coding75.com

Operating System Notes

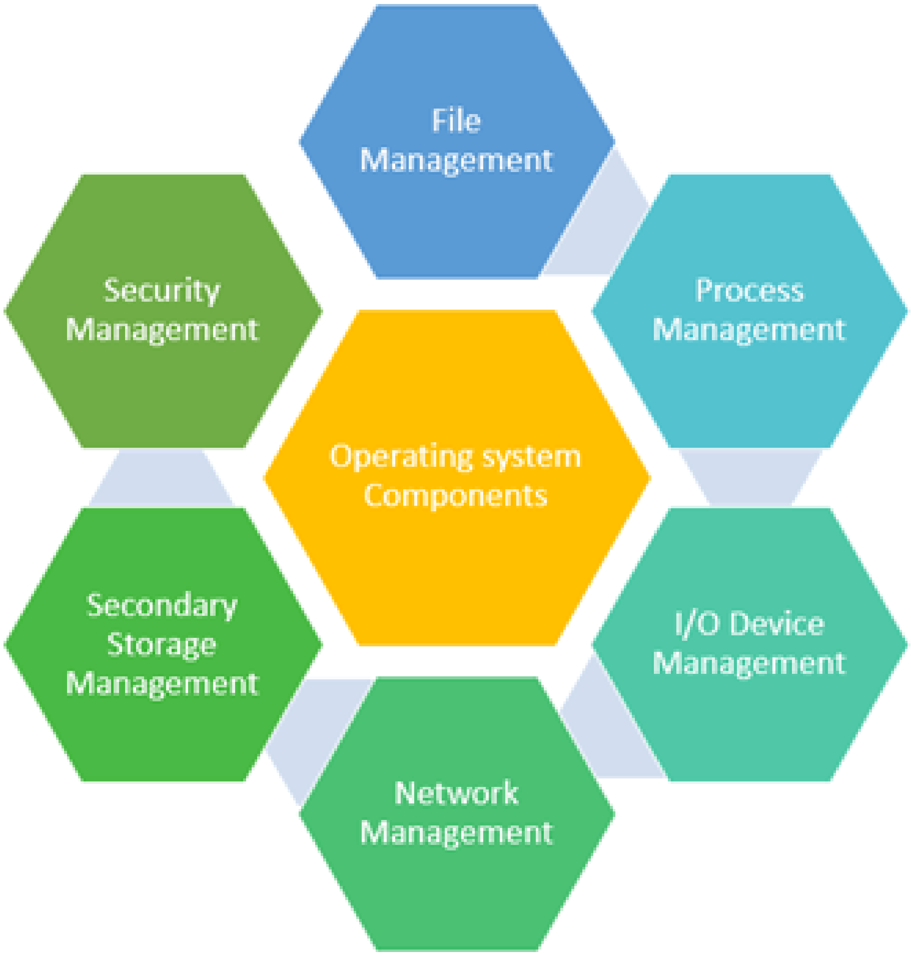
Available here: coding75.com

Join coding75 Pro 🚀

Introduction to OS

What is Operating System?

An Operating System acts like an interface between you and your computer's hardware. It handles tasks like running programs, managing resources, taking care of the CPU, and organizing files. The main goal of an operating system is to create a user-friendly and efficient environment for running programs.



Types of OS

- **Batch OS**: It works by storing similar jobs in the computer's memory. The CPU handles one job at a time, waiting for the current one to finish before moving on to the next.

- **Multiprogramming OS**: In this system, the computer's memory holds several jobs waiting for their turn with the CPU. When one job has to wait for something, like input/output operations, the operating system switches to another job. This way, the CPU is always busy, and you can have multiple tasks running simultaneously.

- **Multitasking OS**: Combining aspects of Multiprogramming and CPU scheduling, a multitasking OS quickly switches between different jobs. This rapid switching allows users to interact with each program as it runs, giving the impression of doing multiple tasks at the same time.

- **Time Sharing OS**: These systems involve direct interaction with the user. Users give instructions to the OS through input devices like keyboards, and the OS responds with output. It's like taking turns to use the computer.

- **Real-Time OS**: Designed for specific tasks with strict deadlines, real-time operating systems are often used in dedicated systems. They focus on completing particular jobs within specific time limits.

Various Operating System:

- **Windows**

Developed by Microsoft, Windows is a widely employed operating system for personal computers, known for its user-friendly interface, extensive software application support, and compatibility with diverse hardware configurations.

- **macOS:**

Crafted by Apple, macOS functions as the operating system for Apple Mac computers, featuring an elegant and intuitive user interface, smooth integration with other Apple devices, and a robust ecosystem of software applications.

- **Linux**

Linux, an open-source operating system, is highly customizable and extensively used in server environments and embedded systems. Recognized for its stability, security, and flexibility, Linux offers numerous distributions like Ubuntu, Fedora, and CentOS to meet varying user needs.

- **Unix**

Unix, a powerful multi-user operating system, serves as the foundation for many others, including Linux and macOS. Praised for its stability and security, Unix is particularly renowned for server applications.

- **Android**

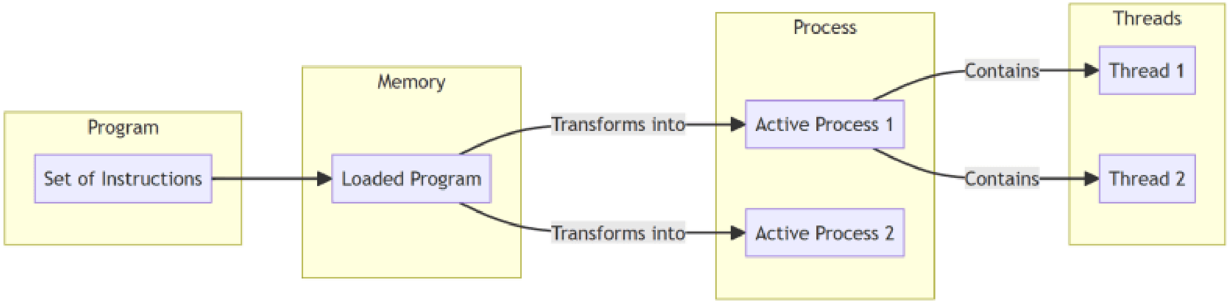
Developed by Google, Android is an open-source operating system designed primarily for mobile devices such as smartphones and tablets. It provides a diverse ecosystem of applications and allows for significant customization.

- **iOS**

Another creation of Apple, iOS operates as the operating system for iPhones, iPads, and iPods. Offering a seamless and secure user experience, iOS prioritizes performance and integrates smoothly with other Apple devices.

Program

A program is a **set of instructions** written in a programming language to perform a specific task. It exists as a static entity, typically stored on non-volatile storage like a disk, until loaded into memory for execution by the operating system.



Process

A process is a program under execution (which is currently running). The program counter (PC) shows the address of the next instruction in the executing process. Each process is identified and managed by a **Process Control Block (PCB).**

Process Scheduling:

1. **Arrival Time-** Time at which the process arrives in the ready queue.
2. **Completion Time-** Time at which process completes its execution.
3. **Burst Time-** Time required by a process for CPU execution.
4. **Turn Around Time-** Time Difference between completion time and arrival time.
    - Turn Around Time = Completion Time - Arrival Time
5. **Waiting Time (WT)-** Time Difference between turn around time and burst time.
    - Waiting Time = Turnaround Time - Burst Time

Thread

A thread is a **lightweight process** and is the basic unit for CPU usage. A process can handle multiple tasks simultaneously by incorporating multiple threads.

- A thread has its own program counter, register set, and stack.
- Threads within the same process share resources like the code section, data section, files, and signals.
- There are two types of threads:
    - User threads (Implemented by users)
    - Kernel threads (Implemented by the operating system)

Process vs Thread

Feature	Process	Thread
Definition	An independent program with its own resources.	A lightweight unit within a process that shares resources.
Memory Space	Each process has a separate memory space.	Threads within the same process share the same memory space.
Communication	Requires Inter-Process Communication (IPC).	Easier communication as threads share the same memory space.
Isolation	Processes are isolated from each other.	Threads within the same process are not isolated.
Parallel Execution	Processes can execute in parallel.	Threads within the same process may execute concurrently, but it depends on the system and programming model.

Multiprocessing

Multiprocessing involves the simultaneous **execution of multiple processes on a system equipped with multiple CPUs or CPU cores**. Each process represents a running program and runs independently with its own memory space and resources. The goal of multiprocess is to enhance system throughput and speed up execution by distributing the workload across several processors.

Multithreading

Multithreading, on the other hand, revolves around the **execution of multiple threads within a single process**. Threads are lightweight units of execution that share the **same memory space and resources**, allowing parallel execution within a process.

Multiprogramming

Multiprogramming is a strategy where **multiple programs are loaded into memory simultaneously**, and the CPU switches between them to execute instructions. The **objective is to maximize CPU utilization** by swiftly **switching between different programs, especially when one is waiting for I/O or other operations**. Each program operates within its distinct memory space.

Multitasking

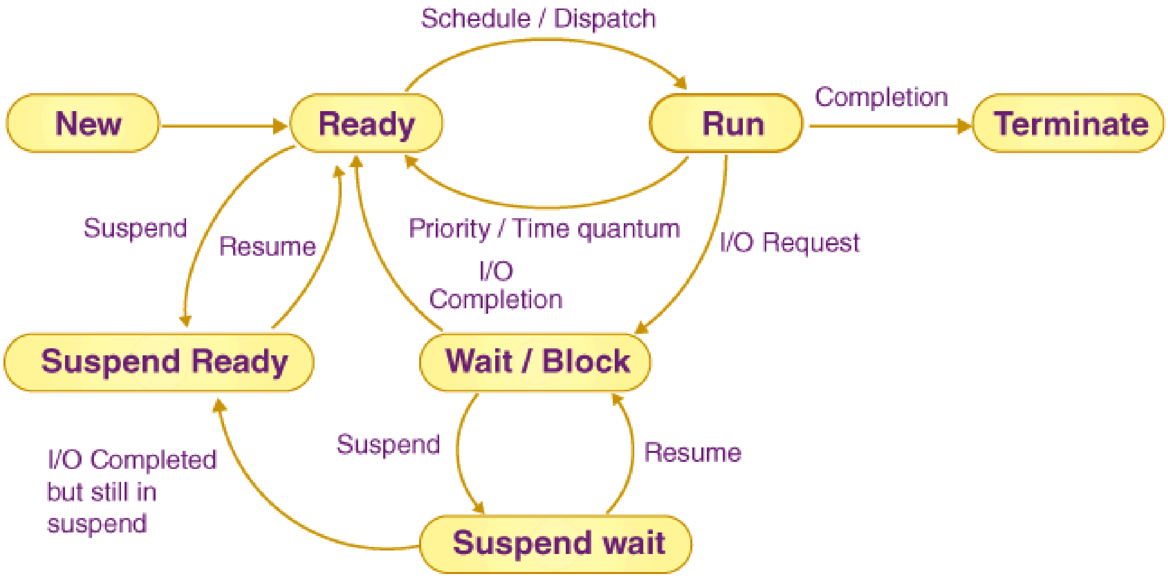
Multitasking is a technique that enables the **concurrent execution of multiple tasks or processes on a single CPU.** CPU time is divided among tasks, creating the illusion of parallel execution. Modern operating systems commonly use multitasking to provide responsiveness and the capability to run multiple applications simultaneously.

Interprocess Communication (IPC)

Interprocess Communication (IPC) serves as a communication bridge in a computer system, enabling different processes or threads to exchange information through shared resources like memory. It acts as an approved channel facilitated by the operating system, allowing processes to communicate and share data.

Process States in OS

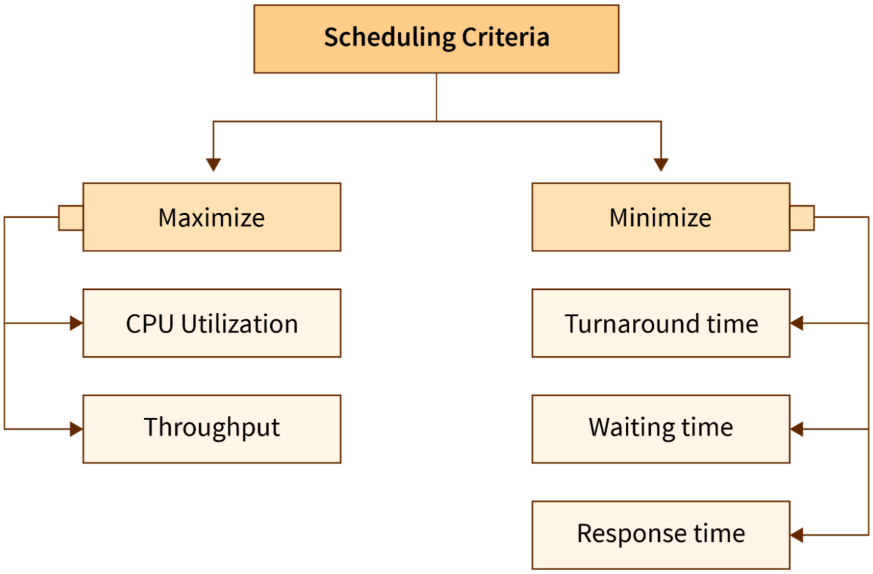
In an operating system, a process undergoes various states throughout its lifecycle. Here is the diagram:



Process State Diagram

CPU Scheduling Algorithms

CPU scheduling algorithms are essential components of operating systems that determine the order in which processes are executed by the CPU. These algorithms aim to maximize CPU utilization, throughput, and fairness among processes. Common CPU scheduling algorithms include:



- **First Come First Serve (FCFS)**: The most straightforward scheduling algorithm that arranges processes based on their arrival times.
- **Shortest Job First (SJF)**: Prioritizes processes with the shortest burst time, scheduling them first.
- **Shortest Remaining Time First (SRTF)**: A preemptive version of SJF where jobs are scheduled based on their remaining time, allowing for interruptions.
- **Round Robin (RR) Scheduling**: Each process gets a fixed time slice in a repeating sequence.

- **Priority-Based Scheduling (Non-Preemptive)**: Processes are scheduled according to their priorities, with the highest-priority process scheduled first. If two processes have equal priorities, scheduling follows their arrival times.
- **Highest Response Ratio Next (HRRN)**: Schedules processes based on the highest response ratio, which considers both waiting time and burst time. This algorithm helps prevent starvation.
    - Response Ratio = (Waiting Time + Burst Time) / Burst Time
- **Multilevel Queue Scheduling (MLQ)**: Processes are sorted into different queues based on priority. High-priority processes are placed in the top-level queue, and lower-priority processes are scheduled only after completion of top-level processes.
- **Multilevel Feedback Queue (MLFQ) Scheduling**: Allows processes to move between queues based on their CPU burst characteristics. If a process consumes excessive CPU time, it is moved to a lower-priority queue.

Starvation and Aging

In operating systems, when using scheduling methods like Priority Scheduling or Shortest Job First Scheduling, a problem called "**Starvation**" can occur. This happens when a process is unable to access the necessary resources for an extended time, leading to low-priority processes being blocked while high-priority processes proceed.

To address this issue, there's a technique known as "**Aging**." Aging helps prevent starvation by gradually increasing the priority of processes that have been waiting for resources for a long time. This ensures fairness in resource allocation, allowing low-priority processes to eventually access the resources they need for execution.

The Critical Section Problem

The critical section denotes a **specific part of code or a block where a process or thread interacts with a shared resource like a variable, file, or database.** To preserve data integrity and prevent conflicts, only one process or thread is permitted to enter the critical section at any given time. This precaution ensures that multiple processes do not interfere with each other, simultaneously modifying shared resources.

- **Critical Section**: This is the part of the code where shared variables are accessed and/or updated.
- **Remainder Section**: The rest of the program excluding the Critical Section.

Process synchronisation (Solution to critical section problem)

Process synchronization is like a **traffic signal to control the flow of vehicles at an intersection**. If multiple processes or threads concurrently handling different tasks, **process synchronization ensures their effective cooperation and communication to prevent conflicts and maintain a proper execution order**. This mechanism addresses potential issues like race conditions, data inconsistencies, or deadlocks that may arise when multiple processes or threads access shared resources simultaneously.

The **key requirements** of synchronization mechanisms include:

- **Mutual Exclusion:** The synchronization mechanism must enforce mutual exclusion, allowing only one process or thread to access a shared resource or enter a critical section at any given time. This prevents conflicts and ensures consistency during concurrent access.
- **Progress:** The synchronization mechanism should allow processes or threads to make progress by ensuring that at least one process/thread can enter the critical section when it desires to do so. It avoids situations where all processes/threads are blocked indefinitely, leading to a deadlock.
- **Bounded Waiting:** The synchronization mechanism needs to guarantee that a process or thread waiting to enter a critical section will eventually be granted access. This prevents starvation, ensuring that a process or thread doesn't wait indefinitely to access a shared resource.

Here are some ways to make sure that different processes or threads work well together:

- **Locks/Mutexes:** These are like special keys that only one process or thread can have at a time. It ensures that only one of them can use a shared resource or critical section. It's like having a key to a room that only one person can use at a time, making sure they have it to themselves.

- **Semaphores:**Think of semaphores like traffic signals for processes or threads. They control access to shared resources.They can be implemented as binary semaphores (mutexes) or counting semaphores. Counting semaphores allow a specified number of processes or threads to access a shared resource simultaneously. Semaphores provide mechanisms for mutual exclusion, signaling, and coordination.

- **Read-Write Locks:** Imagine a situation where some people want to read a book (multiple readers), and some want to write in it (writers). Read-write locks help by letting many people read at the same time, but only one person can write, ensuring things stay organized and don't get messed up. It's like allowing a group to read a book together while making sure only one person writes in it at a time.

Deadlocks

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

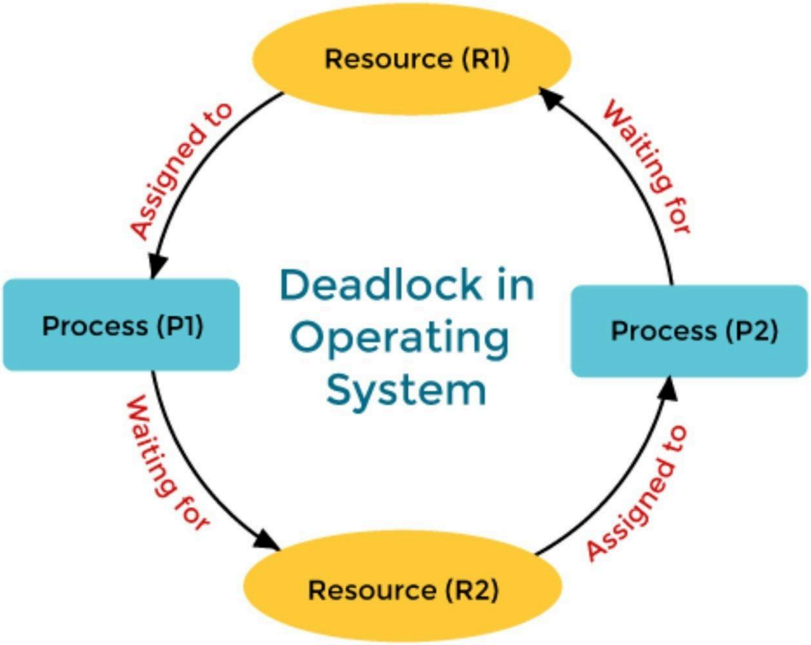
Deadlocks can occur if the following four conditions hold simultaneously (Necessary Conditions):

- **Mutual Exclusion**: One or more resources are non-sharable, meaning only one process can use them at a time.

- **Hold and Wait**: A process is holding at least one resource and waiting for additional resources.

- **No Preemption**: A resource cannot be taken from a process unless the process willingly releases it.

- **Circular Wait**: A set of processes is waiting for each other in a circular form.



Methods for Handling Deadlocks:

There are three ways to handle deadlocks:

- **Deadlock Prevention or Avoidance**: The idea is to prevent the system from entering a deadlock state in the first place.
- **Deadlock Detection and Recovery**: Allow the deadlock to occur, then use preemption to handle it once it has occurred.
- **Ignore the Problem Altogether**: If deadlocks are very rare, let them happen and reboot the system. This approach is taken by both Windows and UNIX.
- **Banker's Algorithm:** The Banker's algorithm is used to avoid deadlock and is one of the deadlock-avoidance methods. It is named after the

banking system, where a bank never allocates available cash in a way that it can no longer satisfy the requirements of all its customers.

Memory Management

Memory Management in an operating system **involves organizing and optimizing the use of a computer's memory**. It includes allocating memory to different processes, ensuring they don't interfere with each other, and freeing up memory when it's no longer needed.

Partition and Memory allocation

Let's talk about two ways of handling memory in computer systems:

- Fixed Partitioning
- Dynamic Partitioning.

Fixed Partitioning

In fixed partitioning, the computer's memory is divided into fixed-sized sections, and each section is assigned to a specific task or process. Each section has a set amount of memory, and this allocation remains the same throughout the process's execution.

This method is simple to implement and allows for quick memory allocation. However, it may lead to inefficient use of memory because, as tasks with varying memory needs run, some allocated memory might not be used, creating what's called internal fragmentation.

Dynamic Partitioning

Dynamic partitioning, also called variable partitioning, improves on fixed partitioning by allowing the system to allocate and deallocate memory dynamically based on the actual needs of processes.

Unlike fixed partitioning, dynamic partitioning adapts to varying memory requirements, leading to better memory utilization. However, efficiently managing memory allocation and deallocation becomes more complex.

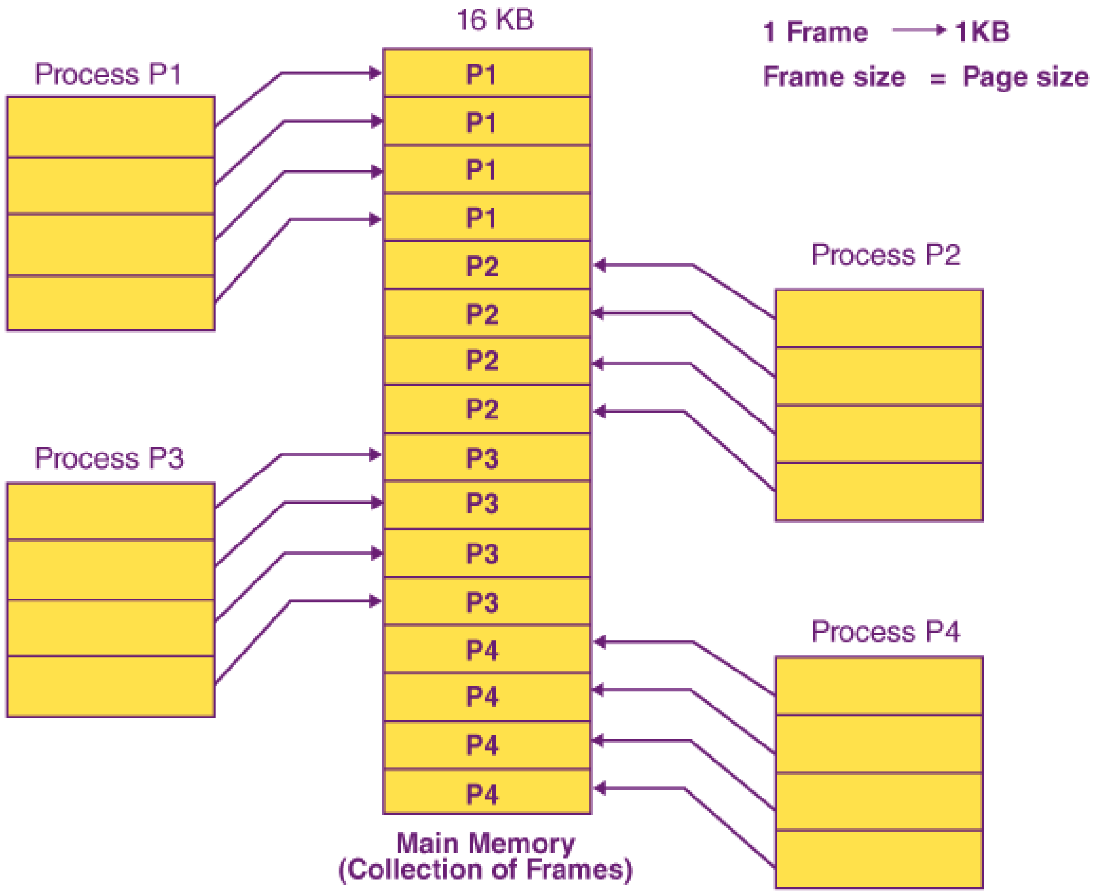
Techniques like compaction and memory allocation algorithms such as first-fit, best-fit, or worst-fit are used in dynamic partitioning systems to optimize memory usage and reduce fragmentation.

- **First Fit**: This algorithm allocates the first available memory block that is large enough for a process. It begins the search from the beginning of the memory and selects the first suitable block encountered. While simple and providing fast allocation, it may result in larger leftover fragments, impacting memory efficiency.
- **Best Fit:** The best-fit algorithm looks for the smallest available memory block that can accommodate a process. It aims to minimize leftover fragments by selecting the most optimal block. Although it can lead to better overall memory utilization, it may require more time-consuming searches to find the perfect fit.
- **Worst Fit:** The worst-fit algorithm assigns the largest available memory block to a process intentionally. This strategy keeps larger fragments for potential future larger processes. Despite seeming counterintuitive, it helps reduce fragmentation caused by small processes and enhances overall memory utilization.

Each algorithm comes with its trade-offs, impacting speed, efficiency, and the extent of leftover memory fragments.

Paging

Paging is a **storage strategy in Operating Systems** that involves bringing processes from secondary storage into the main memory as pages. The concept revolves around dividing each process into pages, and concurrently, the main memory is divided into frames. Each page of a process is stored in one frame of the memory. While pages can be stored in different memory locations, the focus is on finding contiguous frames or holes for optimal storage. **Pages are brought into the main memory only when needed; otherwise, they stay in secondary storage.**



Paging

Page Replacement Algorithms

Page Replacement Algorithms are strategies used by operating systems to decide which pages to replace in memory when a page fault occurs. Here are explanations for three of them:

- **First-In-First-Out (FIFO) Page Replacement Algorithm:**

FIFO maintains a queue of pages in the order they were brought into memory.

When a page fault occurs, the earliest page in the queue (the one brought in first) is selected for replacement.

The new page is brought into memory, and the oldest page in the queue is removed.

- **Optimal Page Replacement:**

This algorithm replaces pages that are not expected to be used for the longest duration in the future.

In theory, it provides optimal performance by minimizing page faults.

However, it's challenging to implement in real-world scenarios as it requires knowledge of future page references, which is usually not feasible.

- **Least Recently Used (LRU):**

LRU replaces the page that has been least recently used.

It operates on the principle that recently accessed pages are more likely to be used again soon.

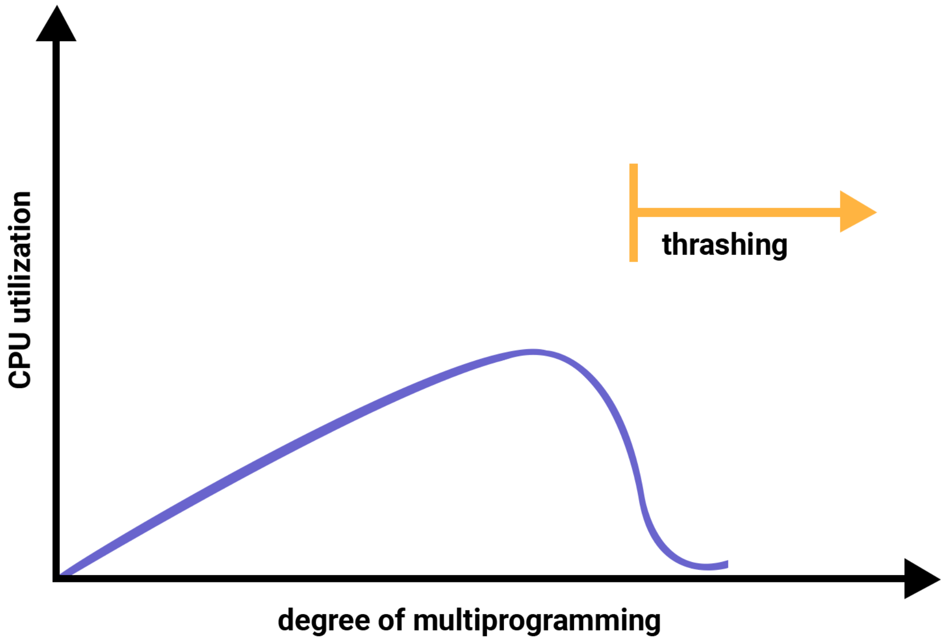
By replacing the least recently used pages, it aims to keep frequently accessed pages in memory, reducing page faults and improving overall system performance.

Page Fault

A page fault occurs when a running program attempts to access a memory page that is part of its virtual address space but is not currently loaded in the computer's physical memory. This triggers an interrupt at the hardware level.

Thrashing

Thrashing is a condition in computer systems **where a substantial amount of time and resources is consumed continuously swapping pages (Page Fault)** between the computer's physical memory (RAM) and secondary storage, like a hard disk. This excessive paging activity results in the system being preoccupied with moving pages in and out of memory rather than performing useful tasks, leading to a significant decline in overall performance.



Demand Paging

Demand paging is like **fetching parts of a book only when you're about to read them**, rather than bringing the entire book at once. Here's how it works:

- **Access Attempt:** When a computer program wants to use a piece of information (page), it checks if it's already in the memory (like a quick reference check).
- **Valid Page (In Memory):** If the needed information is already in the memory, great! The program keeps doing its thing.
- **Invalid Page (Page Fault):** If the required information isn't in the memory, it's like a signal (page fault) saying, "Hey, I need this part."

- **Memory Reference Check:** The computer makes sure the information is valid and can be found in another storage place (secondary storage).
- **Page-In Operation**: If all checks pass, a quick operation happens to bring that needed part into the memory.
- **Restart Instruction:** The program goes back to where it left off, now with the required information in hand.

So, demand paging helps the computer be efficient by only bringing in the necessary information when it's actually needed, making things smoother and saving memory space.

Virtual Memory

Virtual memory is a concept that **expands a computer's usable memory beyond its physical capacity.** It forms an imaginary memory space by combining RAM (physical memory) with secondary storage like a hard disk.

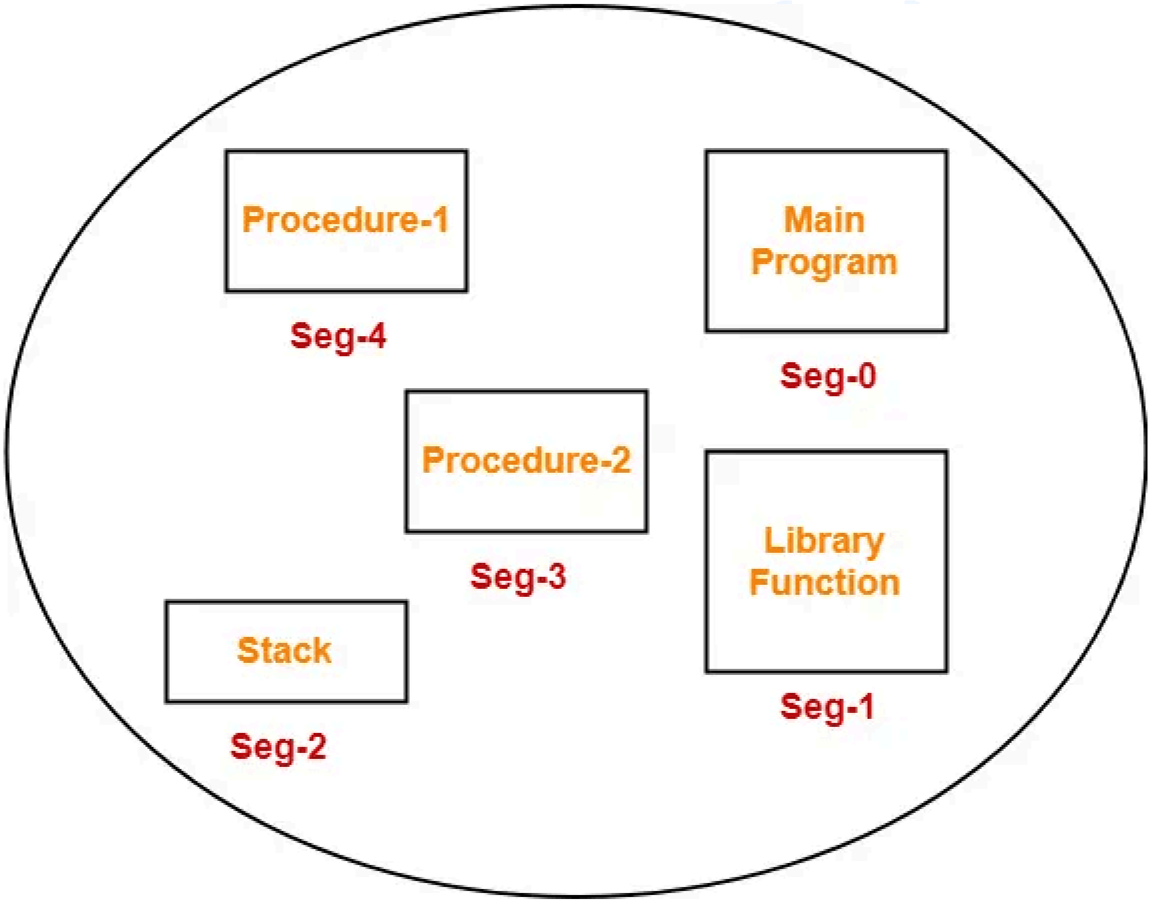
When a program requires more memory than available in RAM, **it temporarily transfers some data to secondary storage**. This enables the computer to execute larger programs and handle multiple tasks concurrently.

Virtual memory also ensures memory protection, preventing programs from directly accessing each other's memory. **Instead of loading a single extensive process into main memory, the Operating System loads various parts of multiple processes, increasing the degree of multiprogramming and enhancing CPU utilization.**

Segmentation

Segmentation is a memory management technique that **divides processes into smaller segments or modules**. Unlike paging, these segments do not need to be placed in contiguous memory locations, eliminating internal fragmentation. The length of each segment is determined based on the purpose of that segment within the user program.

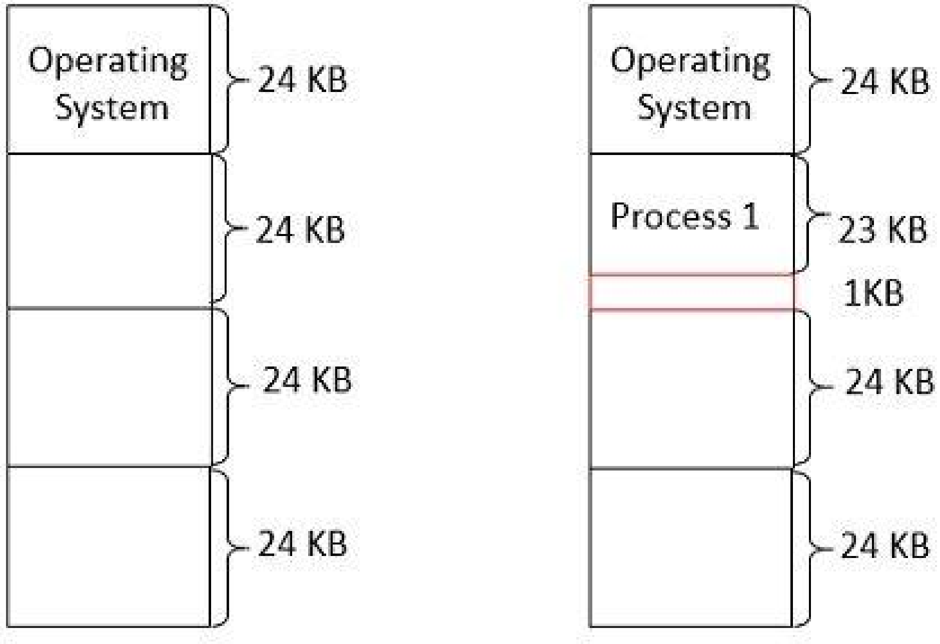
Segmentation was introduced to address issues with the paging technique. In paging, a function or code is divided into pages without considering the possibility of dividing related parts of the code. This results in the CPU having to load multiple pages into frames for the complete execution of related code.



Fragmentation

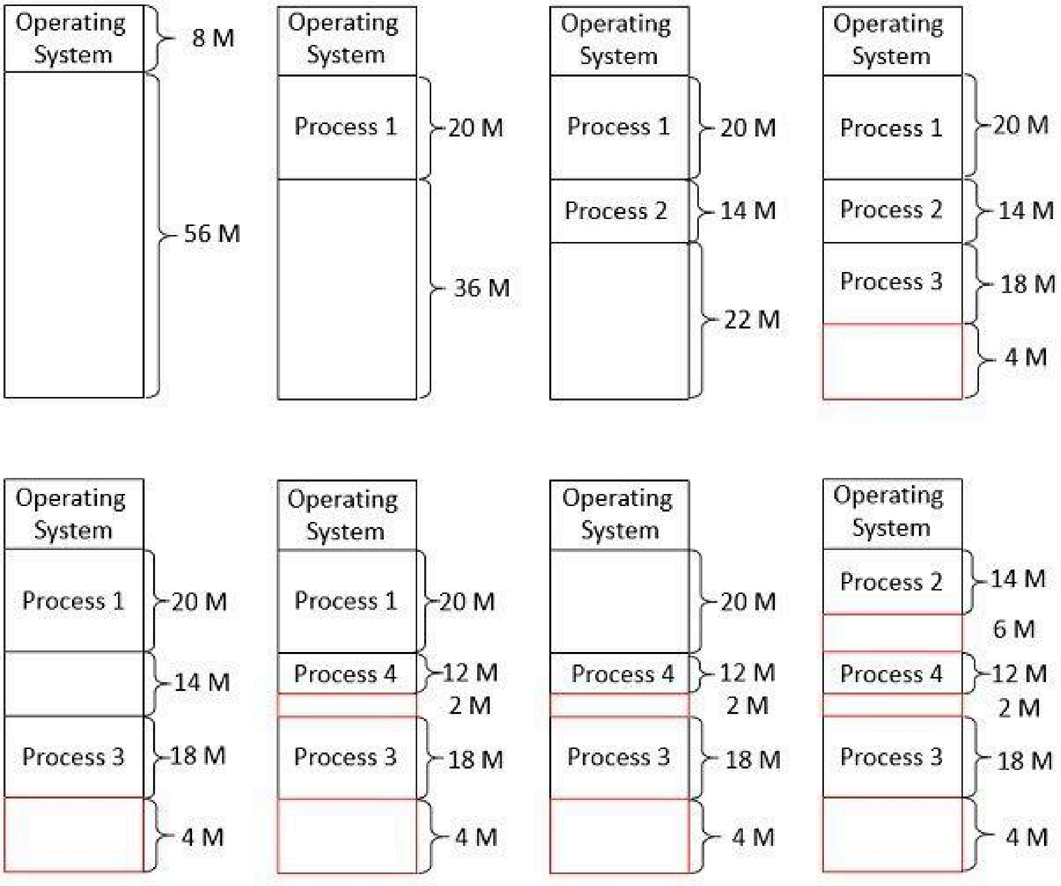
Fragmentation refers to the inefficient use of memory, leading to reduced capacity and performance. This wastage occurs in two main forms:

Internal Fragmentation: This type of fragmentation occurs in systems with fixed-size allocation units. It happens when allocated memory blocks are larger than the actual size of the data they hold. As a result, some space within the allocated block remains unused, contributing to internal fragmentation. This inefficiency can accumulate across multiple allocations, leading to a decrease in available memory for new processes.



Example of Internal Fragmentation

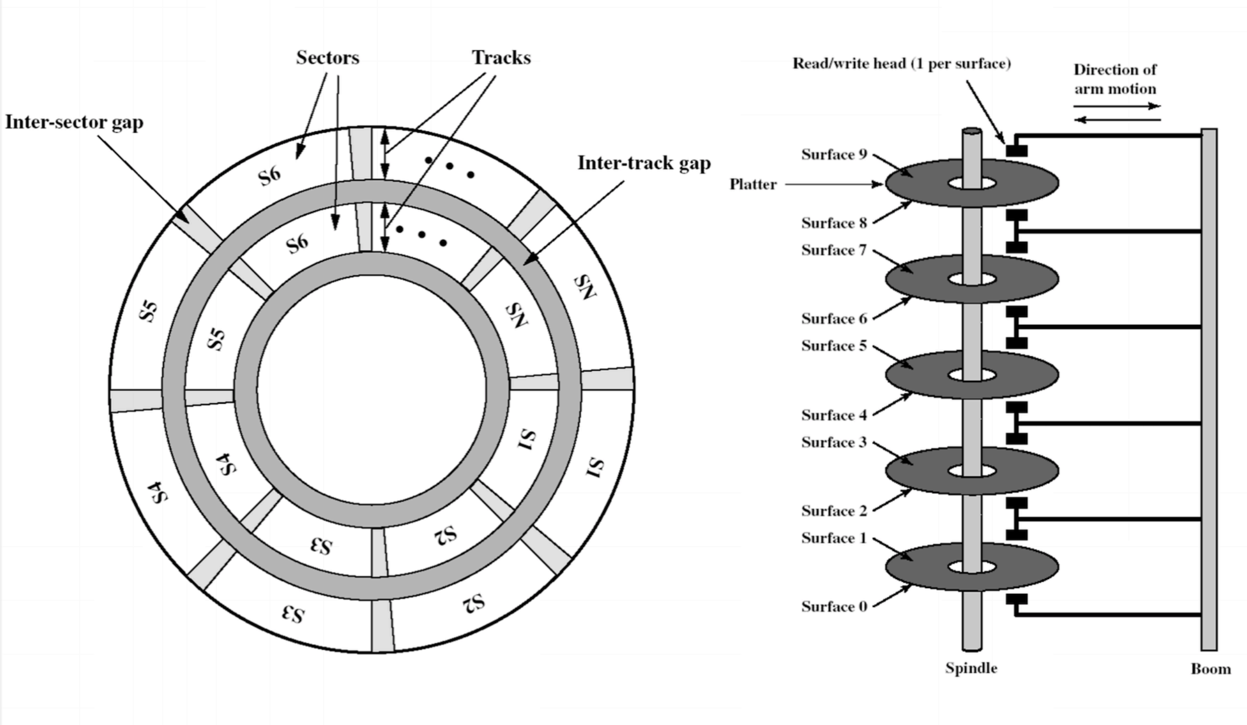
External Fragmentation: Systems with variable-size allocation units experience external fragmentation. This phenomenon arises when free memory exists in non-contiguous chunks, making it challenging to allocate larger memory blocks. Despite the total available memory being sufficient, the scattered nature of free space limits the system's ability to accommodate processes with larger memory requirements. Over time, external fragmentation can result in inefficient memory utilization and impact overall system performance.



Example of External Fragmentation

Disk Management

Disk management is a critical aspect of operating systems that involves efficient handling and organization of disk storage resources. It encompasses various tasks, including disk partitioning, formatting, and implementing disk scheduling algorithms to optimize Input/Output (I/O) operations.



Disk scheduling algorithms

Disk scheduling algorithms play a crucial role in optimizing disk I/O request processing and enhancing overall disk performance. Here are some notable disk scheduling algorithms:

- **FCFS (First-Come, First-Served)**
  - Description: Processes disk requests in the order they arrive.
  - Consideration: Does not account for disk head position or request proximity, potentially causing delays with long seek times.
- **SSTF (Shortest Seek Time First)**
  - Description: Chooses the request with the shortest seek time from the current disk head position.
  - Advantage: Minimizes average seek time, improving overall disk access time.
  - Drawback: May result in starvation for requests located farther away from the current position.
- **SCAN (Elevator Algorithm)**
  - Description: Moves the disk head in one direction, servicing requests along the way, then changes direction upon reaching the end.
  - Advantage: Provides fair service distribution, prevents starvation.
  - Drawback: Longer response times for requests at far ends of the disk.
- **C-SCAN (Circular SCAN)**
  - Description: Similar to SCAN but jumps to the other end of the disk without reversing direction.
  - Advantage: Consistent response time for all requests.
  - Drawback: May cause delays for requests arriving after the head has passed their location.

- **LOOK**
  - Description: A variant of SCAN that only goes as far as the last request in its current direction, then reverses direction.
  - Advantage: Reduces unnecessary traversal of the entire disk, improving response times.
- **C-LOOK (Circular LOOK)**
  - Description: Similar to C-SCAN, jumps to the other end of the disk without servicing requests along the way
  - Advantage: Reduces seek time and improves disk throughput.

Each algorithm has its strengths and weaknesses, and the choice depends on specific system requirements and characteristics.

Recap

Operating System Essentials in Short:

1. **Processes, Threads, and Multiprocessing:**

   Processes: Independent program instances.

   Threads: Smaller units within a process.

   Multiprocessing: Concurrent execution of multiple processes.

2. **Memory Management:**

Fixed Partitioning: Divides memory into fixed-sized partitions.

Dynamic Partitioning: Allocates memory dynamically.

Paging: Divides processes into fixed-sized pages.

Segmentation: Divides processes into modules.

Page Fault: Accessing a page not in physical memory.

3. **Replacement Algorithms:**

   FIFO: Replaces oldest page.

   Optimal: Replaces least-used page.

   LRU: Replaces least recently used.

4. **Disk Scheduling Algorithms:**

   FCFS: Processes requests in arrival order.

   SSTF: Services closest request first.

   SCAN: Moves head in one direction, reverses.

   C-SCAN: Circular SCAN with consistent response.

   LOOK: SCAN variant without unnecessary traversal.

   C-LOOK: Circular LOOK with improved throughput.

5. **Operating Systems:**

   Windows: User-friendly, diverse software support.

   macOS: Sleek interface, Apple ecosystem.

   Linux: Open-source, customizable, stable.

   Unix: Powerful, foundational for Linux, macOS.

   Android: Open-source, mobile-focused.

   iOS: Apple's mobile operating system.

6. **Process Synchronization:**

   Critical Section: Code portion with shared resources.

   Requirements: Mutual exclusion, progress, bounded waiting.

   Mechanisms: Locks, semaphores, read-write locks.

7. **Memory Management Approaches:**

   Fixed Partitioning: Predetermined partition sizes.

   Dynamic Partitioning: Variable partition sizes.

   Allocation Techniques: First Fit, Best Fit, Worst Fit.

8. **Virtual Memory:**

Extends RAM with secondary storage.

Allows running larger programs.

Ensures memory protection.

9. **Fragmentation:**

Internal: Wastage within fixed-size allocation units.

External: Wastage with variable-size allocation units.

10. **Disk Management:**

Partitioning: Divides disks into sections.

File Systems: Organize data on disks.

Formatting: Prepares disks for use.

RAID: RAID is a storage technology that combines multiple physical drives into a single logical unit for data storage. It provides enhanced performance, fault tolerance, or a combination of both.

Cleanup, Defragmentation: Optimizes disk space and performance.

Monitoring: Tracks disk health and usage.

Thank You!

Join Coding75 Pro 🚀

- Live DSA Classes
- Project Building Session
- 1:1 Mock Interviews
- Resume Review
- CS Fundamental Classes
- And much more...

Join Now