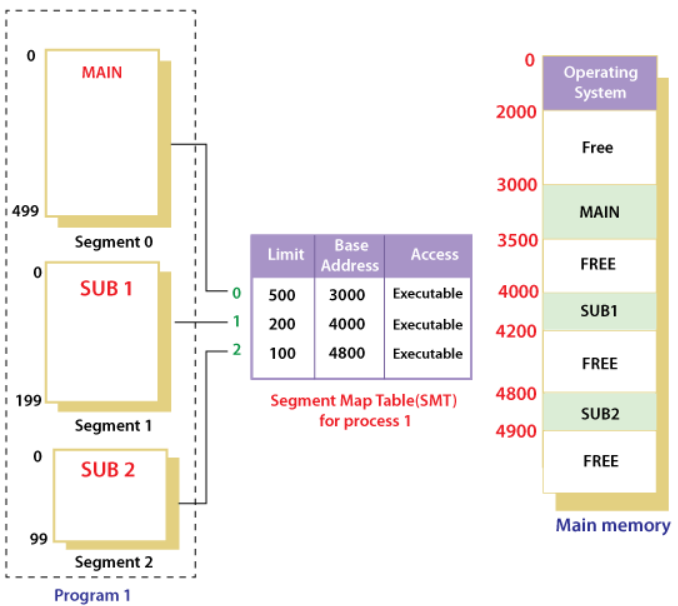
* **No External Fragmentation**: Since pages and frames are of fixed size, there are no gaps or holes in memory, eliminating external fragmentation.
* **Efficient Memory Utilization**: Small free blocks in memory can be used more effectively, as processes are divided into smaller chunks (pages).

**Disadvantages of Paging:**

* **Internal Fragmentation**: Wasted space in the last page.
* **Page Table Overhead**: High memory consumption for large processes.
* **Increased Access Time**: Extra step to look up the page table.

**Segmentation**

Segmentation is a memory management technique where a process is divided into different **segments** based on the user's logical view of the process. Each segment represents a logical unit of the process, such as functions, data structures, or arrays.



**Key Concepts:**

* **Logical Addressing**: A logical address consists of a **segment number** and an **offset** (<segment-number, offset>). This defines the segment and the specific location within it.
* **Segment Table**: The operating system maintains a **Segment Table** that maps these logical addresses to **physical memory**. The **Base Address** indicates where the segment starts in memory, and the **Segment Limit** specifies the size of the segment.
* **User-Oriented Memory**: Unlike **paging**, which focuses on optimizing memory for the operating system by dividing memory into pages, segmentation is more aligned with the **user’s view of memory**.

**Advantages of Segmentation:**

* **No internal fragmentation**: Segments fit the exact memory needs, avoiding wasted space.
* **Efficient memory use**: Segments are allocated contiguously, improving performance.
* **Logical grouping**: Aligns with program structure, keeping related data together.

**Disadvantages of Segmentation:**

* **External fragmentation**: Free memory gets broken into unusable pieces over time.
* **Swapping issues**: Variable segment sizes make finding contiguous memory harder.
* **Complexity**: Managing variable-sized segments adds overhead and complexity.

**Virtual Memory**

* Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.
* By this, user load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.
* Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.
* By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

**Advantages:**

* **No physical memory constraint:** Programs can be larger than the available physical memory.
* **Better multi-programming:** Since each program takes up less physical memory, more programs can run simultaneously, improving CPU utilization and throughput.
* **Large applications:** Users can run large applications with less physical memory, increasing system flexibility.

**Disadvantages:**

* Slower Performance due to time-consuming swapping.
* Thrashing occurs when excessive swapping slows the system.

**Demand Paging:** A process where only the required pages of a program are loaded into memory on demand. This reduces memory usage by avoiding loading unnecessary pages. Pages that aren't used frequently are kept in secondary storage until needed.

**Page Fault:** When a program tries to access a page that isn’t in memory, a page fault occurs. The operating system then loads the required page from the secondary memory into the main memory and resumes the process.

**Page Replacement Algorithms:**

* **First In First Out (FIFO)**: This simplest algorithm uses a queue to track pages, replacing the oldest page first.
* **Optimal Page Replacement**: This algorithm replaces pages that will not be used for the longest time in the future. It’s impractical as future requests are unknown.
* **Least Recently Used (LRU)**: This algorithm replaces the least recently used page.

**Belady's Anomaly:**

Increasing page frames can lead to more page faults in certain algorithms (like FIFO), which is unexpected since we assume more memory should improve performance.

**Thrashing**

Thrashing occurs when a process doesn't have enough frames to support its active pages, leading to constant page faults and replacements. The system spends more time handling page faults than executing processes, severely degrading performance.

**Handling Thrashing**:

* **Reduce multiprogramming**: Run fewer processes to allocate more memory per process.
* **Use efficient page replacement**: Implement algorithms like **LRU** or the **Working Set Model** to manage memory better.
* **Local page allocation**: Assign a fixed number of memory frames per process to avoid overconsumption by any one process.
* **Increase physical memory (RAM)**: More RAM reduces the need for excessive paging, preventing thrashing.

**Terms:**

**Data Binding:** The process of connecting application data to UI elements, allowing dynamic updates and interactions.

**Associative Mapping:** A cache mapping technique where any block can be stored in any cache line, allowing flexible storage but requiring complex lookup.

**Direct Mapping:** A cache mapping method where each block maps to a specific cache line, simplifying implementation but can lead to more cache misses.

**Context Switching:** The process of saving the state of a currently running process and loading the state of another, allowing multitasking in operating systems.

**Bootloader:** A small program that loads the operating system into memory during startup. It initializes the hardware and hands control to the OS.

**Zombie process**: A process that has finished the-execution but still has an entry in process table.

**Orphan process:** A process whose parent process no more exist.

**IPC (Inter-Process Communication):** Mechanisms that enable processes to communicate and synchronize their actions, facilitating data exchange between them.

**Pipe:** A unidirectional communication channel that connects the output of one process to the input of another, allowing data flow.

**Socket:** An endpoint for sending and receiving data over a network, providing a way for processes to communicate across different machines.

**Spooling:** Storing data temporarily before sending it to a device (like a printer) for processing. It helps manage tasks efficiently when devices can't handle them instantly.

**Starvation:**Starvation in an OS happens when a process can't get the needed resources because others are always prioritized. It ends up waiting indefinitely without getting a chance to run.

**Aging:** A technique to prevent starvation by gradually increasing the priority of waiting processes, ensuring they eventually receive the resources they need.

RAID: RAID (Redundant Array of Independent Disks) is a technology that combines multiple physical drives into one logical unit to enhance performance, ensure data redundancy, and provide protection against drive failures.There are 7 RAID levels.