Operating Systems

- **Introduction:**

➢ **What is an Operating System?**

Operating System
An operating system is a software program that manages computer hardware and software resources, providing a platform for other software applications to run. It acts as an intermediary between the user and the computer hardware, enabling users to interact with the system and execute tasks effectively.

Purpose of an Operating System
The primary purpose of an operating system is to provide an environment in which applications can run smoothly and utilize the computer's resources effectively. It manages memory, processors, input/output devices, and other system resources to ensure optimal performance and stability.

Evolution of Operating Systems:
Operating systems have evolved significantly over the years, adapting to the changing needs of computer users. Let's take a brief look at the evolution of operating systems:

1. **Early Operating Systems:**
   In the early days of computing, operating systems were simple and mainly focused on managing hardware resources. Examples of early operating systems include the Batch Processing System and the Multiprogramming System.
2. **Mainframe Operating Systems:**
   With the advent of mainframe computers, operating systems became more sophisticated and capable of supporting multiple users and tasks simultaneously. Examples of mainframe operating systems include IBM's z/OS and UNIVAC's EXEC 8.
3. **Personal Computer Operating Systems:**
   The rise of personal computers brought about the need for operating systems tailored to individual users. Operating systems such as Microsoft Windows and Apple macOS

became popular choices for personal computer users, offering user-friendly interfaces and a wide range of applications.

4. **Modern Operating Systems:**
   Today, modern operating systems like Linux have gained popularity due to their open-source nature, flexibility, and robustness. Mobile operating systems like Android and iOS have also emerged, powering smartphones and tablets with advanced features and seamless app integration.

General Tasks carried by OS –

1. Handling UI
2. Handling input/output
3. Process management
4. File Management
5. Memory Management
6. Security
7. other devices control like printer, SD card

Main Jobs handled by OS:

1. Device Management
2. File Management
3. Memory Management
4. Process Management
5. UI Management
6. Storage Management
7. Application Handling

Functions of an Operating System

An operating system performs various functions to facilitate the operation of a computer system. These functions include:

1. **Process management:** It manages the execution of processes, scheduling them for execution, and allocating system resources to them.
2. **Memory management:** It controls the allocation and deallocation of memory space to processes, ensuring efficient memory utilization.
3. **File system management:** It provides a hierarchical structure for organizing and accessing files on storage devices.
4. **Device management:** It interacts with hardware devices, such as printers, scanners, and disk drives, to facilitate data transfer and communication.
5. **User interface:** It provides a means for users to interact with the system, such as through a command-line interface or a graphical user interface.

Popular Operating Systems

There are several popular operating systems in use today, catering to different computing needs. Let's explore some of the widely used operating systems:

Windows

Microsoft Windows is a widely used operating system for personal computers. It offers a user-friendly interface, extensive software compatibility, and a range of features for both home and business users.

macOS

macOS is the operating system developed by Apple Inc. for its Macintosh computers. Known for its sleek design and seamless integration with other Apple devices, macOS provides a user-friendly and secure computing experience.

Linux

Linux is an open-source operating system that has gained popularity for its stability, security, and versatility. It powers a wide range of devices, from servers to smartphones, and offers a vast array of software options.

Android

Android is a mobile operating system developed by Google. It powers a majority of smartphones and tablets worldwide, offering a customizable user interface, extensive app ecosystem, and integration with Google services.

iOS

iOS is the operating system developed by Apple Inc. for its mobile devices, including iPhones and iPads. It provides a seamless and secure user experience, along with access to a wide range of apps optimized for Apple devices.

- **Types of Operating Systems**

Different kinds of operating systems are available some of the important ones are –

- Batch System
- Distributed Operating System
- Time Sharing System
- Desktop System
- Multiprocessor System
- Clustered System
- Realtime Operating System
- Handheld System

As we all know that a computer has lots of inbuilt power and mathematical computing available. The **operating system manages all the power** provided by the computing system and performs basic tasks like managing different files, processes and UI rendering thus, you can call an operating system as the as the basic unit which is responsible to perform all the task.
For different kinds of computers different types of OSes are suitable we will discuss most important ones detail below –

Batch System:

- **Definition:** A batch system is an operating system that processes a set of similar tasks or jobs in a batch mode without any user interaction.
- **Characteristics**: It executes jobs one after another, without requiring user intervention for each task.

- **Usage**: Batch systems are commonly used in scenarios where a large number of similar tasks need to be processed efficiently, such as payroll processing or large-scale data processing.

Real-Time Operating Systems (RTOS):

- Prioritize predictability and timely execution for critical systems.
- Specifically designed to handle tasks with strict timing requirements.

Distributed Operating System:

- **Definition**: A distributed operating system is a system that runs on multiple interconnected computers and enables them to work together as a single cohesive unit.
- **Characteristics**: It allows resource sharing, load balancing, and fault tolerance across multiple machines in a network.
- **Usage**: Distributed operating systems are used in distributed computing environments where multiple computers collaborate on a task, such as in cloud computing or distributed databases.

Multi-User Operating Systems:

- Allow multiple users to access and utilize a computer system simultaneously.
- Provide mechanisms for user authentication, resource sharing, and process isolation.

Time Sharing System:

- **Definition**: A time-sharing system, also known as a multitasking system, allows multiple users to concurrently share the resources of a single computer.
- **Characteristics**: It divides the CPU time among multiple users, providing each user with the illusion of having their own computer.
- **Usage**: Time-sharing systems are commonly used in scenarios where multiple users need to access and utilize a computer's resources simultaneously, such as in interactive computing or server environments.

Single-User Operating Systems:

- Designed to support one user at a time.

 Prioritize individual productivity on personal computers and workstations.

 Examples include Microsoft Windows, macOS, and Linux distributions like Ubuntu and Fedora.

Desktop System:

- **Definition**: A desktop system refers to an operating system designed for personal computers or workstations used by individual users.
- **Characteristics**: It provides a graphical user interface (GUI) and a range of applications for general-purpose computing tasks.
- **Usage**: Desktop systems are widely used by individuals for tasks such as web browsing, document processing, multimedia consumption, and software development

Modern Implementations and Applications of Batch Operating Systems

Modern applications and implementations of operating systems are diverse and play a crucial role in numerous technological advancements. Here are some notable examples:

1. Mobile Devices and Smartphones:
    - Operating systems like Android and iOS power mobile devices, providing a platform for a wide range of applications.
    - These operating systems enable seamless communication, app distribution, and integration with various mobile hardware components.
2. Internet of Things (IoT):
    - Operating systems designed specifically for IoT devices manage the connectivity and interaction between smart devices.
    - These operating systems enable efficient data processing, secure communication, and remote management of IoT ecosystems.
3. Cloud Computing:
    - Operating systems play a significant role in cloud computing infrastructure, managing virtual machines and containerized environments.
    - They ensure resource allocation, scalability, and secure data handling in cloud platforms like Amazon Web Services (AWS) and Microsoft Azure.

- **Overview of a Batch Operating System**

A batch operating system is a type of operating system that processes a series of tasks, known as jobs, without requiring constant user intervention.

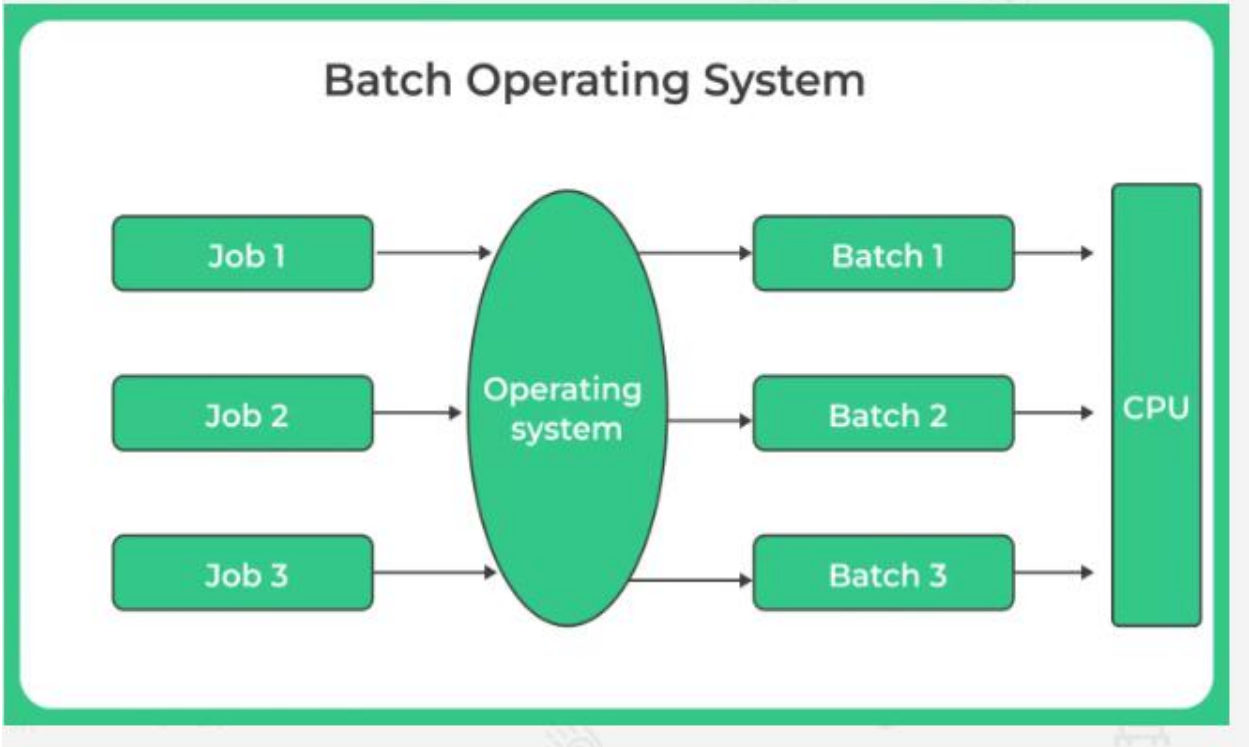
Unlike interactive systems, which rely on user input for each task, batch systems execute a set of jobs in sequence, one after another. This approach allows for efficient utilization of computer resources and streamlines the execution of repetitive tasks.

Batch Processing in Modern Computing

While interactive and time-sharing systems dominate the modern computing landscape, batch processing remains relevant in various scenarios. In data-intensive environments, such as financial institutions and scientific research centers, batch processing is utilized for large-scale data analysis, report generation, and bulk data processing tasks. Batch systems continue to provide efficient and reliable processing capabilities for such workloads.

Are batch operating systems still in use today?Yes, batch operating systems are still utilized in specific industries that require large-scale data processing and efficient execution of repetitive tasks.

Can batch operating systems handle real-time processing?Batch systems are not designed for real-time processing, as they prioritize efficient execution of tasks in batches rather than immediate responsiveness.



Characteristics and Features of Batch Operating Systems

- **Job Sequencing:** Batch operating systems execute tasks in a sequential order, known as job sequencing, without requiring constant user intervention.
- **Minimal User Interaction:** Batch systems minimize the need for user interaction by automating the execution of tasks, allowing users to submit jobs and let the system handle their processing.
- **Queue-Based Processing:** Batch systems typically employ a queue-based mechanism for job scheduling and resource allocation, ensuring efficient utilization of system resources.
- **Priority-Based Execution:** Jobs in a batch operating system can be assigned priority levels, allowing for the execution of higher-priority jobs before lower-priority ones.

Advantages of Batch Operating Systems:

- **Efficient Resource Utilization:** Batch systems allow for continuous processing of tasks without manual intervention, maximizing resource utilization and increasing productivity.
- **Improved Throughput:** By executing jobs in batches, batch operating systems can process a large number of tasks sequentially, resulting in higher throughput and faster job execution.
- **Streamlined Execution:** Batch systems simplify the execution of repetitive tasks by automating the process, reducing the need for user interaction and minimizing errors.
- **Better Control over Resources:** Jobs can be prioritized and scheduled based on their requirements, allowing for effective resource allocation and optimal utilization of system resources.
- **Scalability:** Batch systems can handle large volumes of data and scale to accommodate increasing workloads, making them suitable for data-intensive applications.

Modern Implementations and Applications of Batch Operating Systems

Although the prominence of batch operating systems has diminished in the era of interactive and time-sharing systems, they still find application in specific domains. Industries that require large-scale data processing, such as finance, healthcare, and scientific research, rely on batch systems

for their computational needs. These systems offer reliability, efficiency, and scalability for processing vast amounts of data and executing complex calculations.

- **Distributed Operating System in OS**

Overview of a Distributed Operating System

A distributed operating system refers to an operating system that runs on a network of computers and enables them to work together as a single entity. The primary purpose of a distributed operating system is to achieve efficient resource sharing, fault tolerance, and scalability.

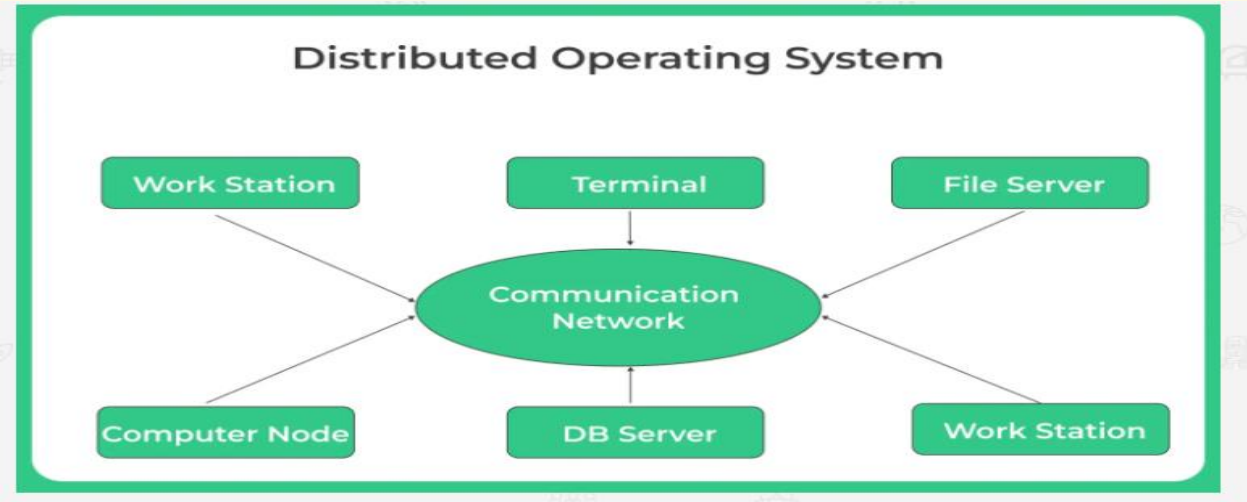
Distributed operating systems are specifically designed to distribute computing tasks across multiple interconnected systems, providing several advantages over traditional operating systems.

Batch Processing in Modern Computing

While interactive and time-sharing systems dominate the modern computing landscape, batch processing remains relevant in various scenarios. In data-intensive environments, such as financial institutions and scientific research centers, batch processing is utilized for large-scale data analysis, report generation, and bulk data processing tasks. Batch systems continue to provide efficient and reliable processing capabilities for such workloads.

How do distributed operating systems handle data consistency?Distributed operating systems employ various techniques, such as replication, distributed transactions, and consensus algorithms, to ensure data consistency across multiple nodes.

Can distributed operating systems be used in real-time applications?Yes, distributed operating systems can be adapted for real-time applications by employing real-time scheduling algorithms and ensuring timely communication and response.



Key Components of Distributed Operating Systems To understand how distributed operating systems function, it's important to explore their key components.

- **Network communication protocols:** Effective communication between nodes is crucial in distributed operating systems. Various network communication protocols, such as TCP/IP, UDP, and RPC, enable seamless interaction and data transfer between different machines.
- **Resource management and synchronization:** In distributed environments, managing and synchronizing resources becomes vital. Distributed operating systems employ techniques like distributed file systems and distributed shared memory to enable efficient resource management and synchronization.
- **Distributed file systems:** Distributed file systems distribute files across multiple machines while providing transparent access to users. Examples of distributed file systems include Google File System (GFS) and Hadoop Distributed File System (HDFS).
- **Process coordination and communication:** Coordinating and communicating processes across distributed systems is essential for achieving parallel processing and load balancing. Distributed operating systems utilize message passing, remote procedure calls (RPC), and other mechanisms to facilitate process coordination and communication.

Advantages of Distributed Operating Systems:

- **Resource Sharing:** Distributed operating systems allow for efficient sharing of computing resources across multiple machines, improving resource utilization.
- **Scalability:** They enable easy scalability by adding or removing nodes from the network, accommodating changing workloads.
- **Fault Tolerance:** Distributed operating systems provide fault tolerance by distributing data and processes, ensuring system reliability even in the presence of failures.
- **Performance Improvement:** Parallel processing and load balancing techniques employed in distributed operating systems can significantly enhance system performance.

Future Trends in Distributed Operating Systems

The field of distributed operating systems continues to evolve. Some potential future trends include:

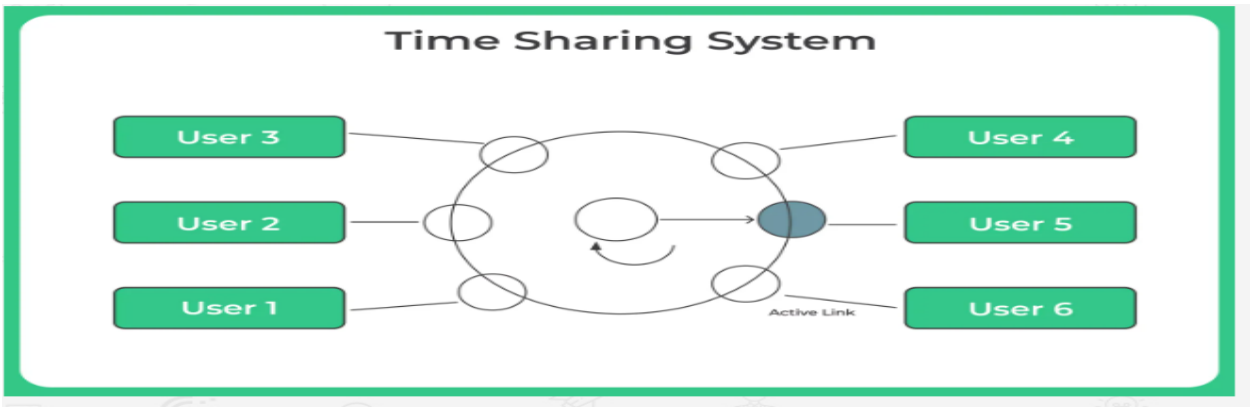
- Integration of machine learning and artificial intelligence techniques to optimize resource allocation and performance in distributed systems.
- Enhanced fault tolerance mechanisms to handle dynamic and unpredictable environments.
- Adoption of blockchain technology to provide secure and transparent transactional capabilities in distributed operating systems.

➢ Time Sharing System in OS

How Does a Time Sharing System Work?

A time sharing system works by rapidly switching the execution context between multiple users or processes. Each user or process is given a fair and equal amount of time to execute their tasks. The time slices are typically very small, ranging from a few milliseconds to a few microseconds, and the context switch between tasks happens so quickly that it creates an illusion of concurrent execution.

> **Time Sharing vs. Batch Processing?**Time sharing systems differ from batch processing systems, which execute a sequence of jobs without user interaction. In batch processing, users submit their jobs to a queue, and the system executes them in a sequential manner. Time sharing systems, on the other hand, allow users to interact with the system in real-time and execute their tasks concurrently.



Characteristics and Features of Time Sharing Operating Systems

- **Interactive and Concurrent Access**
  One of the primary characteristics of a time sharing system is its ability to provide interactive and concurrent access to multiple users. Each user can initiate their tasks, execute commands, and receive prompt responses from the system. Users can work concurrently without significant delays or interruptions, enhancing productivity and collaboration.
- **Fairness and Equal Time Allocation**
  Fairness is a crucial aspect of a time sharing system. It ensures that each user or process receives an equal share of the computing resources. Time slices, also known as quanta, are allocated to each user or process, ensuring a fair distribution of CPU time. This fairness prevents any user or process from monopolizing the system resources and promotes equitable resource utilization.
- **Responsiveness and Real-Time Interaction**
  Time sharing systems prioritize responsiveness to provide users with a real-time interaction experience. Users can input commands and receive immediate feedback or results. The system quickly switches between tasks, giving the impression of simultaneous execution. This responsiveness creates a smooth and interactive user experience, contributing to user satisfaction and productivity.
- **Process Scheduling**
  Process scheduling is a critical feature of a time sharing system. It involves determining the order in which processes or tasks are executed and allocating time slices to each process. Scheduling algorithms, such as Round Robin, Priority Scheduling, and Multilevel Queue Scheduling, are employed to ensure fairness, optimize resource utilization, and meet specific system requirements.
- **Context Switching**
  Context switching is a key mechanism in a time sharing system. It enables the system to switch rapidly between different processes or tasks. During a context switch, the system saves the current execution context of a task and restores the context of the next task to be executed. Efficient context switching minimizes overhead and facilitates seamless transitions between tasks, contributing to overall system performance.

Advantages of Time System Sharing Systems:

- **Enhanced User Productivity**
  One of the significant advantages of time sharing systems is their ability to enhance user productivity. By allowing multiple users to work simultaneously on a single system, time sharing eliminates idle time and maximizes user engagement. Users can execute tasks, run applications, and receive prompt responses, resulting in increased productivity and efficient task completion.

- **Efficient Resource Utilization**
  Time sharing systems maximize the utilization of system resources, such as the CPU, memory, and peripherals. By sharing resources among multiple users or processes, the system ensures that resources are efficiently utilized without any significant downtime. This approach reduces resource wastage and optimizes the overall system performance.
- **Cost Savings**
  Implementing time sharing systems can lead to significant cost savings. Instead of each user or organization maintaining individual systems, time sharing allows for the sharing of resources, reducing hardware and maintenance costs. Multiple users can leverage the same set of resources, resulting in a cost-effective solution for computing needs.
- **Improved System Responsiveness**
  Time sharing systems prioritize system responsiveness, providing users with a real-time interactive experience. Users can input commands, receive immediate feedback, and witness prompt execution of tasks. The quick context switching between processes creates an illusion of simultaneous execution, making the system highly responsive and interactive.

Modern Implementations and Applications of Time Sharing Systems

Time-sharing systems in modern operating systems have various implementations and applications. They enable multitasking, interactive computing, server virtualization, cloud computing, real-time systems, collaboration, and batch processing. These systems allow efficient resource utilization, facilitate concurrent execution of multiple tasks or processes, and provide a responsive and interactive user experience.

➢ Multiprocessor Operating Systems in OS

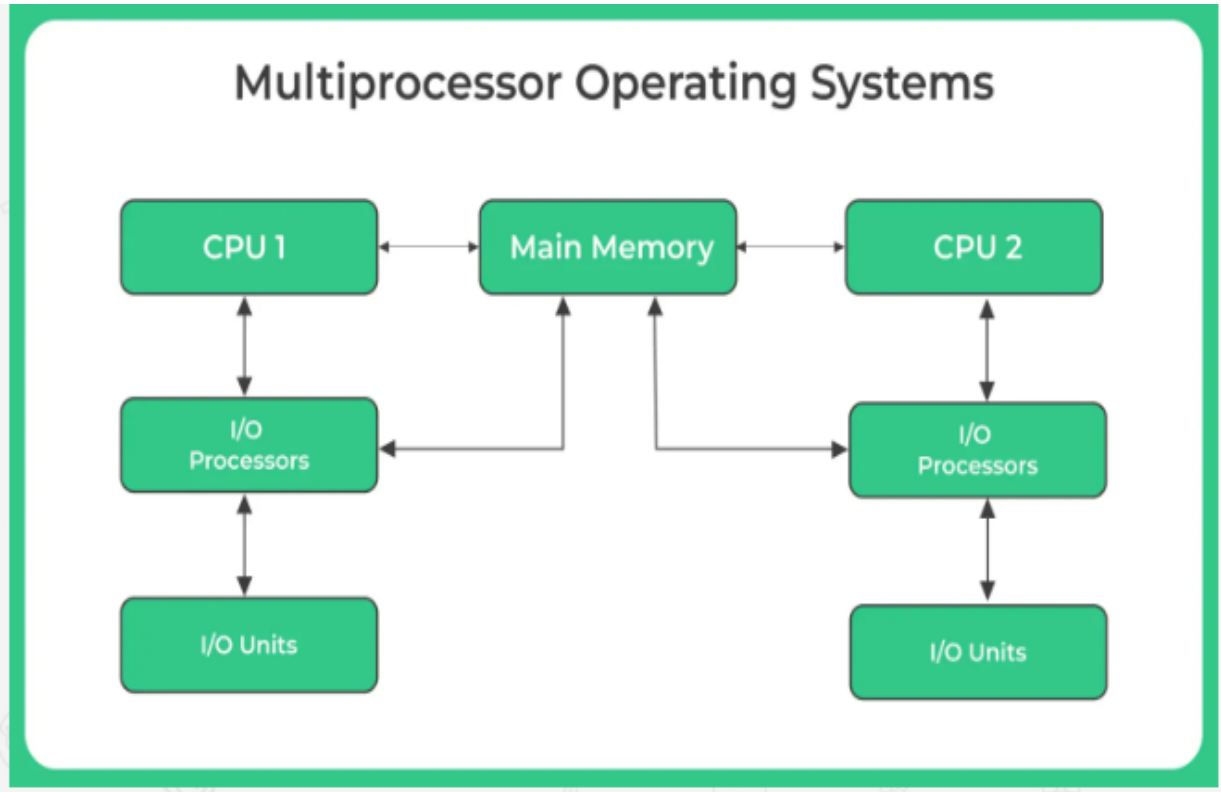
Overview of Multi Processor Operating System

When two or more central processing units operate within a single computer system, it is referred as a multiprocessor system. In such computers, these multiple CPU's have a close connection and communication to share data and programs. These multiple processors also share operating system resources such as memory, buses, printers, and other peripheral devices.

Multiprocessor systems are used when users need extremely high processing speed to process a high volume of data. In most of the cases, such operating systems are used to carry out scientific calculations or operations such as satellite processing, deep data analysis, and weather forecasting.

What are some benefits of multiprocessor operating systems?Multiprocessor operating systems offer enhanced performance, improved reliability, scalability, and load balancing, allowing for efficient utilization of multiple processors.

Which operating systems provide support for multiprocessor systems?Examples of operating systems that support multiprocessor systems include Linux, Windows Server, and Solaris.



Advantages of Multiprocessor Operating Systems

- **Enhanced Performance and Throughput:** Multiprocessor operating systems leverage parallelism to enhance system performance and increase throughput. By distributing tasks among multiple processors, these systems can execute multiple instructions simultaneously, leading to faster execution times and improved overall system responsiveness.
- **Improved Reliability and Fault Tolerance:** Multiprocessor systems provide built-in redundancy, which enhances system reliability and fault tolerance. If one processor fails, the remaining processors can continue to execute tasks without any disruption. This fault-

tolerant capability is especially critical in mission-critical applications where system downtime is unacceptable.

- **Scalability and Load Balancing:**
  Multiprocessor operating systems offer excellent scalability by allowing additional processors to be added to the system as needed. This scalability enables organizations to accommodate increasing workloads without replacing the entire system. Furthermore, load balancing algorithms ensure that tasks are evenly distributed among processors, preventing bottlenecks and maximizing resource utilization.

Key Features and Techniques in Multiprocessor Operating Systems

- **Process and Thread Scheduling:** Multiprocessor operating systems employ advanced process and thread scheduling algorithms to efficiently utilize available processors. Techniques such as load balancing, priority-based scheduling, and affinity scheduling ensure optimal processor allocation and maximize system performance.
- **Inter-Process Communication (IPC):** Inter-Process Communication (IPC) mechanisms facilitate communication and data exchange between processes running on different processors. Techniques such as message passing, shared memory, and remote procedure calls enable efficient coordination and collaboration among processes.
- **Memory Management and Allocation:** Memory management in multiprocessor operating systems involves efficient allocation and deallocation of memory resources. Techniques such as page-based memory management and dynamic memory allocation ensure optimal memory utilization and avoid fragmentation.
- **I/O Handling and Device Management:** Efficient I/O handling and device management are essential for multiprocessor operating systems. Techniques such as asynchronous I/O, interrupt-driven I/O, and device virtualization enable parallel processing of I/O operations and maximize system throughput.

Future Trends and Developments in Multiprocessor Operating Systems

The field of multiprocessor operating systems is continually evolving, driven by advancements in hardware technologies and the increasing demand for high-performance computing. Future trends include enhanced support for heterogeneous processors, improved energy efficiency, and the integration of machine learning techniques to optimize system performance.

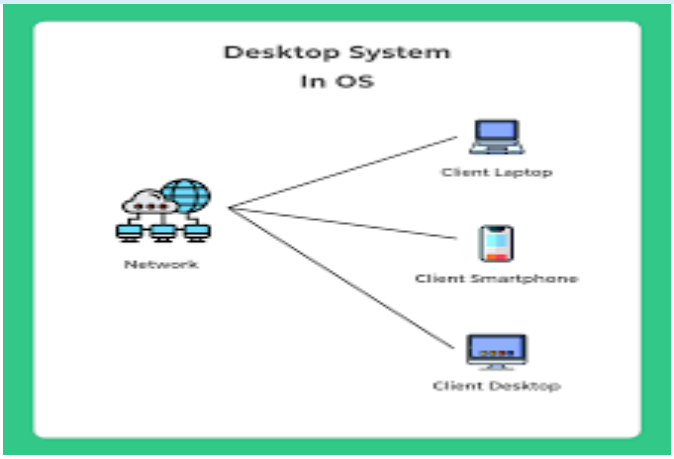
➢ Desktop Systems in OS

Operating Systems for Desktop Systems

An operating system (OS) acts as an interface between the hardware and software of a desktop system. It manages system resources, facilitates software execution, and provides a user-friendly environment. Different operating systems offer distinct features, compatibility, and performance, catering to the diverse needs and preferences of users.

Importance of Desktop Systems

- Desktop systems play a crucial role in various domains, including education, business, entertainment, and personal productivity.
- They provide individuals and organizations with powerful computing capabilities, enabling complex tasks to be completed efficiently.
- Desktop systems facilitate creativity, communication, data analysis, and knowledge sharing, contributing to enhanced productivity and innovation



Components of a Desktop System

- **Central Processing Unit (CPU):** The CPU is the brain of a desktop system, responsible for executing instructions and performing calculations. It processes data and carries out

tasks based on the instructions provided by software programs. The CPU's performance is measured by its clock speed, number of cores, and cache size.

- **Random Access Memory (RAM):** RAM is a type of volatile memory that temporarily stores data and instructions for the CPU to access quickly. It allows for efficient multitasking and faster data retrieval, significantly impacting the overall performance of the system. The amount of RAM in a desktop system determines its capability to handle multiple programs simultaneously.
- **Storage Devices:** Desktop systems utilize various storage devices to store and retrieve data. Hard Disk Drives (HDDs) are the traditional storage medium, offering large capacities but slower read/write speeds. Solid-State Drives (SSDs) are a newer technology that provides faster data access, enhancing the system's responsiveness and reducing loading times.
- **Graphics Processing Unit (GPU):** The GPU is responsible for rendering images, videos, and animations on the computer screen. It offloads the graphical processing tasks from the CPU, ensuring smooth visuals and enabling resource-intensive applications such as gaming, video editing, and 3D modeling. High-performance GPUs are essential for users who require demanding graphical capabilities.
- **Input and Output Devices:** Desktop systems are equipped with various input and output devices. Keyboards and mice are the primary input devices, allowing users to interact with the system and input commands. Monitors, printers, speakers, and headphones serve as output devices, providing visual or auditory feedback based on the system's output.

Evolution of Desktop Systems

Desktop systems have evolved significantly over the years. From the bulky and limited-capability systems of the past to the sleek and powerful computers of today, technological advancements have revolutionized the desktop computing experience.

Smaller form factors, increased processing power, improved storage technologies, and enhanced user interfaces are some of the notable advancements that have shaped the evolution of desktop systems.

Popular Desktop Operating Systems

- **Windows:** Windows, developed by Microsoft, is one of the most widely used desktop operating systems globally.
- **macOS:** macOS is the operating system designed specifically for Apple's Mac computers. Known for its sleek and intuitive interface, macOS offers seamless integration with other Apple devices and services.
- **Linux:** Linux is an open-source operating system that provides a high degree of customization and flexibility. It is favored by developers, system administrators, and tech enthusiasts due to its stability, security, and vast array of software options.

Future Trends in Desktop Systems

The future of desktop systems holds exciting possibilities. As technology continues to advance, we can expect further improvements in processing power, storage capacities, and energy efficiency. Virtual reality (VR) and augmented reality (AR) integration, cloud-based computing, artificial intelligence (AI) integration, and seamless connectivity across devices are some of the trends that will shape the future of desktop systems.

➢ Clustered System in OS

Overview of a Clustered System

A clustered system refers to a group of interconnected computers, or nodes, that collaborate closely to perform tasks in parallel. These nodes work together as a single system, sharing resources and distributing workload effectively. By leveraging the combined power of multiple machines, clustered systems achieve high availability, scalability, and improved performance.

Types of Clustered Systems

Clustered systems come in various forms, each tailored to specific requirements. Let's delve into the three common types of clustered systems:

High-Availability Clusters High-availability clusters focus on providing continuous operation even in the face of hardware or software failures. They ensure that critical services remain accessible by automatically switching to a redundant node when a failure occurs. This type of clustering is widely used in mission-critical applications where downtime can have severe consequences.

Load-Balancing Clusters Load-balancing clusters distribute incoming workload evenly across multiple nodes, optimizing resource utilization and preventing bottlenecks. By intelligently allocating tasks, these clusters improve performance and handle traffic spikes gracefully. Load-balancing clusters are commonly employed in web servers, where high traffic volumes need to be handled efficiently.

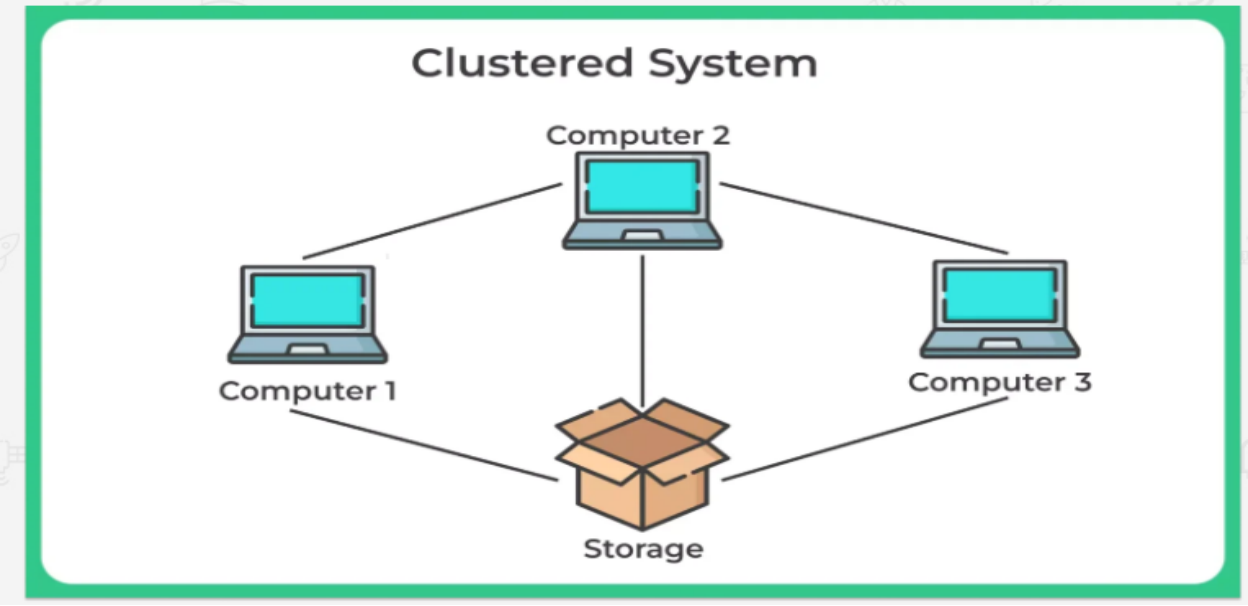
Failover Clusters Failover clusters are designed to provide seamless failover in case of node failures. When a node becomes unavailable, another standby node takes over its responsibilities to ensure uninterrupted operation. Failover clusters are commonly used in database systems and critical applications that require uninterrupted service.

Components of a Clustered System To understand how clustered systems work, it is essential to familiarize ourselves with their key components: Cluster Nodes Cluster nodes are individual computers or servers that make up the cluster. These nodes work in harmony, communicating and collaborating to achieve common goals.

Cluster Nodes Cluster nodes are individual computers or servers that make up the cluster. These nodes work in harmony, communicating and collaborating to achieve common goals.

InterconnectInterconnect refers to the communication infrastructure that allows cluster nodes to exchange data and coordinate their activities. High-speed networks, such as Ethernet or InfiniBand, are often used as interconnects.

Cluster Software Cluster software provides the necessary tools and services to manage and control the cluster. It facilitates resource sharing, load balancing, failover, and other crucial functionalities.



Examples of Clustered Systems

Several well-known clustered systems are widely used across various industries. Let's explore a few notable examples:

- **Microsoft Windows Server Failover Clustering** Microsoft Windows Server Failover Clustering enables high availability and automatic failover for Windows-based applications and services. It ensures that critical workloads remain accessible, even if individual servers or services experience failures.
- **Linux-HA** Linux-HA, or Linux High Availability, provides a framework for building highly available Linux clusters. It offers various tools and resources to configure and manage failover and load-balancing clusters.
- **Apache Hadoop** Apache Hadoop is an open-source framework used for processing and analyzing large datasets across a cluster of computers. It provides fault tolerance and scalability, making it suitable for big data processing and analytics.

How Clustered Systems Work

Clustered systems employ specialized algorithms and protocols to distribute workload, monitor node health, and ensure efficient task execution. These systems leverage the power of parallel processing to handle complex tasks by dividing them into smaller subtasks that can be executed concurrently across multiple nodes.

➢ Real Time Operating System in OS

Understanding the Basics of Operating Systems

An operating system acts as an intermediary between the hardware and software of a computer system, managing tasks, memory, storage, and input/output operations.

It provides a layer of abstraction that enables applications to run efficiently on various hardware platforms.Unlike general-purpose operating systems, such as Windows or Linux, which prioritize fairness and resource sharing, RTOS prioritizes responsiveness and deterministic behavior.

Types of Real-Time Systems

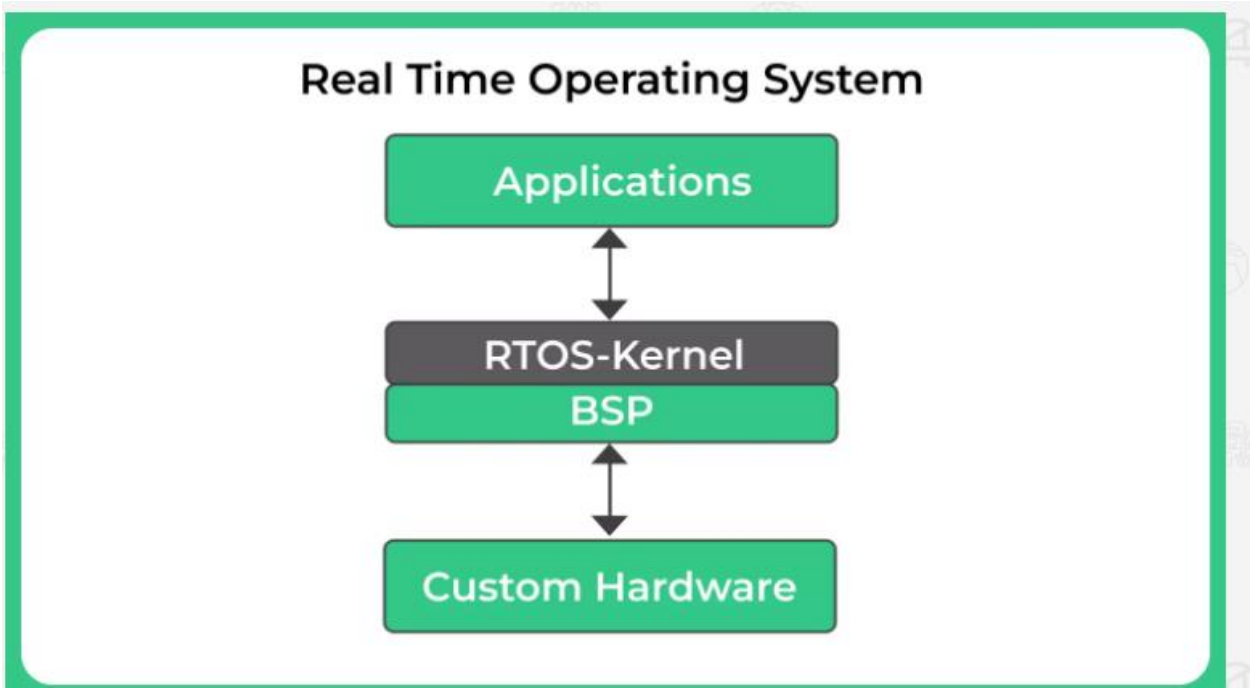
- **Hard Real-Time Systems**: In hard real-time systems, missing a deadline can have catastrophic consequences. These systems are used in critical applications such as aerospace, medical devices, and industrial control systems, where timing accuracy is of utmost importance.
- **Soft Real-Time Systems**: Soft real-time systems have more relaxed timing requirements. While missing a deadline in a soft real-time system may not result in catastrophic failure, it can lead to degraded performance or reduced system efficiency. Multimedia streaming, video games, and interactive applications often fall into this category.
- **Firm Real-Time Operating System:** The firm real-time operating system operates within time constraints, for example, Visual inspection in industrial automation, video conferencing, etc. The time limits in these systems are not stringent, but if these deadlines are missed, it is highly likely that some undesired results may occur.

What Sets Real-Time Operating Systems Apart?Real-Time Operating Systems differ from general-purpose operating systems due to their ability to execute time-critical tasks predictably and reliably. They are primarily classified into two categories: hard real-time systems and soft real-time systems.

Applications of RTOS in Embedded Systems

RTOS facilitates real-time control and coordination in various embedded system applications, including:

- Robotics and automation systems
- Medical monitoring and diagnostic devices
- Aerospace and avionics systems
- Automotive systems and driver assistance
- Industrial control systems
- Home automation and smart devices



Benefits of Using RTOS in Embedded Systems

Using an RTOS in embedded systems offers several advantages, including:

- Deterministic and predictable behavior
- Efficient resource utilization
- Real-time response to external events
- Task isolation and protection
- Simplified development and debugging
- Scalability for varying system complexities

Real-Time Operating Systems in the Internet of Things (IoT) The Internet of Things (IoT) has witnessed a significant rise in recent years, with numerous devices interconnected and communicating with each other. RTOS plays a crucial role in IoT devices, ensuring real-time response and efficient management.

RTOS in IoT Applications

RTOS enables real-time monitoring, control, and coordination of IoT devices. It ensures timely data processing, event handling, and communication, making it essential for various IoT applications such as:

- Smart home automation

- Industrial IoT (IIoT)
- Wearable devices
- Environmental monitoring
- Healthcare and telemedicine devices

Benefits and Challenges of Using RTOS in IoT Devices

Using RTOS in IoT devices offers benefits like deterministic behavior, low latency, and efficient resource utilization. However, it also presents challenges in terms of security, power optimization, and managing diverse device architectures. Striking the right balance between real-time requirements and IoT constraints is crucial.

CPU Scheduling

- **What is a process? in OS**

Understanding the Process States

A **process** can exist in different states during its lifetime. The common process states include:

- **New**: This is the initial state when a process is being created.
- **Ready**: The process is loaded into main memory and waiting to be assigned to a processor.
- **Running**: The process is being executed by a processor.
- **Blocked**: The process is unable to execute further until a certain event occurs, such as waiting for user input or completion of I/O operations.
- **Terminated**: The process has completed its execution or has been forcefully terminated.

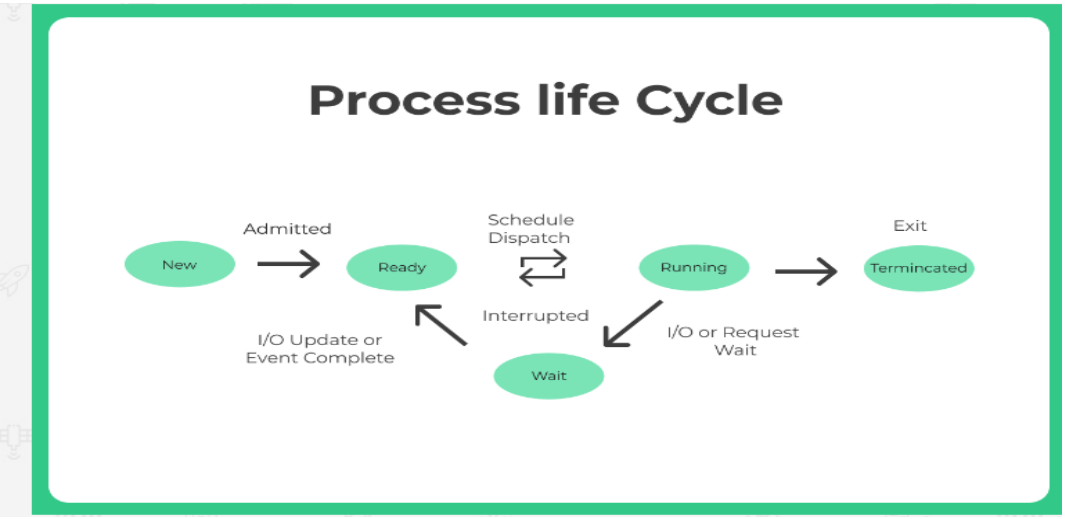
Process Scheduling

Process scheduling is a vital aspect of operating system design. It determines the order in which processes are executed by the processor.

Different scheduling algorithms, such as First-Come, First-Served (FCFS), Shortest Job Next (SJN), Round Robin (RR), and Priority Scheduling, are used to allocate CPU time to processes efficiently. These algorithms aim to optimize resource utilization and minimize process waiting times.

Process Creation

Processes are created in various ways, such as when a user initiates a program or when the operating system launches system processes during startup. The process creation involves allocating necessary resources, setting up the execution environment, and initializing the process control block. Once created, the process enters the new state and awaits execution.



Process Termination

Process termination occurs when a process finishes executing its task or when it is explicitly terminated by the operating system. Upon termination, the operating system releases the allocated resources, updates the process's state to "terminated," and removes its PCB from the system.

Interprocess Communication

In modern operating systems, processes often need to communicate and share information with each other. Interprocess communication (IPC) mechanisms facilitate this exchange of data between processes. Common IPC methods include shared memory, message passing, pipes, and sockets. These mechanisms enable efficient coordination and collaboration among different processes.

States of Processes

A process can be in the following states –

1. New
2. Ready
3. Run
4. Wait
5. Running
6. Terminated

Apart from this a process in modern system can also be stated in the following two states –

1. Suspended Ready
2. Suspended Block

We will discuss these in detail below

Real life Scenario –

Let us consider we write a **program** to play a sound, when we execute this program this program creates a **process** for operating system. The operating system now queues up the command to implement and music is played. Simple !!!

Well, its not that simple even for a process to run in a modern OS there are many other stages in the life-cycle of a process, let's discuss those in details.

Process Life Cycle

Process life cycle consists of the following stages –

1. **New –** Whenever, a fresh process is created it gets available in the new state, it can collect relevant resources like data, I/O access etc in the mean time
2. **Ready –** It gets assigned to ready stage where it waits to be assigned to a processor. This is a sort of a queue for all processes which are waiting for processor, such queue is called the **Ready Queue**.
3. **Running –** The process is being executed by processor it gets into running stage.
4. **Waiting –** Sometimes, cases happen when a process has to accept an additional input from user or maybe a higher priority process needs processor, in such cases the process goes to wait stage or waiting queue where all the processes are waiting for processor and complete their execution.
5. **Terminated –** When process has completed executing its all instructions or when it's ended by the user directly it goes to terminated stage.

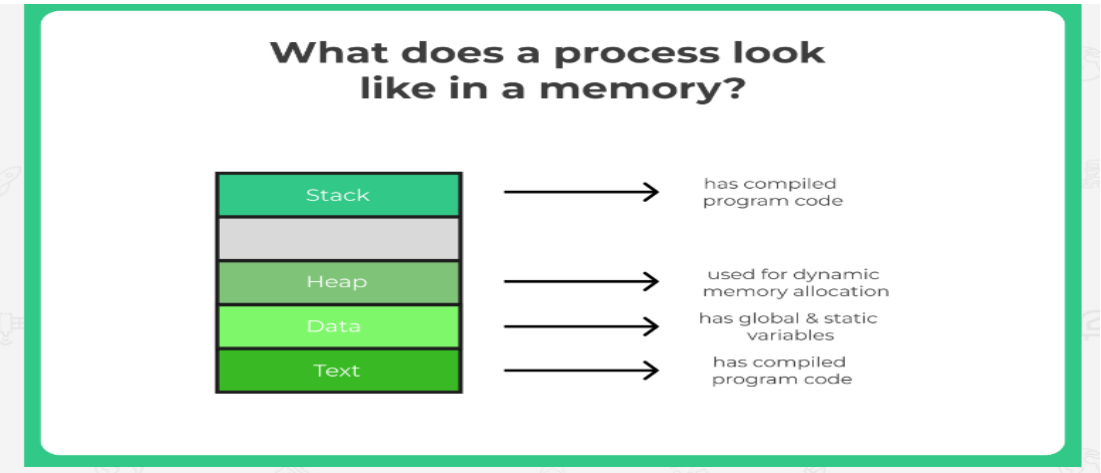
Apart from this new age system also classify –

1. **Suspended Ready –** If ready queue is full and has no space for new process then entry level process is in suspended Ready State
2. **Suspended Block –** If the waiting queue is full

Process Elements

Every process has a process memory, which is generally divided into 4, this is done to make the most efficient functioning as possible –

1. **Stack –** This houses all the temporary data and local variables like function parameters and addresses.
2. **Heap –** When the process is in its run time, for dynamic memory allocation, for operations such as new(), delete(), malloc(), etc heap is used for fastest access at the run time.
3. **Text –** Contains value of Program Counter and the contents of the processor's registers which just the most low level instructions of the compiled program
4. **Data –** Has static and global variables.



Process Control Block (PCB)

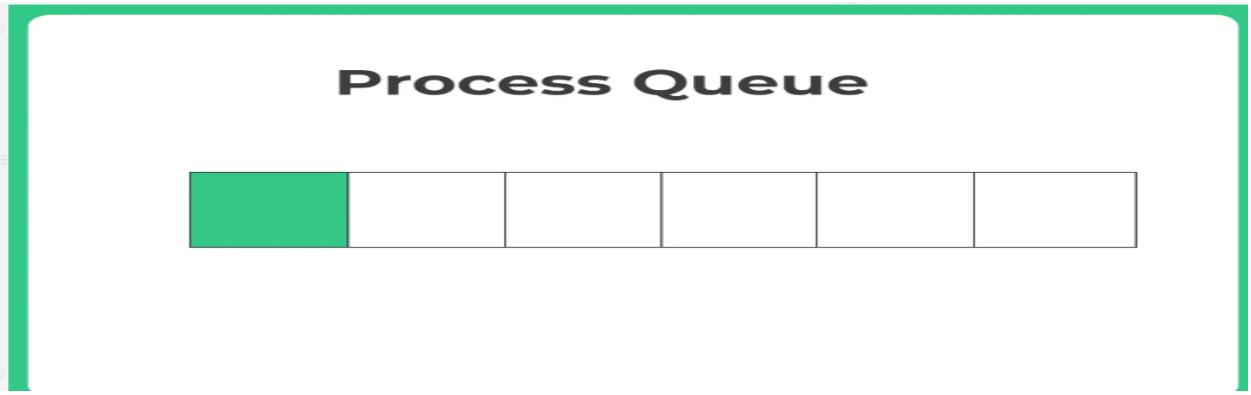
- The **Process Control Block (PCB)** is a data structure used by the operating system to manage and keep track of processes.
- It contains important information about a process, including its current state, program counter, CPU registers, memory allocation details, and more.
- The PCB allows the operating system to effectively manage and switch between processes, ensuring fair allocation of resources.

Process Life Cycle in Operating System OS

A process can be in any of the following states –

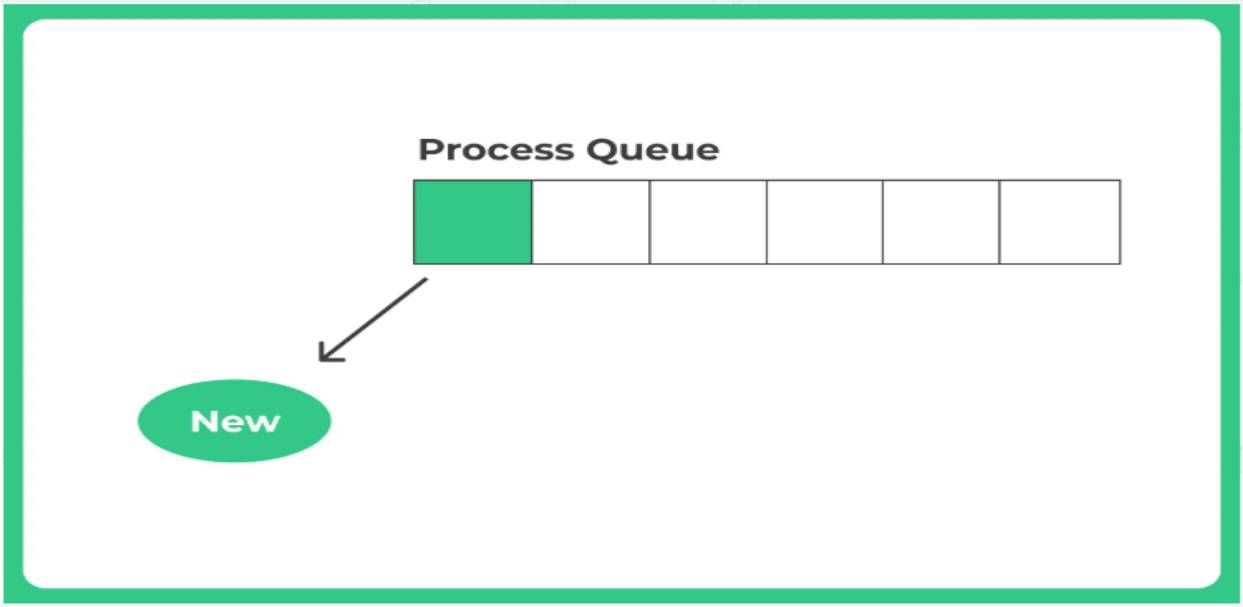
- New state
- Ready state
- Running state
- Waiting state
- Terminated

Imagine a unit process that executes a simple addition operation and prints it.



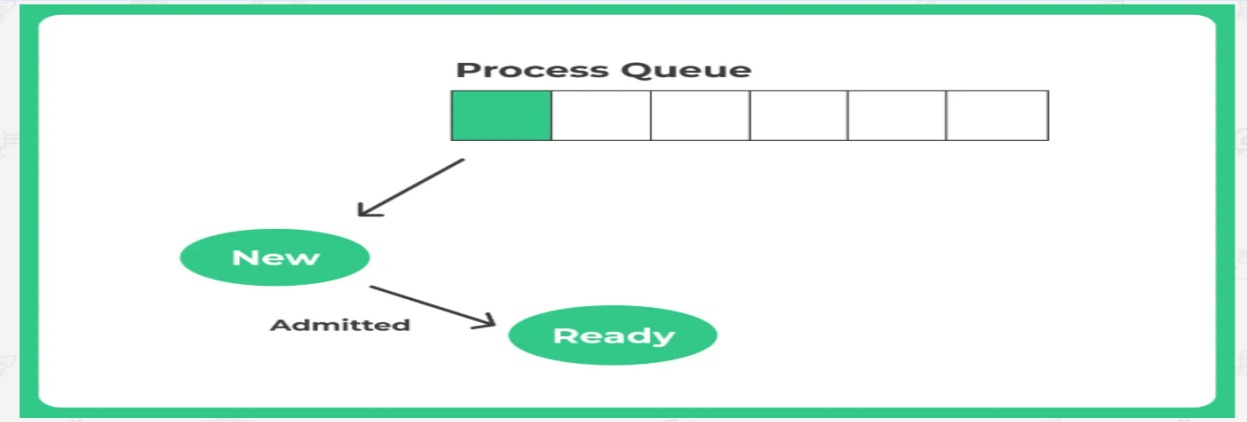
New State

- Process it submitted to the process queue, it in turns acknowledges submission.
- Once submission is acknowledged, the process is given **new status.**



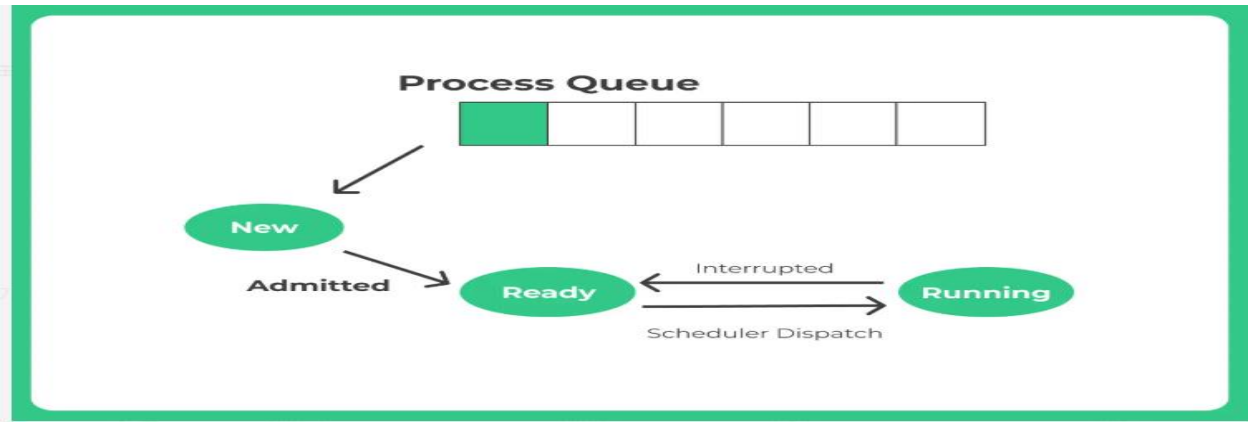
Ready State

- It then goes to **Ready State**, at this moment the process is waiting to be assigned a processor by the OS



Running State

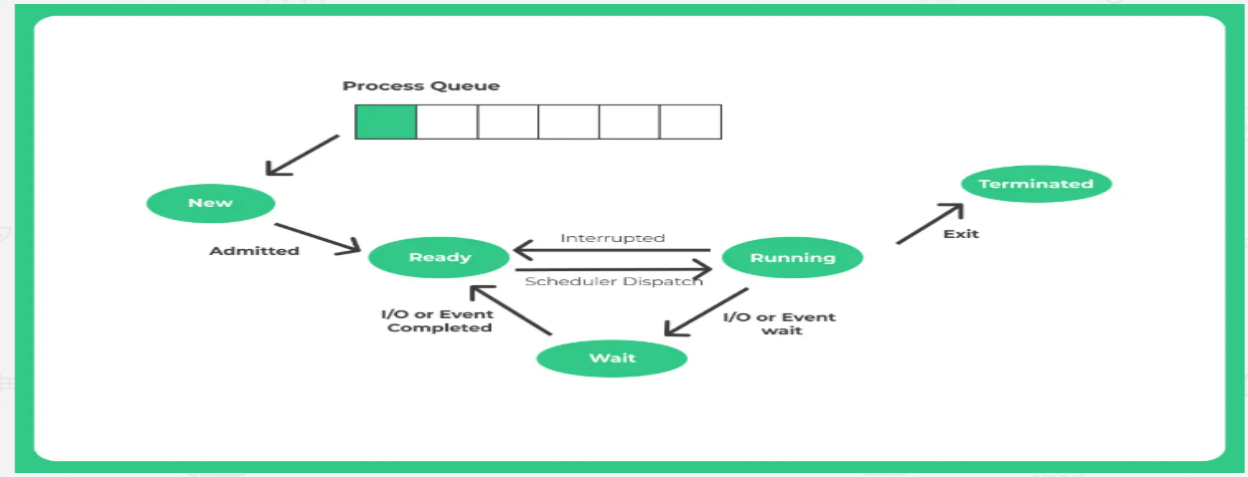
- Once the Processor is assigned, the process is being executed and turns in **Running State.**



Wait and Termination State

- Now the process can follow the following transitions –
  - The process may have all resources it needs and may get directly executed and goes to **Termination State.**
  - Process may need to go to **waiting state** any of the following
    - Access to Input/Output device (Asking user the values that are required to be added) via console
    - Process maybe intentionally interrupted by OS, as a higher priority operation maybe required, to be completed first

        - A resource or memory access that maybe locked by another process, so current process goes to **waiting state and waits** for the resource to get free.
    - Once requirements are completed i.e. either it gets back the priority to executed or requested locked resources are available to use, the process will go to **running state** again where, it may directly go to termination state or may be required to wait again for a possible required input/resource/priority interrupt.
- Termination

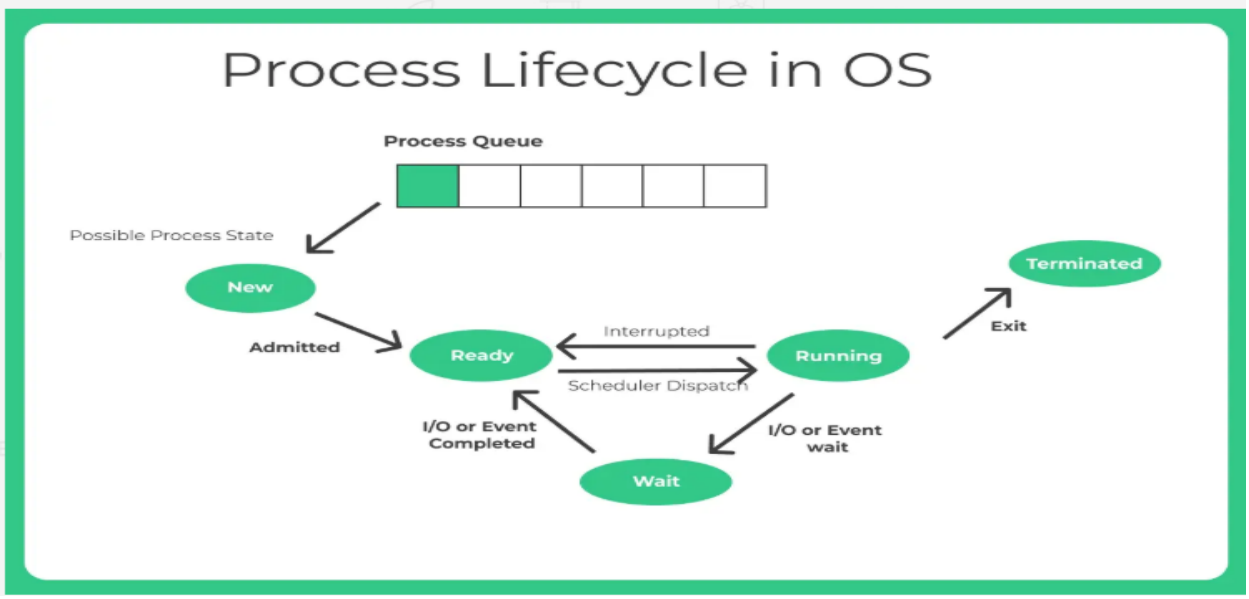


Summary

1. **New –** New Process Created
2. **Ready –** Process Ready for Processor/computing power allocation
3. **Running –** Process getting executing
4. **Wait –** Process waiting for signal
5. **Terminated –** Process execution completed

Apart from the above some new systems also propose 2 more states of process which are –

1. **Suspended Ready –** There maybe no possibility to add a new process in the queue. In such cases its can be said to be suspended ready state.
2. **Suspended Block –** If the waiting queue is full

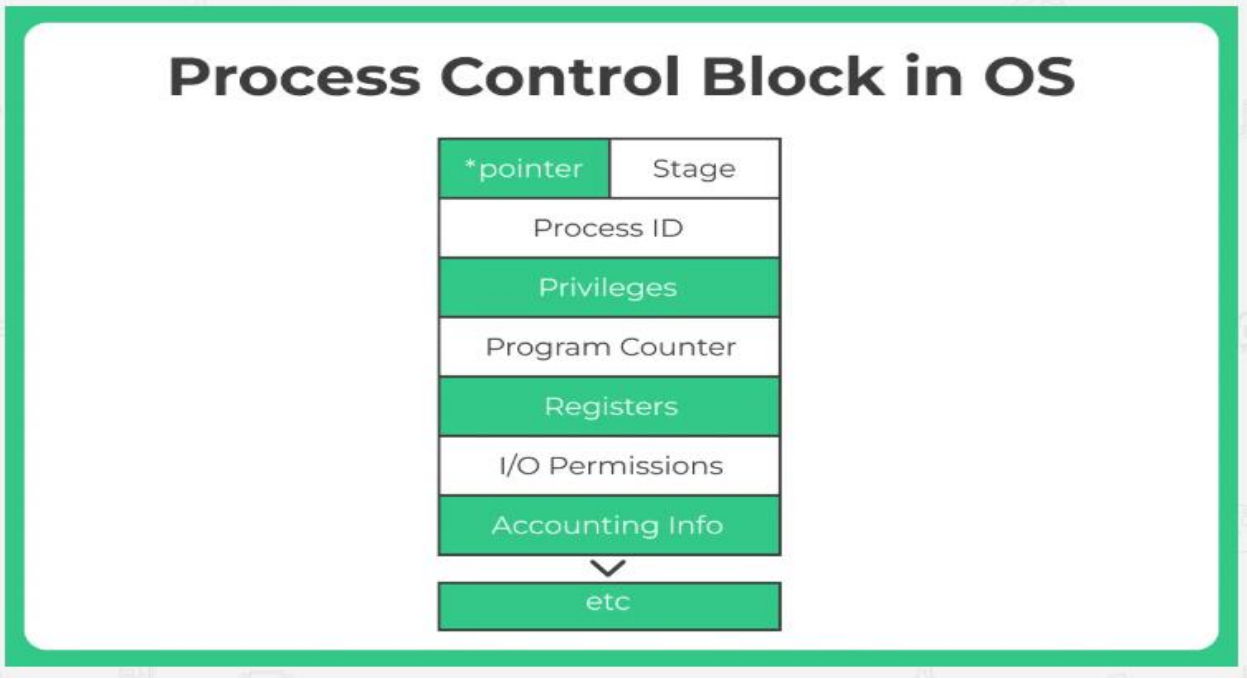


➢ Process Control Block in Operating System

What is a process control block or PCB?

PCB or Process control Block is a data structure which has all the information that is needed by a schedular to schedule a particular process, this data structure rests in the operating system Kernel.

Structure of Process Control Block



Process ID or PID

This basically is an unique integer ID for a particular process being carried out. The PCB keeps the track of this and other relevant information related to any process.

The following are kept track of by the PCB –

1. **Process ID or PID –** Unique Integer Id for each process in any stage of execution.
2. **Process Stage –** The state any process currently is in, like Ready, wait, exit etc
3. **Process Privileges –** The special access to different resources to the memory or devices the process has.
4. **Pointer** – Pointer location to the parent process.
5. **Program Counter –** It will always have the address of the next instruction in line of the processes
6. **CPU Registers –** Before the execution of the program the CPU registered where the process needs to be stored at.
7. **Scheduling Information –** There are different scheduling algorithms for a process based on which they will be selected in priority. This section contains all the information about the scheduling.
8. **Memory Management Information –** The operating system will use a lot of memory and it needs to know information like – page table, memory limits, Segment table to execute different programs MIM has all the information about this.
9. **Accounting Information –** As the name suggest it will contain all the information about the time process took, Execution ID, Limits etc.
10. **I/O Status** – The list of all the information of I/O the process can use.

The PCB architecture is completely different for different OSes. The above is the most generalised architecture.

➢ Process Scheduling in Operating System

What is Process scheduling in Operating System

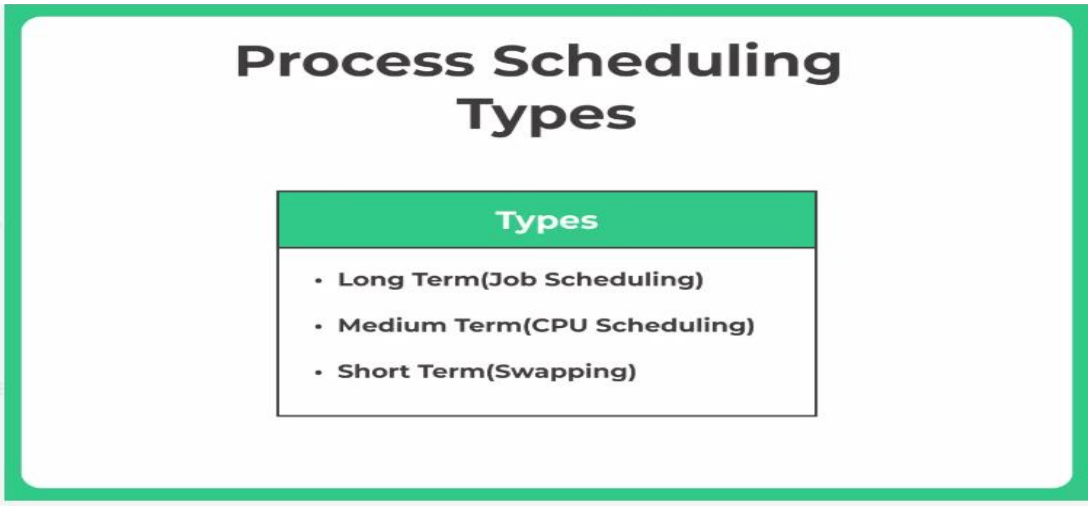
The decision to move different parallel processes competing with one another for execution, to different states like Ready to running or running to exit state based on a certain decision strategy is known as Process Scheduling in Operating System.

Handled by – Process Manager

Why CPU Scheduling?
One or more processes may be in the running state at the same time and they all share the CPU time using multiplexing.

Focus for Intelligent Scheduling is to –

1. Keep CPU busy as much as possible if processes are waiting for execution
2. Decrease the time of execution of critical processes
3. Intelligently prioritise processes for execution
4. Keep the response time minimum.



Types of Process Scheduling
We divided Process Scheduling into following two

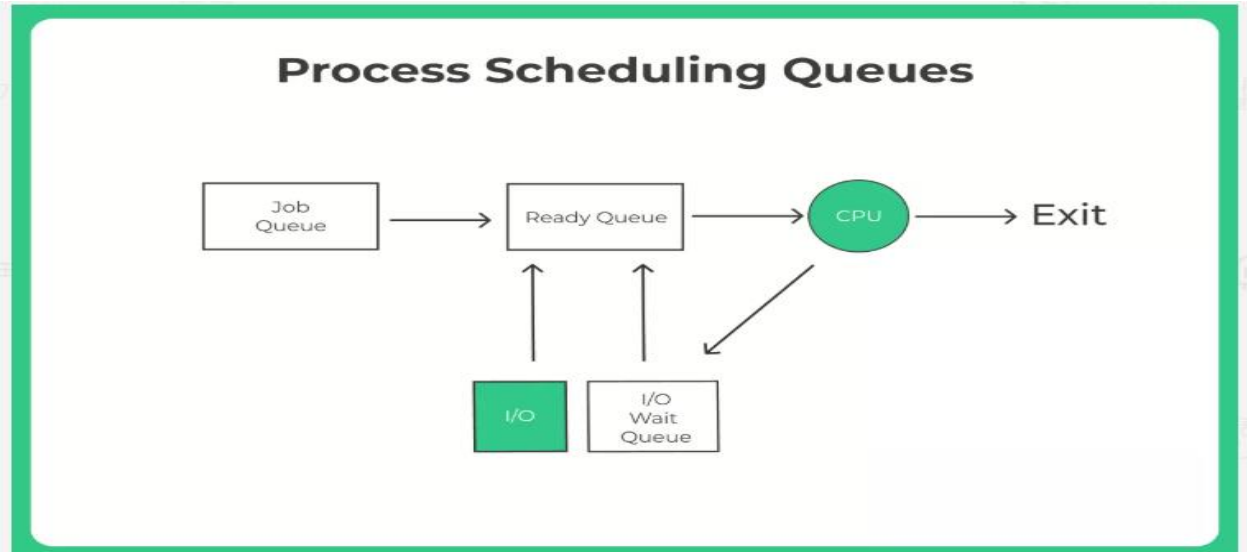
- **Preemptive Scheduling –**The scheduling in which a running process can be interrupted if a high priority process enters the queue and is allocated to the CPU is called preemptive scheduling. In this case, the current process switches from the running queue to ready queue and the high priority process utilizes the CPU cycle.
- **Non Preemptive Scheduling –** The scheduling in which a running process cannot be interrupted by any other process is called non-preemptive scheduling. Any other process which enters the queue has to wait until the current process finishes its CPU cycle

Meaning of Prempt – take action in order to prevent (an anticipated event) happening; forestall.

Process Scheduling Queues

There are the following types of queues –

1. Job Queue – Whenever any process enters the system its there in job Queue
2. Ready Queue – Processes in the ready queue waiting for run time are in the ready queue.
3. Device Queue – Some processes may be waiting some I/O operation, such processes are in the device queue.



Different Types of Schedulers

There are three type of schedulers categorically –

1. Long-term Scheduler

2. Short-term Scheduler
3. Medium-term Scheduler

Long Term Scheduler

Also knowns as **Job Scheduler**. It selects the process that are to be placed in ready queue. The long term scheduler basically decides the priority in which processes must be placed in main memory. Processes of long term scheduler are placed in the ready state because in this state the process is ready to execute waiting for calls of execution from CPU which takes time that's why this is known as long term scheduler.

Medium-Term Scheduler

It places the blocked and suspended processes in the secondary memory of a computer system. The task of moving from main memory to secondary memory is called **swapping out**. The task of moving back a swapped out process from secondary memory to main memory is known as **swapping in**. The swapping of processes is performed to ensure the best utilisation of main memory.

Short-Term Scheduler

Also knowns as **CPU Scheduler**. It decides the priority in which processes is in the ready queue are allocated the central processing unit (CPU) time for their execution. The short term scheduler is also referred as central processing unit (CPU) scheduler.

Context Switch

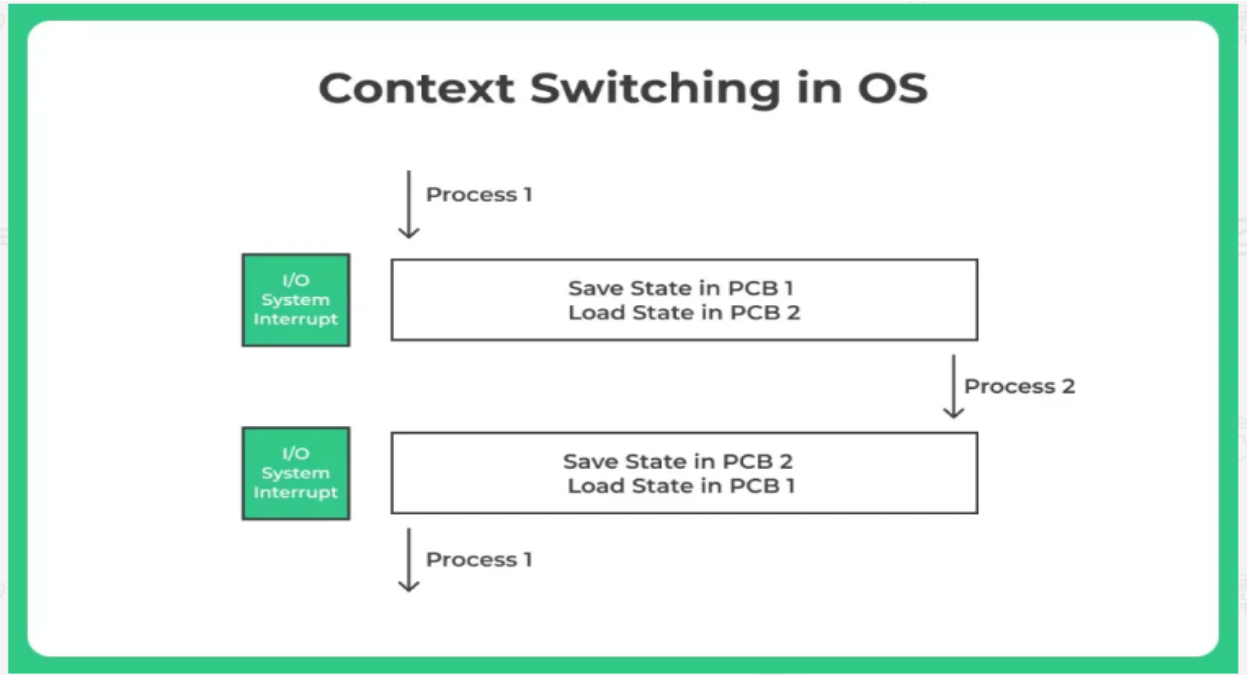
A context switch is a procedure that a computer's CPU (central processing unit) follows to change from one task (or process) to another while ensuring that the tasks do not conflict. Effective context switching is critical if a computer is to provide user-friendly multitasking.

Context Switching in Operating System

Context Switch in OS

Context Switching is a cost and time saving measure performed by the CPU that is handing task 1 and has to stop executing this task to priority execute another task 2. To do this effectively, the system stores the initial task in its processed form so that when this task is resumed, it can be loaded and resumed from the same progress point as earlier.

This allows the computer to multitask and multiple processes can share the same CPU. This is also done to make sure that tasks do conflict one another, as they may be holding a same resource for execution like memory etc.



Steps Leading to Context Switching

In the above diagram the following happens –

1. Process 1 running in the system
2. Interrupt or system call appears
3. Causes Process 1 to be saved into PCB 1 and switched out
4. Process 2 is loaded from PCB 2 and executed
5. Again interrupt or system call appears
6. Process 2 is saved into PCB 2 and process 1 loaded from PCB 1
7. Process 1 is executed

Triggers for Context Switching

1. **Multitasking**
2. **Interrupt Handling**
3. **User and Kernel mode Switch –** System may switch from user mode to the kernel mode, you can think of this as windows 10 running and system is switched to boot mode.

CPU Scheduling in Operating System

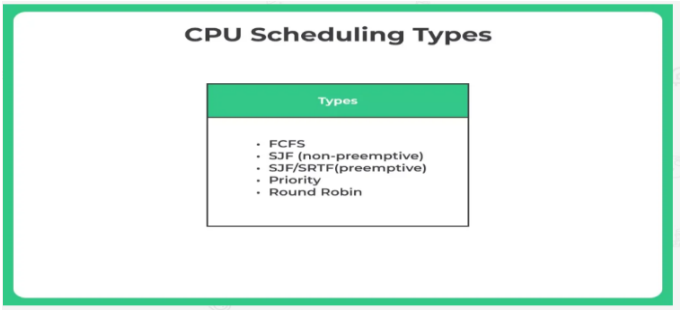
What is CPU Scheduling in Operating System

CPU scheduling is the system used to schedule processes that wants to use CPU time. It allows one process (say P1) to use the CPU time by possibly putting other process (say P2) on hold or in waiting queue since P2 may be waiting for an additional resource like input or locked file etc.

We use CPU scheduling to make efficient use of CPU to increase the CPU utility and to reduce optimize time usage.

If the CPU gets idle, it may select a process from the ready queue using various algorithms to intelligently select the process that needs to be executed next.

This is done by **short term scheduler i.e. CPU scheduler**.



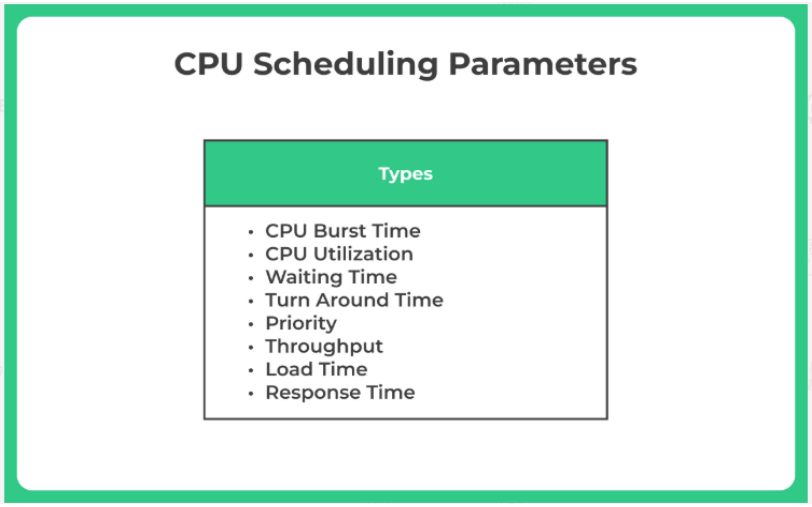
Why we need CPU Scheduling?
A process needs –

- Data/File Resources
- I/O time
- CPU time

While in a single process system a lot of CPU time may be wasted, since the current process may have to wait for input or locked resource etc. This causes CPU time to be wasted. To over come this we use multi-programming systems, which supports process switch if another process in running state is waiting for any resource or I/O.

In such cases, any process from the ready queue is selected based on a pre-defined logic, this is done by short term scheduler.

CPU Scheduling Parameters
Don’t worry if you don’t understand the definitions now, you will learn more about them when we discuss different types of Scheduling algorithms in detail.



- **CPU Burst Time –** In simple terms, the duration for which a process gets control of the CPU is the CPU burst time, and the concept of gaining control of the CPU is the CPU burst. Similarly, an I/O burst is the concept of performing I/O operations.
- **CPU Utilization –** CPU utilization can be defined as the the percentage of time CPU was handling process execution to total time

Formulae –

CPU Utilization = (Total time – Total idle time)/(Total Time)

A computer with 75% CPU utilization is better than 50%, since the CPU time was better utilized to handle process execution in 75% and lesser time was wasted in idle time.

- **Waiting Time** is the total amount of time spent in the ready queue to gain the access of the CPU for execution.
- **Turn Around Time –** From the time the process is submitted to the time the process is completed, is defined as Turn Around Time.
- **Throughput –** In a unit time the number of processes that can be completed is called the throughput.
- **Load Time –** It is defined as the average number of process that are pending in the Ready Queue and waiting for execution time.

- **Response Time –** can be defined as the time interval between when the process was submitted and when the first response is given to the process.

Different types of Scheduling Algorithms –

- First come first serve (FCFS)
- Shortest Job First (SJF) – non-preemptive
- Shortest Job First (SJF) – preemptive
- Priority Scheduling
- Round Robin

Listed below are some other algorithms:

1. Highest Response Ratio Next (HRRN)
2. Multilevel Queue Scheduling:
3. Multi level Feedback Queue Scheduling

Preemptive vs Non Preemptive Scheduling

Non Preemptive Scheduling

In this type of scheduling, if a process enters the CPU and gets processing time. The process will keep on executing until it has terminated or has to forcibly go to waiting state as it needs a resource that it locked by another parallel process.

Example – SJF i.e. Shortest Job First (Non Preemptive)

Preemptive Scheduling

In this type of scheduling, if a process enters the CPU and gets processing time. It can get switched by another process that have higher priority.

Priority may be forced i.e. may be assigned to each process or time based dynamic priority as in the case of SRTF algorithm.

Example – SRTF i.e. Shortest Remaining time First (Also known as SJF – Preemptive)

Note – This information is given wrong on Geeks4Geeks they have mentioned SRTF non preemptive.

Note – It's okay to not to understand the above completely as of now. We suggest going through all the different scheduling algorithms and solve first and then come back here to understand it again, then it will be easy.

FCFS Scheduling Algorithm in Operating System

What is First Come First Served Scheduling Algorithm in Operating System?

There are various scheduling algorithm in Operating System but today we will learn about FCFS(First Come First Served) scheduling algorithm in operating system. Here we learn from basic knowledge to and various effects in it so you understand the topic as efficiently as possible.

First Come First Served Scheduling Algorithm in Operating System

This is the most basic algorithm in Operating System, as the name suggests, first come first serve (FCFS) the process which enters the system first gets the CPU time first.

Consider the following scenario –

Lets say you're waiting at a railway station's ticket counter at the front of the queue the person who arrived first will be given chance to buy the ticket, and you will have to wait for your turn until all the people in front of you in the queue or those who have arrived earlier have bought tickets.

These are the traits of FCFS algorithm –

- It is parallel to how FIFO in stack is
- Process which arrives firsts gets CPU time first
- It is a non preemptive type of algorithm

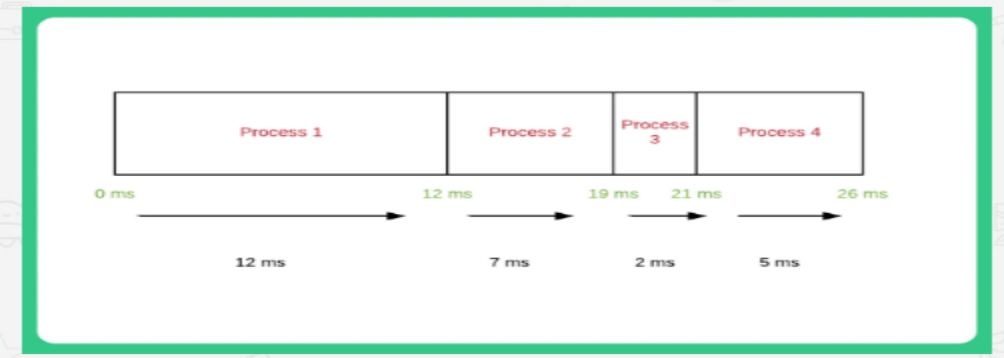
Let us try to understand FCFS Scheduling with the help of an example –

Note – If order and arrival time is not given then consider the order as is and arrival time to be 0 for all the processes.

[table id=241 /]

1. Turn Around Time = Completion Time – Arrival Time
2. Waiting Time = Turnaround time – Burst Time

Lets make a gantt Chart for this first.



Waiting time –

1. P1 waiting time – 0ms
2. P2 waiting time – 12ms
3. P3 waiting time – 19ms
4. P4 waiting time – 21ms

Thus, average waiting time – (0 + 12 + 19 + 21)/4 = 13ms

Turn around time – Time when process entered the system and when its out of the system

1. P1 turn around time – (12 – 0) ms
2. P2 turn around time – (19 – 0) ms
3. P3 turn around time – (21 – 0) ms
4. P4 turn around time – (26 – 0) ms

Average turn around time (12 + 19 + 21 + 26)/4 = 19.5 ms

Disadvantages of FCFS Algorithm

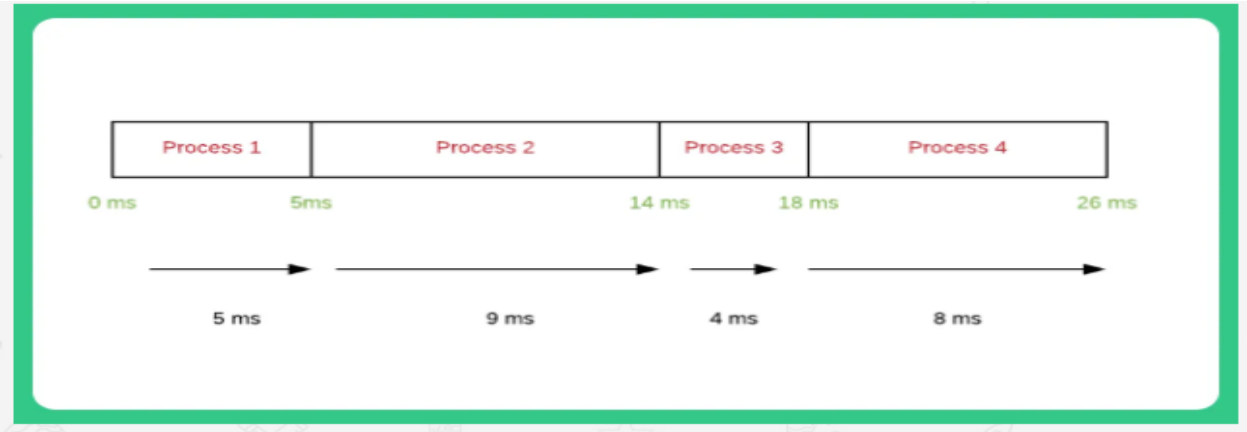
- Since, its a non preemptive algorithm thus, there is no intelligence applied and no priority can be given to processes.
- If critical system process arrives it may have to wait for a process like calculator.
- Waiting time may get too high, along with the turn around time.
- Causes convoy effect

What is convoy effect?

FCFS may suffer from the **convoy effect** if the burst time of the first job is the highest among all. As in the real life, if a convoy is passing through the road then the other persons may get blocked until it passes completely. This can be simulated in the Operating System also.

If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time. This is called **convoy effect** or **starvation**.

In this case the Gantt Chart looks like



The 2 seconds in the start can be avoided as the system only starts at 2 ms when the first process arrives.

Here have close watch on the arrival time for each process –

arrival time –

1. P1 arrival time – 0 ms
2. P2 arrival time – 4 ms
3. P3 arrival time – 6 ms
4. P4 arrival time – 13 ms

Waiting time –

1. P1 waiting time – 0 ms
2. P2 waiting time – (5 – 4) = 1 ms
3. P3 waiting time – (14 – 6) = 8 ms
4. P4 waiting time – (18 – 13) = 5 ms

Thus, average waiting time – (0 + 1 + 8 + 5)/4 = 3.50ms

Turn Around time –

1. P1 turn around time – 5 ms
2. P2 turn around time – (14 – 4) = 10 ms

3. P3 turn around time – (18 – 6) = 12 ms
4. P4 turn around time – (26 – 13) = 13 ms

Thus, average Turn Around time – (5 + 10 + 12 + 13)/4 = 12.5 ms

Shortest Job First Scheduling Non Preemptive

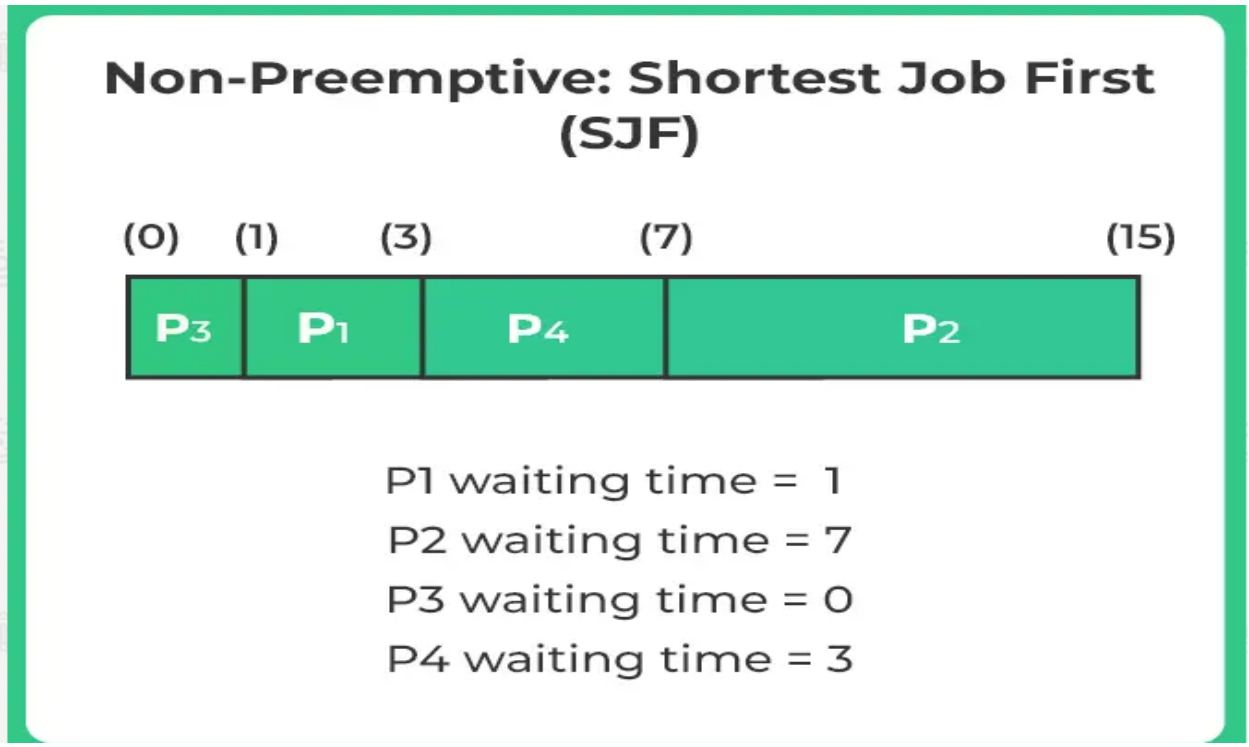
Shortest Job First (SJF) – Non Preemptive
Shortest Job First Scheduling (Non Preemptive Algorithm) in Operating System

- Shortest Job First (SJF) is a Scheduling Algorithm where the process are executed in ascending order of their burst time, that is, the process having the shortest burst time is executed first and so on.
- The processor knows burst time of each process in advance.
- It can be thought of as shortest-next-cpu-burst algorithm, as Scheduling depends on length of the next CPU burst of a process. (The duration for which a process gets control of the CPU, is the Burst time for a process.)
- SJF can be Pre-emptive or Non- preemptive. Under Non-preemptive Scheduling , once a process has been allocated to CPU, the process keeps the CPU until the process has finished its execution.

SJF Non Preemptive Example
Let us try to understand SJF Non-Preemptive Scheduling with an example –

Here, processes are assumed to be arrived at the same time.

The order in which the CPU processes the process are(Gantt Chart) –



Steps –

1. Since Burst time for P3 (Burst = 1 sec) is lowest it is executed first.
2. Then Burst time for P1 is lowest in order thus it gets executed the 2nd time.
3. Similarly P4 and then P2

Waiting time –

1. P1 waiting time = 1
2. P2 waiting time = 7
3. P3 waiting time = 0
4. P4 waiting time = 3

The average waiting time is = (1+ 7 + 0 + 3)/ 4 = 2.75

Features of SJF

- SJF is a greedy Algorithm

- It has Minimum average waiting time among all scheduling algorithms.
- Difficulty of SJF is knowing the length of next CPU request.
- It's used frequently in Long-term scheduling in Batch System as in this, the time limit is provided by the user specifying the process. We presume that user provides accurate time limit as lower accurate value means faster response.
- SJF can't be implemented in Short-term scheduling as one can't know the exact length of next CPU burst. It can be Approximated by doing an exponential average of already measured length of previous CPU burst.

Drawback

- in SJF Scheduling, a process with high burst time may suffer starvation. Starvation is the process in which a process with higer burst time is kept on waiting and waiting , but is not allocated to the CPU. It's prevented by **aging.**
- Total execution time must be known beforehand of a process.

What is aging?

- Aging is a technique which is used to reduce starvation of the processes.
- Aging takes into account the waiting time of the process in the ready queue and gradually increases the priority of the process.
- Hence, as the priority of a process gets increased this ensures that all process get completed eventually.

Shortest Job First – Preemptive Scheduling with Example (SJF)

What is SJF Preemptive Scheduling Algorithm in OS

Shortest Job First Preemptive Scheduling is also known as Shortest remaining Time(SRT) or Shortest Next Time(SNT).

1. The choice of preemptive and non preemptive arises when a new process arrives at the ready queue and a previous process is not finished and is being executed. If the next CPU burst of new process is shorter than current executing process, then in preemptive version , it will stop that process and will start executing the newly arrived process.

2. While, in non preemptive version of SJF, even if the arriving process is shorter than currently executing process, current process is not stopped . After the current process finishes , then the new process gets in the queue. This is the key difference between preemptive and preemptive version of SJF.
3. The current state of the process is saved by the context switch and the CPU is given to another process.

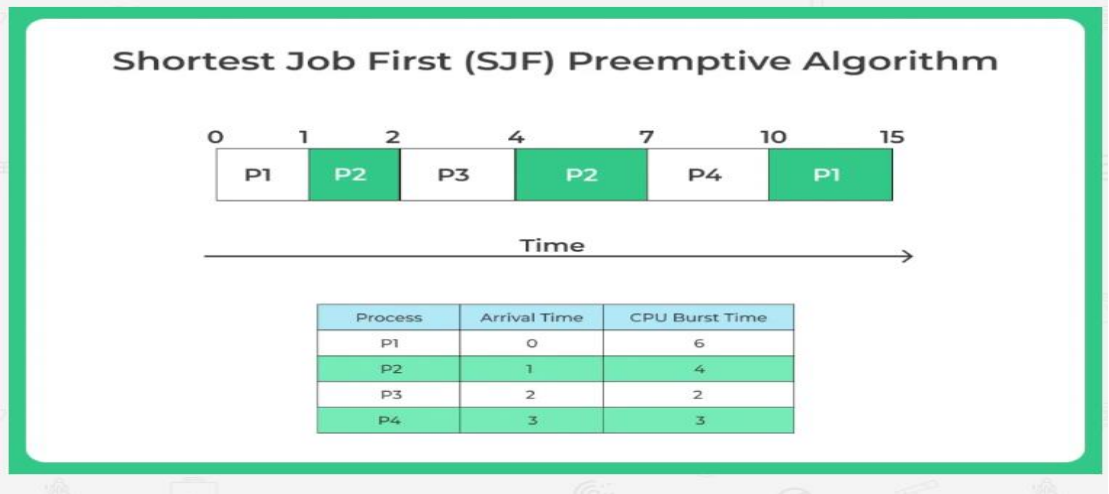
Note – If 2 processes have same execution time, then jobs are based on First Come First Serve Basis.

Shortest Job First – Preemptive Scheduling with Example (SJF)

Let's understand SJF Scheduling with the help of an example.

[table id=254 /]

The order in which the CPU processes the process are (Gantt Chart) –



1. At (t =0ms), P1 arrives. It's the only process so CPU starts executing it.
2. At (t = 1ms), P2 has arrived . At this time, P1 (remaining time) = 5 ms . P2 has 4ms , so as P2 is shorter, P1 is preempted and P2 process starts executing.
3. At (t = 2ms), P3 process has arrived. At this time, P1(remaining time) = 5ms, P2(remaining time) = 3 ms , P3 = 2ms. Since P3 is having least burst time, P3 is executed .
4. At (t = 3ms), P4 comes , At this time, P1 = 5ms, P2 = 3ms, P3 = 1ms, P4 = 3ms. Since P4 does not have short burst time, so P3 continues to execute.
5. At (t= 4ms),P3 is finished . Now, remaining tasks are P1 = 5ms, P2 = 3ms, P4 = 3ms. As ,P2 and P4 have same time, so the task which came first will be executed first. So, P2 gets executed first.
6. At (t = 7ms),P4 gets executed for 3ms.

7. At (t = 10ms), P1 gets executed till it finishes.

Waiting time = Completion time – Burst Time – Arrival Time

1. P1 waiting time = (15-6-0) = 9ms
2. P2 waiting time = (7-4-1) = 2ms
3. P3 waiting time = (4-2-2)=0ms
4. P4 waiting time = (10-3-3)= 4ms

The average waiting time is (9 +2 + 0 + 4)/4 = 15/4 = 3.75

Disadvantages

It has the same disadvantages as non-preemptive version of SJF. Here also process with larger burst time may experience process starvation, which could be prevented with aging.

Round Robin Scheduling Algorithm in Operating System

Round Robin Scheduling Algorithm in Operating System

In Round robin Scheduling Algorithm, each process is given a fixed time called quantum for execution. After the Quantum of time passes, the current running process is preempted and the next process gets executed for next quantum of time.

CPU Scheduler goes around the ready queue and allocates CPU to each process for the interval upto 1 time quantum. Ready Queue is like a First In First Out Structure, where new processes are at last of the ready queue.

Now, while a process is in that quantum executing, one of the two things will happen

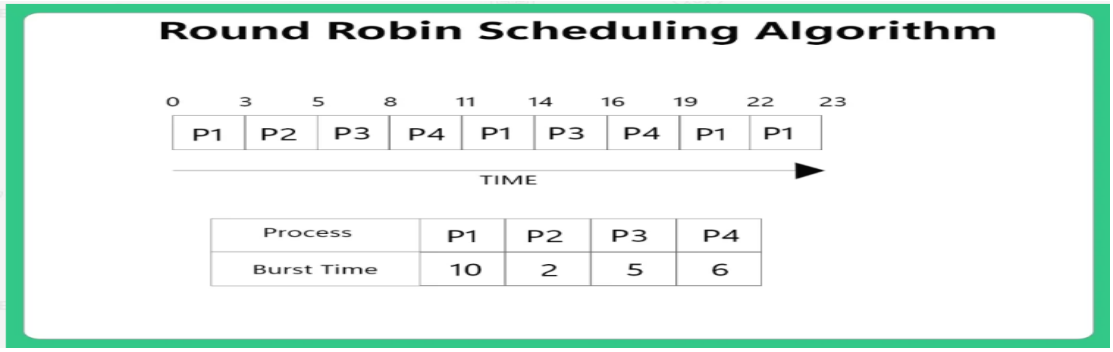
1. If the process has a CPU burst of less than quantum time, then as the process gets finished , CPU resources are released voluntarily. the next process in the ready queue occupies the CPU.
2. If the CPU burst is larger than the quantum, the timer will go off and the process is preempted, its state is saved by context switching and process is put at last of ready queue. After this, the next process in the ready queue comes in CPU quantum.

Round Robin Example

Here, we have taken quantum as 3ms. So,each process is assigned 3ms before switching to next process.

- P1 is executed for 3ms. The, P2 is executed. As, P2 is only 2ms long, So after 2 ms , next process occupies the CPU.

- P3 then is executed for 3ms. So, P3 remaining = 2ms. Then , P4 is executed for 3ms. P4 remaining time = 3ms
- P1 is again executed for 3ms. P1 remaining time = 4ms
- P3 is executed for 2ms. Then, P4 is executed for 3ms.
- Now remaining process is P1 only . It executed for 3 ms. After that, Since P2,P3 and P4 are already finished, P1 is executed again for remaining time (1ms).



Let's calculate Average Waiting time for each process –

- Average waiting time for P1 = 19- (3+3) = 13ms
- Average Waiting time for P2 = 3ms
- Average Waiting time for P3 = 14-3 = 11ms
- Average Waiting time for P4 = 16-3 = 13ms

Total average waiting time = (13 +3 +11+13)/4 = 10ms

Advantages –

- It does not cause starvation as all process get equal time of CPU.

Disadvantages –

- There is an overhead of context switching as mentioned earlier, too small of quantum time causes overhead and slower execution of process. So, time quantum must be large with respect to Context Switch time.

Priority Scheduling Algorithm

Priority Scheduling Algorithm in Operating System

In Priority Scheduling, processes are assigned Priorities and the process with the highest is executed first. Each process is assigned a priority & processes are given CPU time according to their priority.

Priority may also be of two types :

1. **Static**
2. **Dynamic**

Rules
In some systems, lower priority number means that higher priority (Example P1 with priority 1 and P2 with priority 4, in this case P1 will have higher priority). However, in some system its opposite, higher priority number means higher priority.

Students may get confused with this. Thus, Gate now specifically mentions which one is the highest and which one is the lowest. However, if this information is not given in exams like placement tests etc. Then lower priority number means higher priority.

This information is given incorrectly in Geeks4Geeks thus, most students solve the questions incorrectly.

- If 2 processes have equal priorities the those process are scheduled in First Come First Serve order.
- Priority can be defined internally or externally.
- It can be either preemptive or non preemptive algorithm.

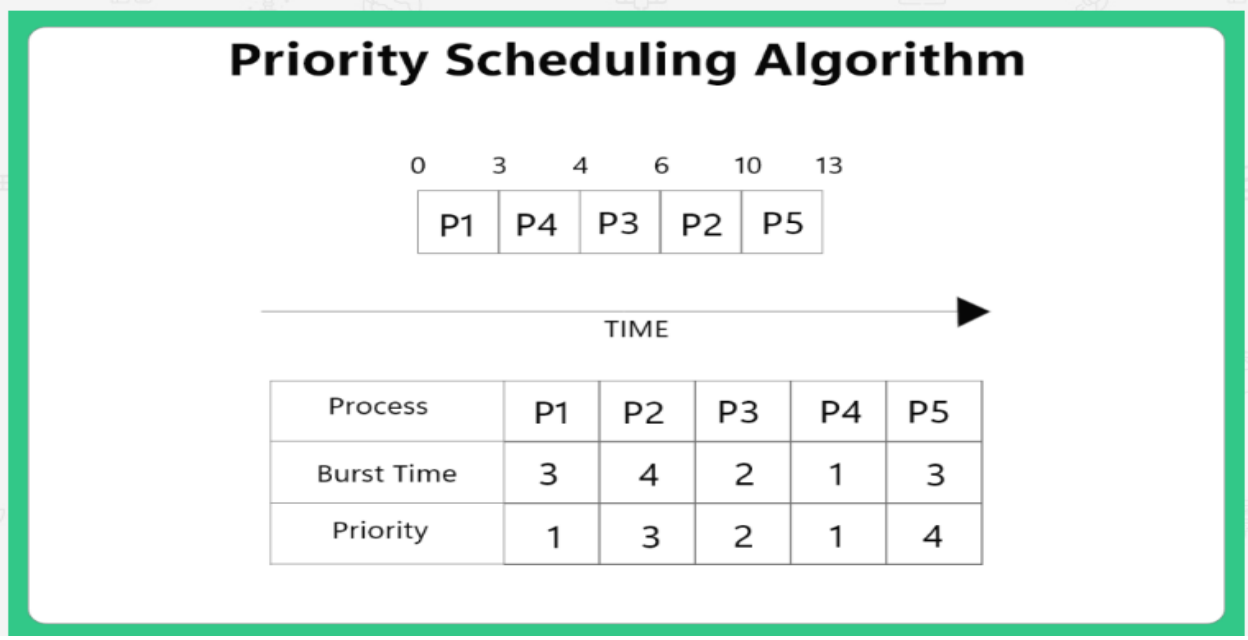
Preemptive vs Non Preemptive
First Come First Serve (FCFS) is a special type of priority scheduling where all processes are assigned equal process. Similarly, Shortest Job first (SJF) is also a special type in which the one having least CPU burst time is given a high priority.

As we know, Preemptive Priority Scheduling Algorithm are those algorithm where if a new process having higher priority arrives than the process currently executing , CPU stops the current executing process and executes the newly arrived higher priority process. Non Preemptive Priority Scheduling Algorithm is an algorithm where even if a higher priority process comes, if a process is already being executed, it will first finish the current process . Only then, next process will be executed.

Priority Scheduling Example 1 (Non-Preemptive)
Let’s try to understand Priority Scheduling with the help of an example (For Ease of understanding, for first example arrival time for all is 0) –

In our example consider the lowest value as the highest Priority and note we are considering non preemptive algorithm in nature thus no dynamic priority change will be there.

We suggest solving it on your own and not looking the solution and explanation below –



- Both P1 and P4 have the highest Priority(1). Thus we will use FCFS to settle clash and P1 will execute first.
- P4 will execute once P1 is finished executing
- P3 has priority 2 thus it will execute next.
- Then finally P2 and P5

Average Waiting Time for processes are –

Average Waiting Time = Completion Time – Burst Time – Arrival TIme

- Average Waiting Time for P1 = 3-3-0 = 0ms
- Average Waiting Time for P2 = 10-4-0 = 6ms
- Average Waiting Time for P3 = 6-2-0 = 4ms
- Average Waiting Time for P4 = 4-1-0 = 3ms
- Average Waiting Time for P5 = 13-3-0 = 10ms

Average waiting Time for all processes = (0 + 6 + 4 + 3 +10)/5 = 4.6ms

Disadvantages

Like FCFS and SJF , a process having lower priority can be indefinite blocked or starved. To prevent this, we do aging where the priority of the process is increased as it waits in the queue. So, a process which has been in a queue for a long time will reach high priority, hence it won't be starved.

Priority Scheduling Example 2 (Non-Preemptive)

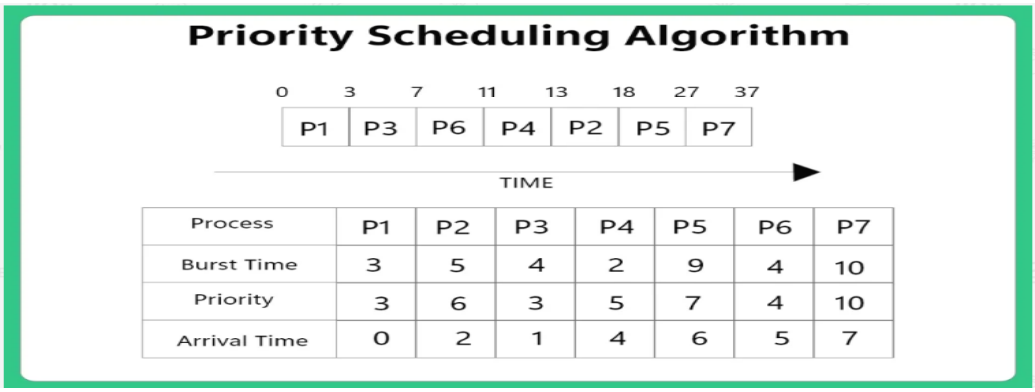
In this example we will complicate things as the arrival time for algorithms will not be same.

Note – Consider lower number to have higher priority.

Steps –

1. At time t = 0, there is only process that has arrived i.e. P1 so it will execute first for next 3 ms.
2. At time t = 0, P1 is completed and there are two more processes that have arrived, P2 and P3
    1. P2 has priority 6 and P3 has 3
    2. Thus P3 will execute for next 4 ms, i.e. till t = 7 ms.
3. At time t = 7ms all the processes have arrived.
4. Thus, P6 with priority 4 will be next
5. Similarly, P4 -> P2 -> P5 and then P7

The Gantt Chart will look like –



Process	P1	P2	P3	P4	P5	P6	P7
Burst Time	3	5	4	2	9	4	10
Priority	3	6	3	5	7	4	10
Arrival Time	0	2	1	4	6	5	7

Turn Around Time = Completion Time – Arrival Time

Waiting Time = Turn Around Time – Burst Time

- Average Waiting Time for P1 = 3-0-3 =0
- Average Waiting Time for P2= 18-2-5 =11
- Average Waiting Time for P3= 7-1-4 =2
- Average Waiting Time for P4= 13-4-2 =7
- Average Waiting Time for P5= 27-6-9 =12
- Average Waiting Time for P6= 11-5-4 =2
- Average Waiting Time for P7= 37-7-10 =20

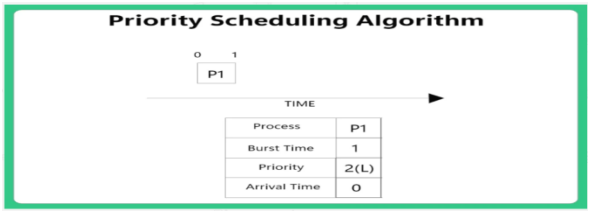
Avg Waiting Time = (0+11+2+7+12+2+20)/7 = 54/7 units = 7.714 ms

Priority Scheduling Algorithm Preemptive

Note – Consider lower number to have higher priority.

- At time t = 0

Only P1 has arrived and since its only process currently even though has lowest priority it will get processing time and has burst time of 1ms.



- At time t = 1

At time t = 1 P2 has arrived and no other process thus it will get processing time

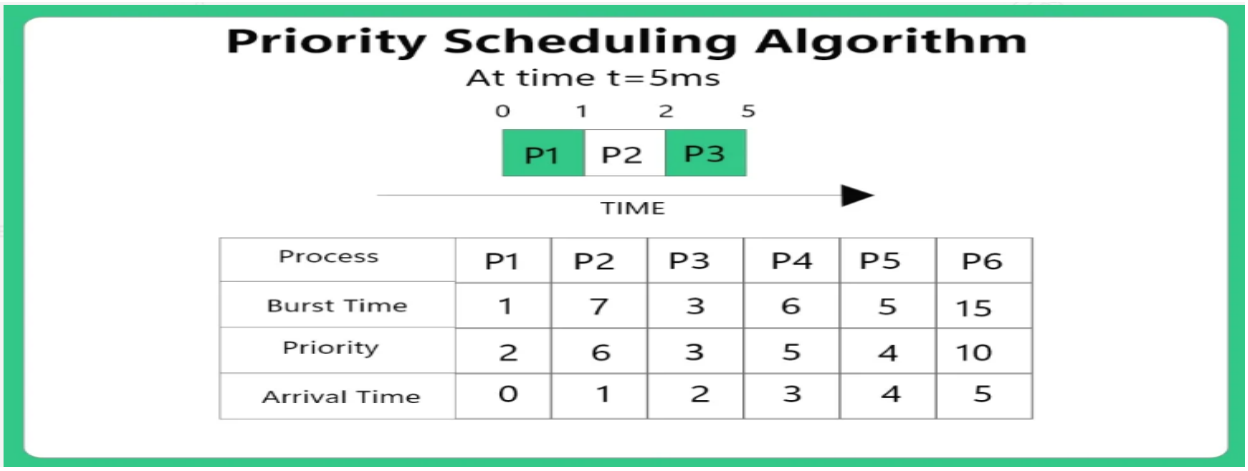
- At time t = 2

P2 is still running but, P3 has also arrived.

P3 has priority of 3 while P2 has priority of 6. Thus –

1. P2 will stop and P3 will get execution time.
2. P3 has execution time of 3ms in this time 3 more processes arrive namely – P4, P5 and P6. But all of them have lower priority than P3, thus these can't preempt the process and P3 will complete its execution time.

- Till time t = 5 P3 will run and complete its processing

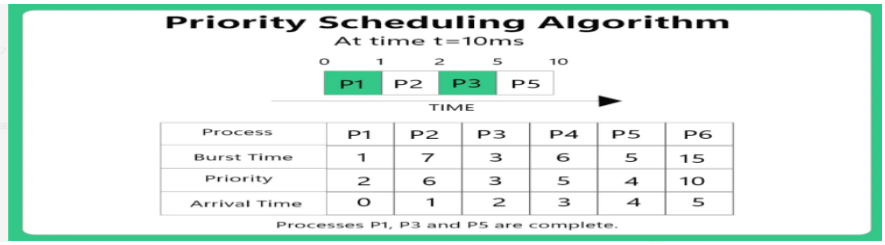


- At time t = 5

There are P2 (Priority = 6), P4 (Priority = 5), P5 (Priority = 4) and P6 (Priority = 10) that have arrived, out of which P5 has the highest priority as it has lowest value. Thus P5 will continue.

P5 has execution time of 5 ms. In this time P7 will have also arrived but. That also has lower priority than P5 thus P5 will continue its execution time.

- Till time t = 10 P5 will run and complete its processing.

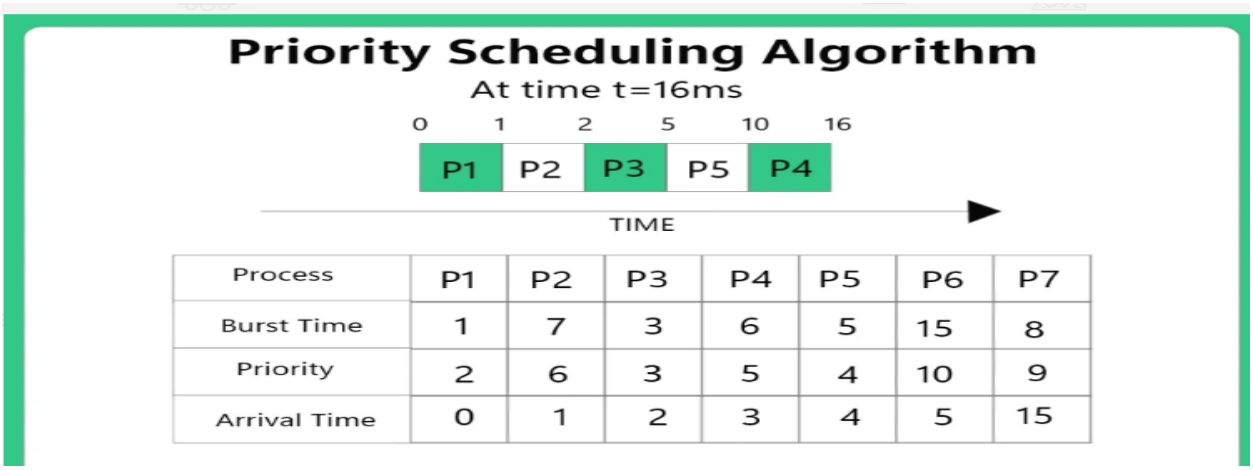


- At time t = 10

P2 P4 and P6 have arrived and P7 which will arrive at 15 ms.

Thus P4 will run as it has lowest priority of 5 and has burst time of 6 ms.

However, at t = 15ms P7 has arrived but, still priority of P4 is higher. It will continue to run.



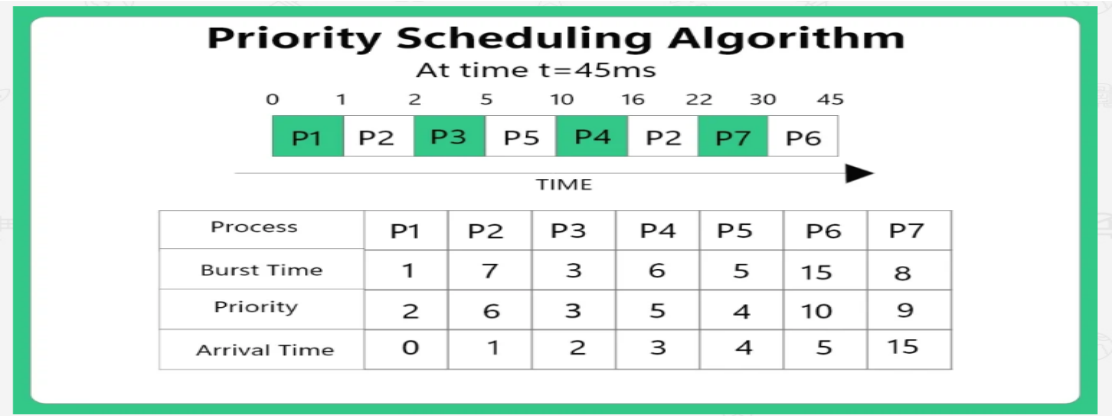
- At time t = 16 ms

All processes have arrived, thus there is no need for any type of preemption. It will follow normal priority procedure.

Currently processes are P2, P6 and P7. P2 has lowest priority of 6 and then P7 has next of 9 and finally P6 of 10.

Thus P2 (Burst time = 7 but 1ms was served previously, so 6ms) -> P7 and P8 will run in sequence and complete the whole process.

- At t = 45 ms process will complete.



Process	P1	P2	P3	P4	P5	P6	P7
Burst Time	1	7	3	6	5	15	8
Priority	2	6	3	5	4	10	9
Arrival Time	0	1	2	3	4	5	15

1. Turnaround Time = Completion Time – Arrival Time
2. Waiting Time = Turn Around Time – Burst Time

Average Waiting Time = (0+14+0+7+1+25+7)/7 = 54/7 = 7.71ms

Convoy Effect in Scheduling Operating System

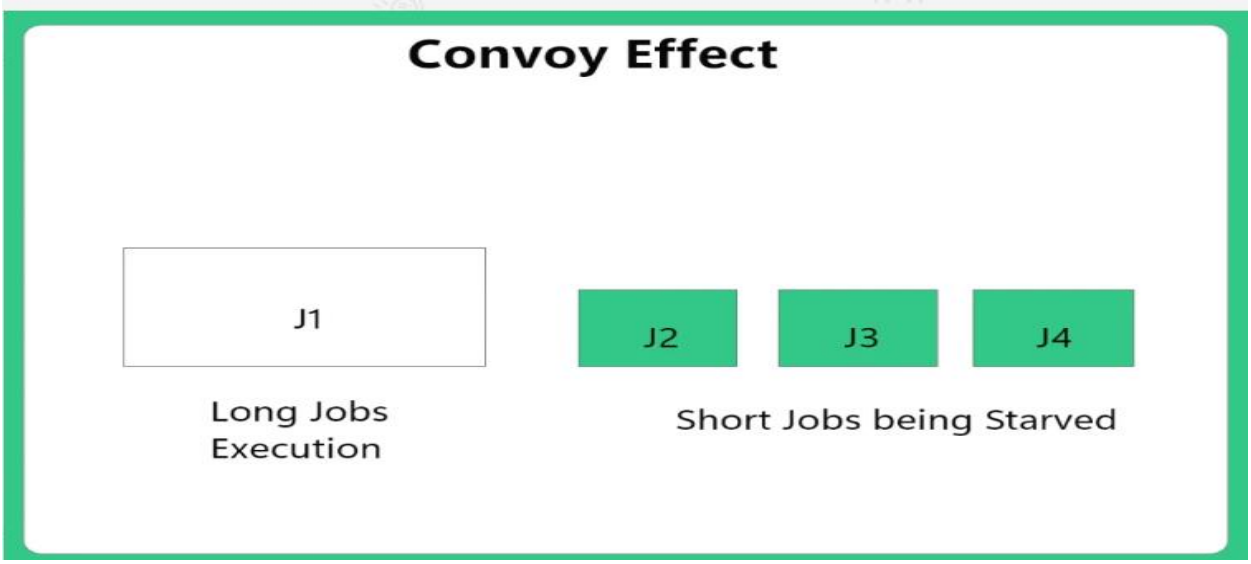
Convoy Effect in Scheduling Operating System

Convoy effect is a phenomenon which occurs in First Come First Serve Scheduling Algorithm. Consider a process that is too large that occurs at the very start of the ready queue. Due to this long process other processes in the queue may starve for a very long time.

To understand this consider an example. Where a very long train is passing by and smaller cars and other vehicles may have to wait for a very very long time to pass by.

So, as in First Come First Serve Algorithm (FCFS), Jobs are executed as they come. As FCFS is a non-preemptive algorithm, meaning jobs can’t be preempted (stopped) if they have begun execution, so unless the process does not get finished other process can’t get CPU resources.

This situation can be likened to a convoy passing through the road. So, unless the convoy passes completely, other people can’t use the road.



Steps are as following below which leads to convoy effect:

- The I/O bound processes are first allocated CPU time ,quickly get executed and goto I/O queues as they are less CPU intensive.
- Now, the CPU intensive process is allocated CPU time. As its burst time is high, it takes time to complete.
- While the CPU intensive process is being executed, the I/O bound processes complete their I/O operations and are moved back to ready queue.
- However, the I/O bound processes are made to wait as the other CPU intensive process still hasn't finished. This leads to I/O devices being idle and hence performing inefficiently.
- When the CPU intensive process gets executed, it is sent to the I/O queue so that it can access an I/O device.
- Meanwhile, the I/O bound processes get their required CPU time and move back to I/O queue.
- However, they are made to wait because the CPU intensive process is still accessing an I/O device. As a result, the CPU is sitting idle now.

Hence in Convoy Effect, one slow process affects and slows down the performance of the other processes, and leads to wastage of CPU time and other resources.

To avoid Convoy Effect, preemptive scheduling algorithms like Round Robin Scheduling can be used – as the smaller processes don't have to wait much for CPU time – making their execution faster and leading to less resources sitting idle.

Difference Between Scheduler and Dispatcher

Scheduler

Schedulers are system software that handle the removal of the running process from the CPU and the selection of another process. It selects a process out of several processes that need to be executed.

There are three types of schedulers in an operating system.

1. Long Term Scheduler

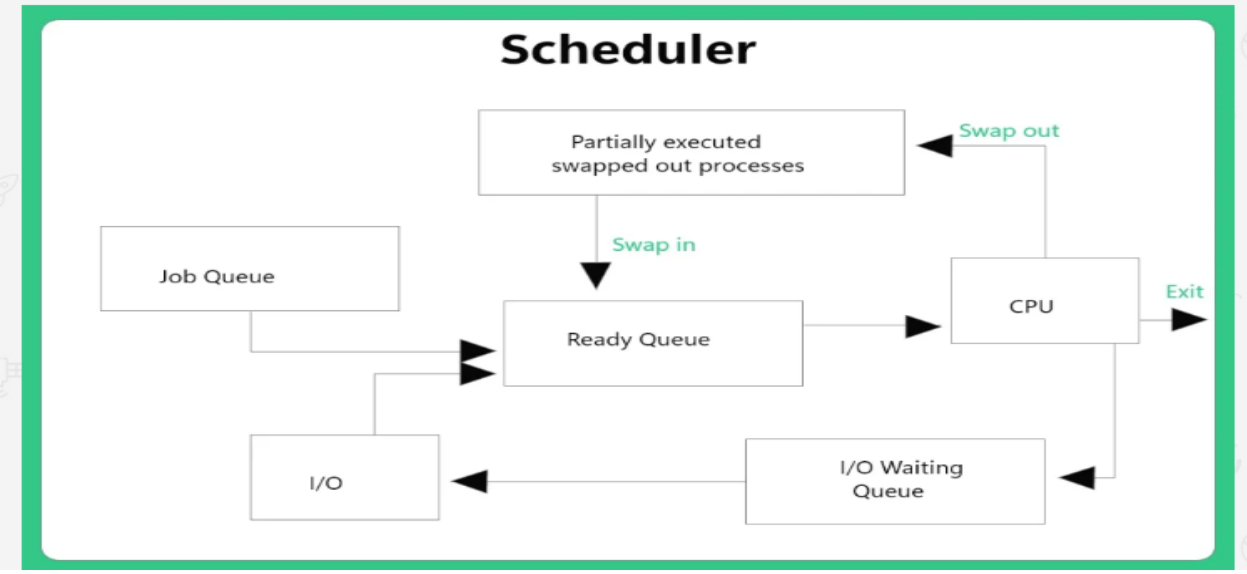
It is also known as a job. It decides the degree of multi-programming of the system. In simple words, when multiple processes are waiting for the execution, processes having small size are placed in the secondary storage. Then long term scheduler selects the processes from the job queue and brings that process to the ready queue in the main memory.

2. Medium-Term Scheduler

A running process needs input/output operations. The process goes to waiting state when not in need. This process is called suspended. However, CPU is used to its full utilization, while running the other process. Meanwhile, the suspended process is transferred back to the secondary memory. Now, the transferred process can return back to the main memory and continue its execution. This transfer of suspended process to the secondary memory is called swapping out and bringing the process back to the main memory is known as swapping in**.** It removes processes from memory, and thus, reduces the degree of multi-programming.

3. Short Term Scheduler

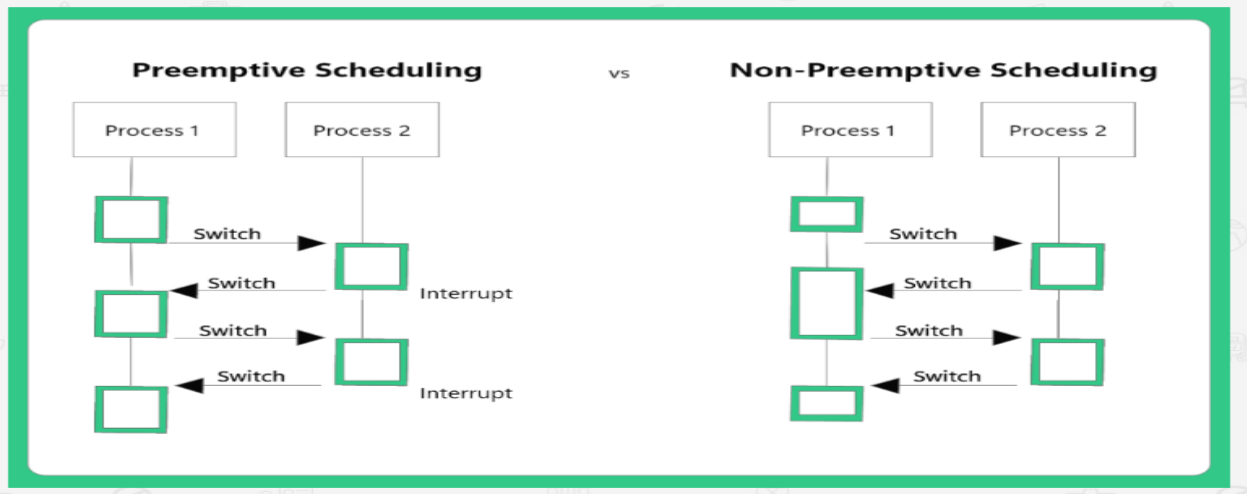
It is also known as CPU scheduler. It selects a process in the ready queue that should be allocated for execution. When it picks the process from the ready queue, the previous task goes to the waiting state. This process is completed in a fraction of seconds to avoid wasting the CPU time.



Preemptive Scheduling vs Non Preemptive Scheduling

Preemptive Scheduling vs Non Preemptive Scheduling

The process manager in an operating system is responsible for process scheduling. It is an operating system activity in which a running process is removed from the CPU and is replaced by another high priority process following certain set of rules. Process scheduling is an important part of a multi programming system where multiple processes need execution. The process of scheduling can be categorised into two, Pre-emptive and non-pre-emptive scheduling.



Pre-emptive Scheduling

Scheduling in which a process is switched from the running state to the ready state and from the waiting state to the ready state. In this concept, the CPU is allocated to the process for a limited

time and is again taken back when its time is over. The current state of the program is saved and is resumed again when the process is assigned to the CPU.

Non Pre-emptive Scheduling

In this case, once the process is allocated the CPU, it is only released when the processing is over. The states into which a process switches are the running state to the waiting state. It is an approach in which a process is not interrupted when it is under process by the CPU.

Preemptive Scheduling

Overview

In an operating system (OS), a process scheduler performs scheduling of process. It includes switching of processes between the ready queue and waiting queue and allocating them to the CPU. The operating system assigns priority to each process and maintains these queues. The scheduler selects the process from the queue and loads it into memory for execution. This process scheduling takes place in two types: preemptive scheduling and non-preemptive scheduling.

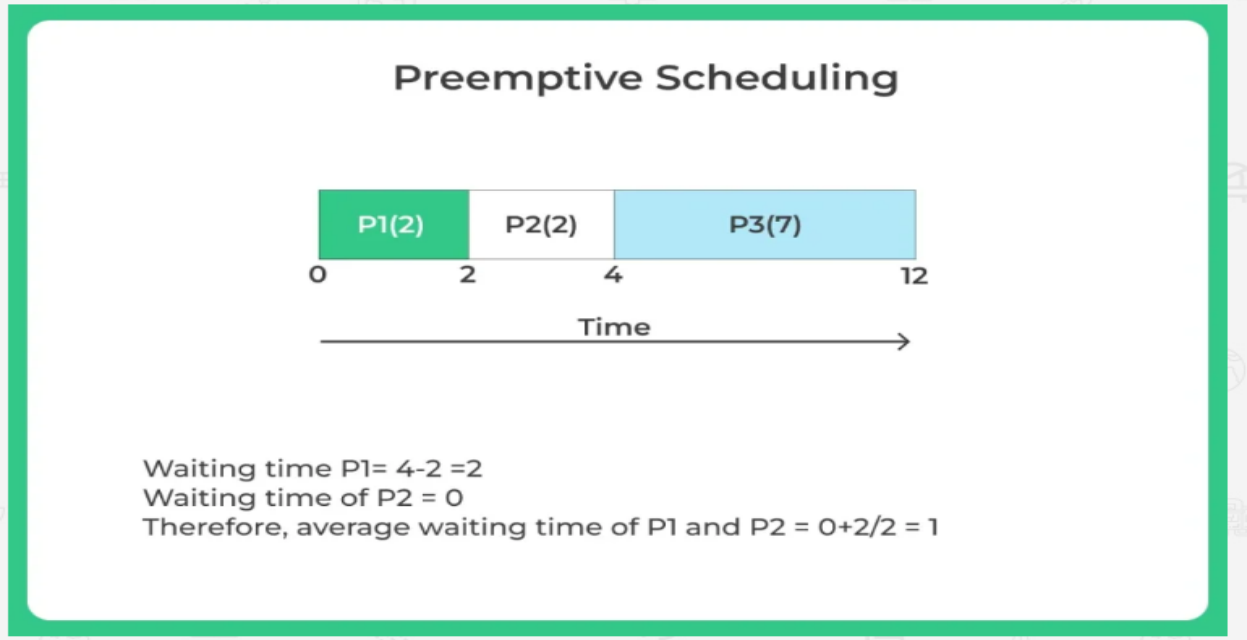
Preemptive Scheduling

In preemptive type of scheduling algorithm, a low priority process gets suspended from its execution, if a high priority process is waiting in the same queue for its execution. It is used when a process or task switches from running state to ready state or from waiting state to ready state. In this scheduling, the higher priority process is executed first.

At times, it is necessary to run a process that has a higher priority. If a high priority process arrives in the ready queue, it does not have to wait for the current process to complete its burst time. Instead, the running process is interrupted and is placed in the ready queue until the process with high priority is utilizing the CPU cycles. In this way, each process in the ready queue gets some time to run CPU.

Preemptive Scheduling Example

P1 and P2 have different arrival times in the queue. Based on their priority or burst time, these processes are interrupted and allocated to the CPU. A process P1 arrives at time 0 and is allocated the CPU. Process P2 arrives at time 2, before P1 finishes execution. See the table below. [table id=628 /] Learn more about preemptive type scheduling algorithms through **Shortest Job First Scheduling (SJF) algorithm here**.



Types of Preemptive Scheduling algorithms

Shortest Job First (SJF) Scheduling

It is also called the **Greedy In Preemptive Shortest Job First Scheduling**, the processes are put into the ready queue as per their arrival time. When a process with the shortest burst time arrives, the existing process is removed from execution, and the process with the shortest burst time is executed first.

Round-Robin (RR) Scheduling

It is also called **time-slicing scheduling**. It is primarily designed for a time-sharing In Round-Robin (RR) Scheduling, the process is allocated to the CPU for the specific time period (time slice). If the process completes its execution within this time period, then it is removed from the queue or else it has to wait for another time slice.

Non Preemptive Scheduling in Operating System (OS)

Overview

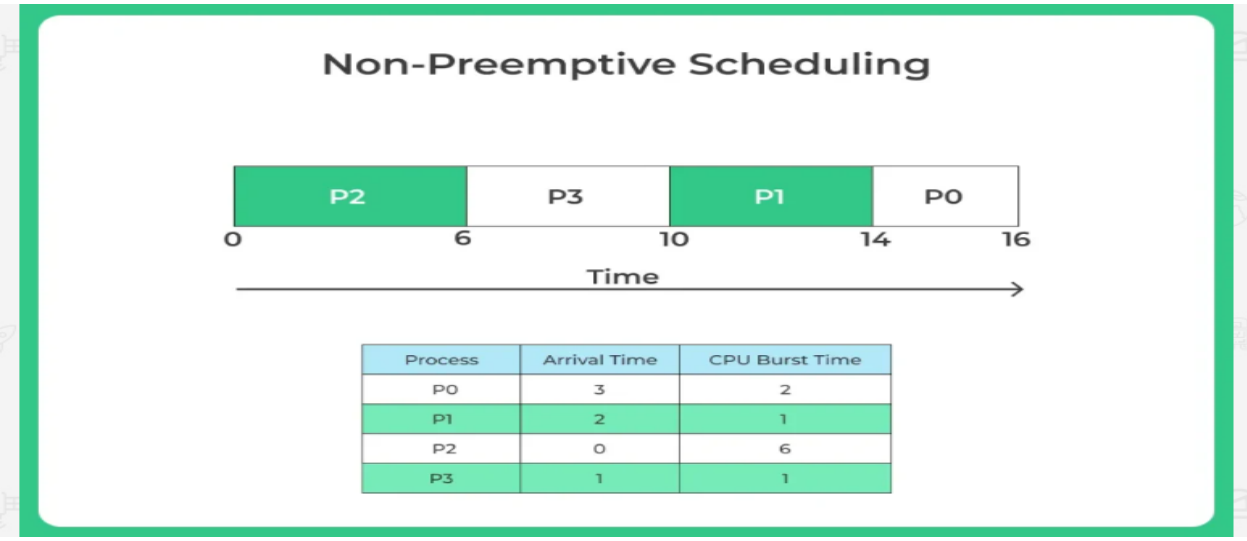
An operating system often assigns the CPU to processes that need execution. The method used to carry out such a function is known as **Scheduling**. The method can be of two types, **pre-emptive, and non-preemptive scheduling**. In the first case, every process is allocated a fixed amount of time to the CPU. When this time is over, the process is replaced by another process which is waiting for the state. On the other hand, the second type of scheduling is the **non-preemptive scheduling** in which, the process releases the CPU only after its execution is completed.

Non Pre-emptive Scheduling

In this case, once the process is allocated the CPU, it is only released when the processing is over. The states into which a process switches are the running state to the waiting state. It is an approach in which a process is not interrupted when it is under process by the CPU.

Non-Preemptive Type Scheduling
Some of the important points about this approach are:

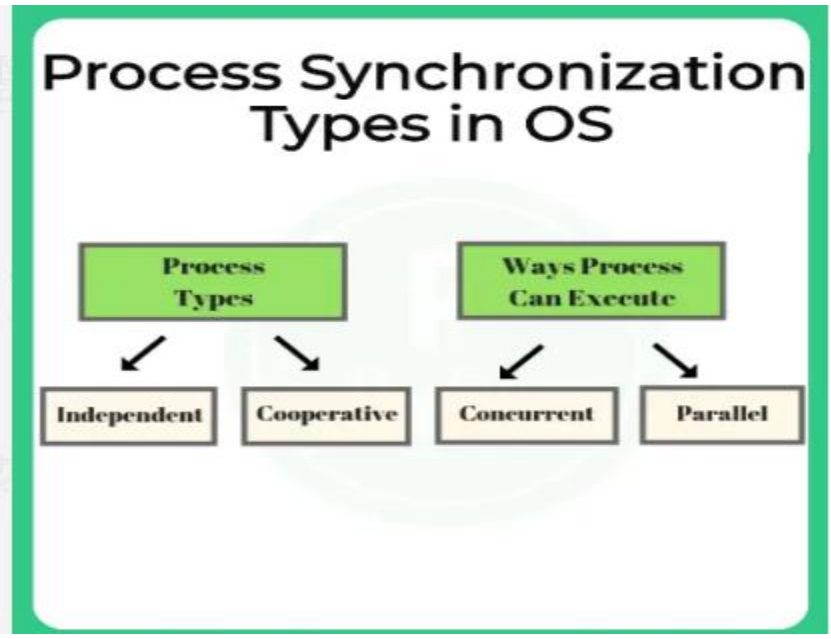
- Once the CPU starts executing a process, it finishes before picking up another process.
- Less CPU utilization as compared to pre-emptive scheduling.
- In this case, the response time and the waiting time is more.
- Once a process enters the running state, its state is not deleted from the scheduler until the process is finished executing or finishes its service time.
- If a process with a long burst time is being executed, then a process with minimum burst time might have to starve.
- The scheduling approach is also known as rigid.
- Some of its examples are FCFS, SJF, Priority, etc.



Process Synchronization in Operating System (OS)

Process Synchronization in Operating System
There are two ways any process can execute –

- **In Concurrent Execution –** the CPU scheduler switches rapidly between processes. A process is stopped at any points and the processor is assigned to another instruction execution. Here, only one instruction is executed at a time.
- **Parallel execution –** 2 or more instructions of different process execute simultaneously on different processing cores.



Why is Process Synchronisation Important

When several threads (or processes) share data, running in parallel on different cores , then changes made by one process may override changes made by another process running parallel. Resulting in inconsistent data. So, this requires processes to be synchronized, handling system resources and processes to avoid such situation is known as Process Synchronization.

Classic Banking Example –

- Consider your bank account has 5000$.
- You try to withdraw 4000$ using **net banking** and simultaneously try to withdraw via **ATM** too.
- For **Net Banking** at time t = 0ms bank checks you have 5000$ as balance and you're trying to withdraw 4000$ which is lesser than your available balance. So, it lets you proceed further and at time t = 1ms it connects you to server to transfer the amount
- Imagine, for ATM at time t = 0.5ms bank checks your available balance which currently is 5000$ and thus let's you enter ATM password and withdraw amount.
- At time t = 1.5 ms ATM dispenses the cash of 4000$ and at time t = 2 net banking transfer is complete of 4000$.

Effect on the system

Now, due to concurrent access and processing time that computer takes in both ways you were able to withdraw 3000$ more than your balance. In total 8000$ were taken out and balance was just 5000$.

How to solve this Situation

To avoid such situations **process synchronisation is used**, so another concurrent process P2 is notified of existing concurrent process P1 and not allowed to go through as there is P1 process which is running and P2 execution is only allowed once P1 completes.

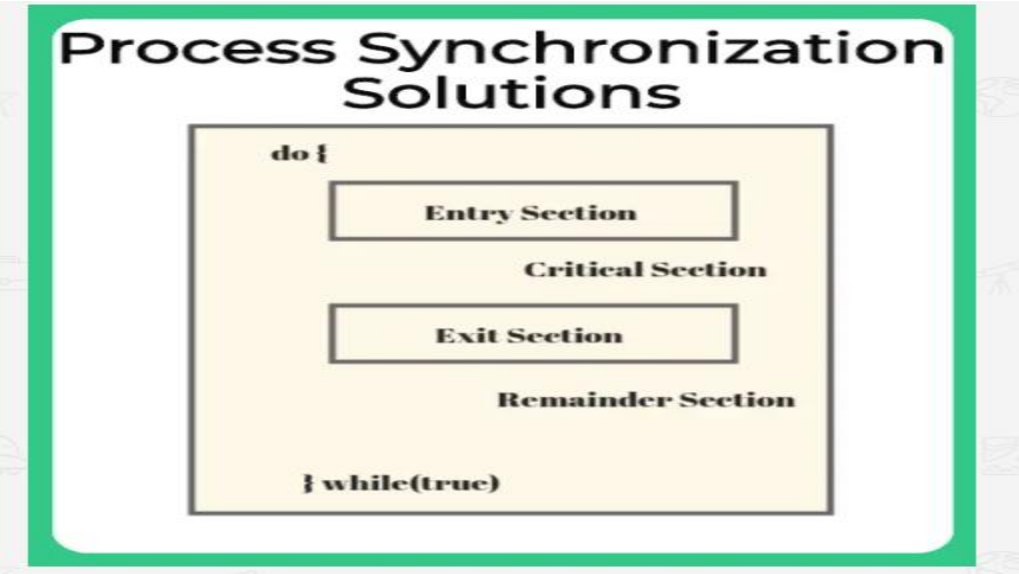
Process Synchronization also prevents **race around condition**. It's the condition in which several processes access and manipulate the same data. In this condition, the outcome of the execution depends upon the particular order in which access takes place.

Types of Process Synchronization

On the basis of synchronization, processes are categorized as one of the following two types:

- **Independent Process** : Execution of one process does not affects the execution of other processes.
- **Cooperative Process** : Execution of one process affects the execution of other processes.

To understand more about process synchronization, a through understanding of process structure is required. So, let's have a look at the elements of process .



The main blocks of process are –

1. **Entry Section –** To enter the critical section code, a process must request permission. Entry Section code implements this request.
2. **Critical Section –** This is the segment of code where process changes common variables, updates a table, writes to a file and so on. **When 1 process is executing in its critical section, no other process is allowed to execute in its critical section.**
3. **Exit Section –** After the critical section is executed , this is followed by exit section code which marks the end of critical section code.
4. **Remainder Section –** The remaining code of the process is known as remaining section.

Solution for Process Synchronization

Mutual Exclusion

Processes have to abide to this rule under this, only one process at a time is allowed to operate inside critical section. All cooperating processes should follow this.

Many hacks on banking system try to inject code inside this to break this rule. So this is one of the most important part of concurrent process codes.

Progress

As the name suggests, the system must promote progress and maximum utilisation of the system. If there is no thread in the critical section then, and there are threads waiting for critical section utilisation. Then they must be allowed immediately to access critical section.

Bounded Waiting

If a process P1 has asked for entry in its critical section. There is a limit on the number of other processes that can enter critical section. Since if a large number of processes are allowed to enter critical section. Then the process P1 may starve forever to gets its entry.

To further solve such situations many more solutions are available which will be discussed in detail in further posts –

1. **Semaphores**
2. **Critical Section**
3. **Test and Set**
4. **Peterson's Solution**
5. **Mutex**

The following are not required to be known as of now, read them later, we will cover these in our future posts –

1. Synchronization Hardware
2. Mutex Locks

Inter Process Communication in Operating System (OS)

There are 2 types of process –

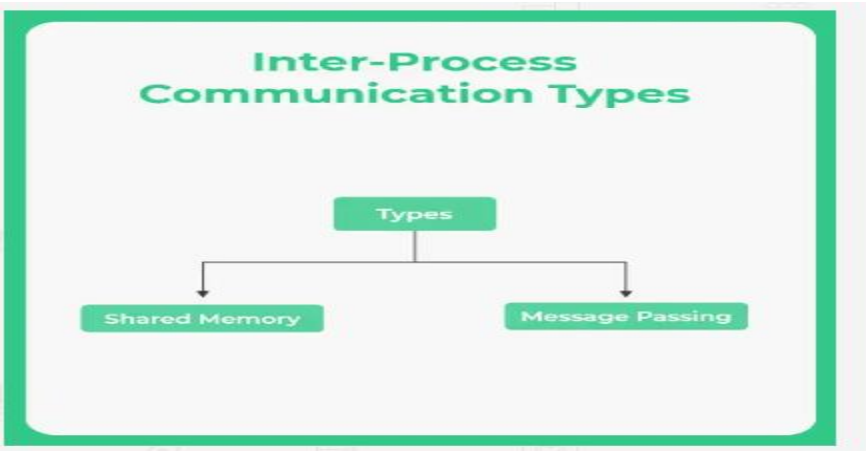
- **Independent Processes –** Processes that do not share data with other processes.
- **Cooperating Processes –** Processes that shares data with other processes.

The cooperating process requires Interprocess communication (IPC) mechanism.

Inter-Process Communication is the mechanism by which cooperating process share data and information.

The mechanism that will allow them to exchange data and information are the following:–

- Shared memory
- Message Passing

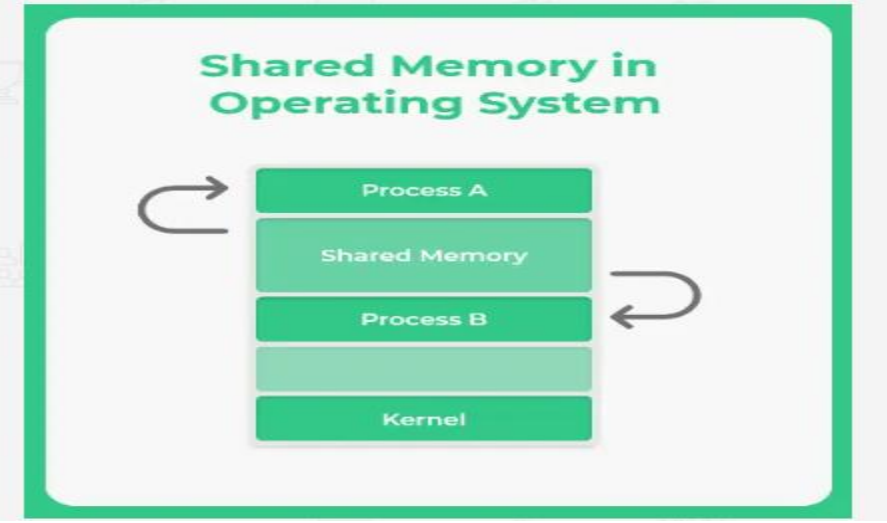


Types
Let's look at few of the important points of each .

Shared Memory

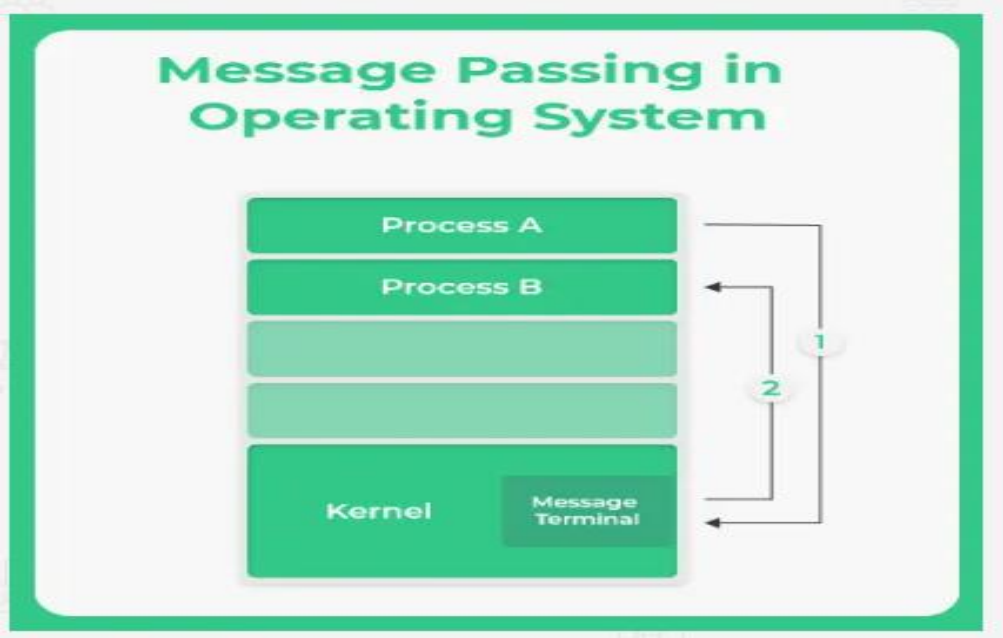
1. A particular region of memory is shared between cooperating process.

2. Cooperating process can exchange information by reading and writing data to this shared region.
3. It's faster than Memory Parsing, as Kernel is required only once, that is, setting up a shared memory . After That, kernel assistance is not required.



Message Parsing

1. Communication takes place by exchanging messages directly between cooperating process.
2. Easy to implement
3. Useful for small amount of data.
4. Implemented using System Calls, so takes more time than Shared Memory.



What is Critical Section in Operating System

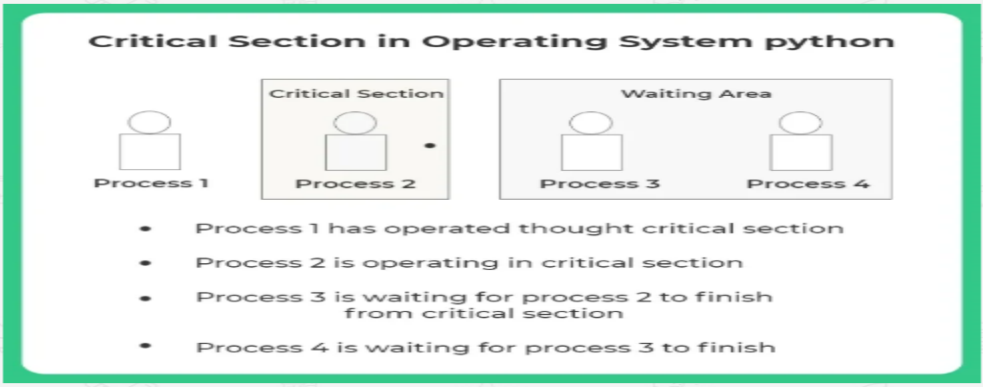
Critical section

Critical Section is any piece of code that is shared between different processes. If more than one process tries to operate in critical section we can reach to undesired output or else if one process starts to operate in critical section and does not release it then it may lead to deadlock, therefore, a single process is allowed to enter at a time. Whenever a process has entered a critical section, it doesn't allow other processes to enter it by locking it for time being.

First, we have an entry section where each process checks whether the critical section is lock-free or not. If there is a lock then the process needs to wait else if there is no lock then the current process acquires the lock and enters the critical section. Once the process has been executed it releases the lock it has acquired. Whenever the next process comes it acquires the lock and then moves inside the critical section.

Necessary and sufficient conditions for a solution to the critical section problem:

1. Mutual Exclusion — Only a single process is allowed to enter a critical section at a time. If a process pi is executing in its critical section then no other process is allowed to enter the critical section at that time.
2. Progress — a process operating outside of its critical section cannot prevent other processes from entering theirs; processes attempting to enter their critical sections simultaneously must decide which process enters eventually.
3. Bounded Waiting — a process attempting to enter its critical region will be able to do so eventually



Real-life example

Bathroom: At a time only a single person can use the bathroom. Whenever a person enters the bathroom, he/she locks the bathroom so that no other person can enter it. Any other person who wants to use the bathroom needs to wait outside till the person comes out. Here washroom is an example of a critical section and people are examples of the process.

Pseudocode of Critical Section Problem

```
while(true){

// entry section

process_entered=1;

//loop until process_eneterd in 0

while(process_entered);

// critical section

// exit section

process_entered=0

// remainder section

}
```

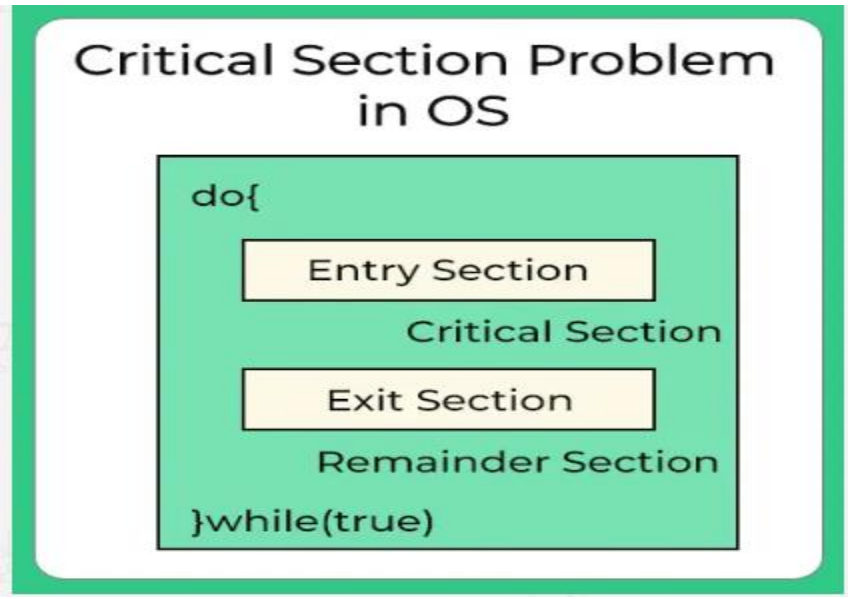
Critical Section Problem in Operating System (OS)

Critical Section Problem in Operating system

Critical Section problem is a classic computer science problem in many banking systems, where there are many shared resources like memory, I/O etc. Concurrent access may override changes made by other process running in parallel.

We already know about Critical section of code from Process synchronization post. Critical section is a segment of code in which process changes common variable, updates file etc.

The Critical Section Problem is to design a protocol which processes use to cooperate .To enter into critical section, a process requests permission through entry section code. After critical section code is executed, then there is exit section code, to indicate the same.



Constituents of Critical Section

The main blocks of process are –

1. **Entry Section –** To enter the critical section code, a process must request permission. Entry Section code implements this request.
2. **Critical Section –** This is the segment of code where process changes common variables, updates a table, writes to a file and so on. **When 1 process is executing in its critical section, no other process is allowed to execute in its critical section.**
3. **Exit Section –** After the critical section is executed , this is followed by exit section code which marks the end of critical section code.
4. **Remainder Section –** The remaining code of the process is known as remaining section.

Example of Negative Effect

When several threads (or processes) share data, running in parallel on different cores , then changes made by one process may override changes made by another process running parallel. Resulting in inconsistent data. So, this requires processes to be synchronized, handling system resources and processes to avoid such situation is known as Process Synchronization.

CLASSIC BANKING EXAMPLE –

- Consider your bank account has 5000$.
- You try to withdraw 4000$ using net banking and simultaneously try to withdraw via ATM too.
- For Net Banking at time t = 0ms bank checks you have 5000$ as balance and you're trying to withdraw 4000$ which is lesser than your available balance. So, it lets you proceed further and at time t = 1ms it connects you to server to transfer the amount

- Imagine, for ATM at time t = 0.5ms bank checks your available balance which currently is 5000$ and thus let's you enter ATM password and withdraw amount.
- At time t = 1.5 ms ATM dispenses the cash of 4000$ and at time t = 2 net banking transfer is complete of 4000$.

EFFECT ON THE SYSTEM

Now, due to concurrent access and processing time that computer takes in both ways you were able to withdraw 3000$ more than your balance. In total 8000$ were taken out and balance was just 5000$.

Critical Section Problem conditions

A critical solution problem satisfies 3 things –

1. Mutual Exclusion must be preserved.
2. Progress Requirement must be satisfied.
3. Bounded waiting time requirement is met.

Let's look at each of these in detail –

1. **Mutual Exclusion is preserved –** If a process P1 is executing in its critical section, no other process can execute in critical section.
2. **Process or Progress Requirement must be satisfied –** If no process is executing in its critical section, then the decision as to which process will go next into executing its critical section code is decided by only those process which are in entry section, that is, not before it. That is the process which are in remainder section can not take part in this decision. Also, this selection has to be made in certain time, it can't be postponed indefinitely.
3. **Bounded Waiting time requirement is met –** If a process A has made a request to enter its critical section, then there is a limit on the number of times , other processes go before into their critical section. So, waiting time, as a result is bonded. After a certain time, the process gets to be in critical Section, doesn't wait indefinitely.

Advanced Topics

To handle critical section is Operating System, There are 2 ways –

- Preemptive Kernel
- Non Preemptive Kernel

Preemptive Kernel

- Allows process to be preempted while it's running in kernel mode.
- Can be prone to race around condition, if poorly designed. So, must be carefully designed .

- Difficult to Design for SMP Architecture as in this environment, it's possible for 2 Kernel -Mode processes to run simultaneously on 2 different processes.
- It's more responsible, as less risk for any process to run indefinitely.
- Suitable for Real Time Programming.

What is Race Around Condition ?

If many kernel processes in OS, it may lead to race around condition.

Eg – Consider a kernel data structure that maintains a list of all open files in system. List is modified if a new file is opened or closed. If 2 process simultaneously try to open files , it may separate updates to list leading to race around condition

Non Preemptive Kernel –

- Does not allow a process running in Kernel Mode to be preempted, that is stopped. Only after the process exits the Kernel Mode by self, will the next process be given the kernel.
- Free from Race Around Condition as only 1 process is executed at a time in kernel.

The software based solution to critical section problem is Peterson's Algorithm, Semaphores, Monitors etc

Mutex in Operating system (OS)

Mutex in Operating System

Mutex lock in OS is essentially a variable that is binary nature that provides code wise functionality for mutual exclusion. At times, there maybe multiple threads that may be trying to access same resource like memory or I/O etc. To make sure that there is no overriding. Mutex provides a locking mechanism.

Only one thread at a time can take the ownership of a mutex and apply the lock. Once it done utilising the resource and it may release the mutex lock.

Mutex Highlights

Mutex is very different from Semaphores, please read Semaphores or below and then read the difference between mutex and semaphores here.

1. Mutex is Binary in nature
2. Operations like Lock and Release are possible
3. Mutex is for Threads, while Semaphores are for processes.
4. Mutex works in user-space and Semaphore for kernel
5. Mutex provides **locking** mechanism

6. A thread may acquire more than one mutex
7. Binary Semaphore and mutex are different

Semaphores in OS

While mutex is a lock (wait) and release mechanism. Semaphores are signalling mechanisms that signal to processes the state of the Critical section in OS and grant access to the critical section accordingly.

Semaphores use the following methods to control access to critical section code –

1. Wait
2. Signal

Semaphore Types

We have two types of semaphores –

1. **Binary Semaphore** –
    1. Only True/False or 0/1 values
2. **Counting Semaphore** –
    1. Non-negative value

Semaphore Implementation

Wait and Signal are two methods that are associated with semaphores. While some articles are represented as wait(s) or signal(s) however in some blogs are represented as p(s) for wait and v(s) for signal

Wait p(s) or wait(s)

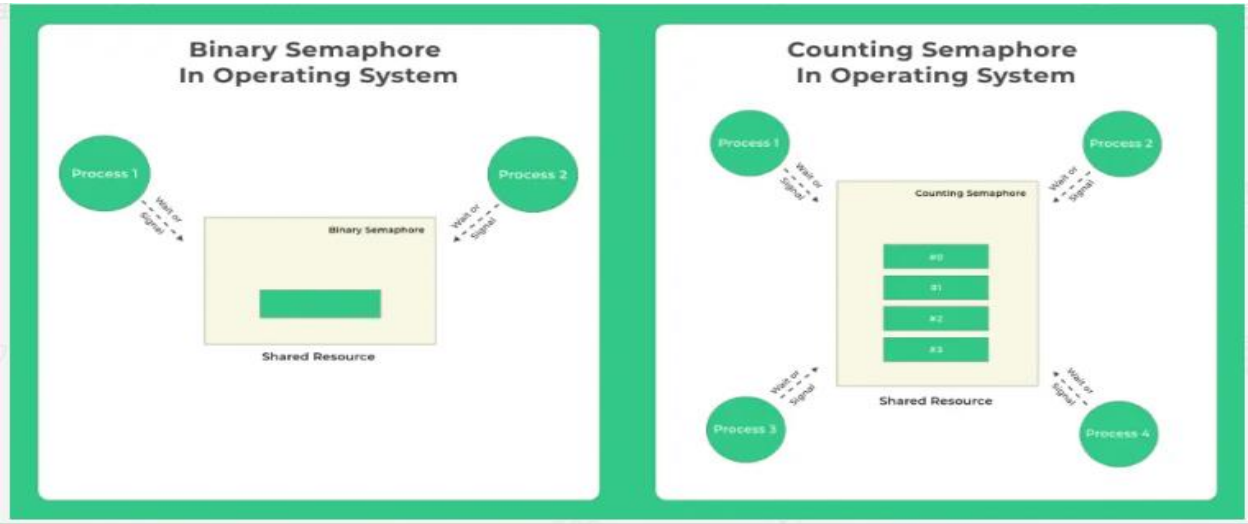
1. Wait decrements the value of semaphore by 1

Signal v(s) or signal(s)

1. Signal increments the value of semaphore by 1

Semaphore

1. Semaphore can only have positive values
2. Before the start of the program, it is always initialised to
    1. n in Counting semaphore (Where n is the number of processes allowed to enter critical section simultaneously)
    2. 1 in the case of a binary semaphore



Signal Operations

Increments semaphore by 1

Signals that the process has completed its critical section execution

signal(S)

{

S++;

}

Wait Operations

Wait operation decrements the value of semaphore S if S is a positive number

Else if S is 0 or negative then code gets stuck at while loop as it keeps implementing infinitively

The semi-colon after while forces while loop definitively if S is 0 or negative

Thus the code doesn't move ahead in hopes that the value of S will increase because of some other signal operation elsewhere

Code Logic for Incrementing – Decrementing value of Semaphore –

wait(S)

{

```
    while (S<=0);

   S--;
}
```

Actual Working for both functions together to achieve access of critical section –

```
    // some code

    wait(s);



    // critical section code



    signal(s);



    // remainder code
```

Semaphore in Operating System (OS)

Signal Operations

- Signal operation is simple
- It increases the value of semaphore by 1 as shown below

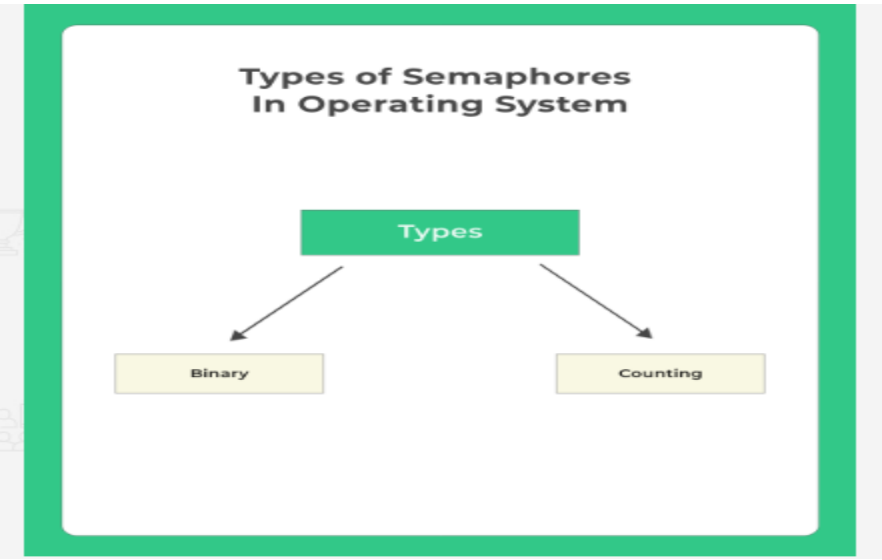
Semaphore in Operating System

Semaphore uses signalling mechanism to allow access to shared resources namely by two –

1. Wait
2. Signal

We recommend you to go through this article below first –

What is Mutex Lock in Operating System



What is Semaphore in Operating System

Semaphore Types

There are two types of Semaphores –

1. Binary Semaphore – Only True/False or 0/1 values
2. Counting Semaphore – Non-negative value

Semaphore Implementation

Semaphore can have two different operations which are wait and signal. In some books wait signals are also denoted by P(s) and signal by V(s). Where s is a common semaphore.

Wait p(s) or wait(s)

1. Wait decrements the value of semaphore by 1

Signal v(s) or signal(s)

1. Signal increments the value of semaphore by 1

Semaphore

1. Semaphore can only have non-negative values
2. Before the start of the program, it is always initialised to
    1. n in Counting semaphore (Where n is the number of processes allowed to enter critical section simultaneously)
    2. 1 in the case of a binary semaphore

Mutex Vs Semaphore in OS

What is the difference between Mutex and Semaphore in Operating System

Mutex is essentially a locking and releasing mechanism and however, Semaphore is a signalling mechanism. Both are used for Critical section and mutual exclusion problems.

Most people think that Binary Semaphore and Mutex are essentially the same but they are not.

We first recommend reading the post about Semaphores and mutex here before proceeding further.

Differences

Mutex

Example – Imagine mutex as a key to a single toilet. Only one person can be inside the toilet at a time and he would want to lock the toilet and others waiting for toilet access must wait for person already inside to release the toilet. When finished the person gives the key to next person in the queue.

- Mutex is for thread.
- Mutex is essentially atomic and singular in nature.
- Mutex is binary in nature.
- Operations like Lock and Release are possible
- Mutex works in userspace
- Only one thread can acquire a mutex at a time
- Mutex is an object

Semaphore

Example – Imagine a bathroom with 4 identical toilets here, identical keys can open the bathrooms. Here as many as 4 people can use the bathroom simultaneously and they wait and signal one another about the occupancy of the bathrooms

- Semaphore is for processes
- Semaphores are also atomic but not singular in nature
- Semaphores are of two types that are binary and counting
- Semaphores work in kernel space
- Only one process can acquire binary semaphore at a time but multiple processes can simultaneously acquire semaphore in case of counting semaphore
- Semaphore is an integer variable

What is SpinLock ?

While a process is in critical section, any other process which waits to enter critical section must loop continuously in call to acquire. This is a spinlock as the process "Spins" while waiting for the lock to be available.

Advantages –

- No Context switch is required for a process waiting on a lock.
- Good for systems where it's known that locks are of short duration.
- Employed in Multiprocess system, where one thread can spin on 1 processor while another thread processes its critical section in another processor.

Semaphores –

- It's a Generalized Mutex.
- Semaphores are integer variable , which is accessed through 2 atomic operations, wait() and signal ().
- It's a signalling Mechanism.

Mutex in Operating System

Mutex lock in OS is essentially a variable that is binary nature that provides code wise functionality for mutual exclusion. At times, there maybe multiple threads that may be trying to access same resource like memory or I/O etc. To make sure that there is no overriding. Mutex provides a locking mechanism.

Only one thread at a time can take the ownership of a mutex and apply the lock. Once it done utilising the resource and it may release the mutex lock.

Mutex Highlights

Mutex is very different from Semaphores, please read Semaphores or below and then read the difference between mutex and semaphores here.

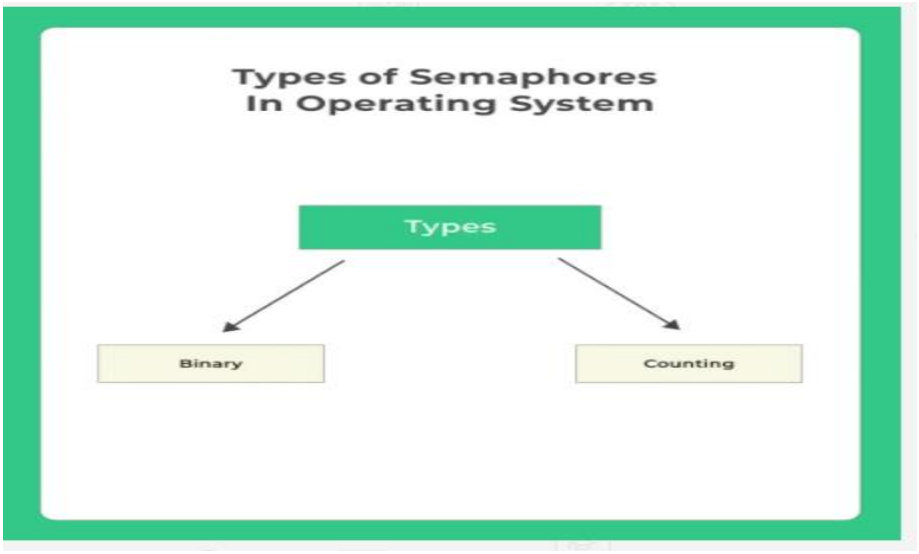
1. Mutex is Binary in nature
2. Operations like Lock and Release are possible
3. Mutex is for Threads, while Semaphores are for processes.
4. Mutex works in user-space and Semaphore for kernel
5. Mutex provides **locking** mechanism
6. A thread may acquire more than one mutex
7. Binary Semaphore and mutex are different

Semaphores in OS

While mutex is a lock (wait) and release mechanism. Semaphores are signalling mechanisms that signal to processes the state of the Critical section in OS and grant access to the critical section accordingly.

Semaphores use the following methods to control access to critical section code –

1. Wait
2. Signal



Atomic Operations in OS

Atomic Operations

Atomic operations are the operations that execute as a single unified operation. In simple terms, when an atomic operation is being executed, no other process can read through or modify the data that is currently used by atomic operation.

Atomic operations are used in concurrent programming where program operations run independently, without being interleaved by any other operation. These operations are used in many operating systems which implements parallel processing.

Type of Atomic Operations

Listed below are some of the atomic operations which are being used regularly:

1. **Fine-grained atomic operations:** These operations can be implemented without any interruptions. For example, loading and storing registers.
2. **Coarse-grained atomic operations:** These include a series of fine-grained operations which cannot be interrupted. For example a call of a synchronized method in JAVA.

Advantage and Disadvantage of Atomic Operations

Advantage

Atomic operations are usually faster when compared to locks. Also, they do not suffer from issues of deadlocks and convoying.

Disadvantage

One restraining factor of atomic operations is that they only execute a limited number of processes and cannot function efficiently when complicated processes are to be executed.

Example of Atomic Operation

- Consider a process in which the value of temp is to be fetched and incremented by 1.
- Let initially, the value of temp be 0.
- Now after the execution of the process the value of temp is increased by 1 and it becomes equal to 1.
- However, now this processes is pre-empted and a new process is initiated, which also use the value of temp and increase its value by 5.
- Now, the value of temp becomes 6 as the initial value was 1, which is then increased by 5.

- After, completion of this process, when the first process starts its execution again, the value of temp becomes 5+1=6.
- However, the expected result, in this case, was 1+1=2. Hence, a wrong solution is generated.
- Nevertheless, in the case of atomic operations, the value of temp can neither be read nor modified, thereby ensuring the accuracy of results.

"Atomic" comes from Latin and means "indivisible". The increment operator is not indivisible; it has 3 separate parts:

1. Load value from memory into register.
2. Increment register.
3. Store value from register back into memory.

So, for example, if you have two threads that try to increment a number, if the number starts from zero you could get either 1 or 2 as a result, at random. You get 1 when an interleaving similar to the below happens:

1. Thread 1 loads zero into a register.
2. Thread 2 loads zero into a register.
3. Thread 1 increments register (0 -> 1).
4. Thread 2 increments register (0 -> 1).
5. Thread 1 writes the value "1" into memory.
6. Thread 2 writes the value "1" into memory.

In many languages there are special variants of the increment operator that *are* atomic. Normally they are supported by the hardware itself (this is harder than it sounds; AFAIK it's not just the CPU, but the caches and main memory and everything needs to work together to provide this). You should either use those, or manual synchronization (say, a lock – so that the increments run start to finish, and others are prohibited from starting while another one is running).

Popular Algorithms:

Readers Writers Problem in Operating System (OS)

Readers Writers Problem in Operating System

This is a synchronisation problem which is used to test newly proposed synchronisation scheme. The problem statement is, if a database or file is to be shared among several concurrent process, there can be broadly 2 types of users –

- Readers – Reader are those processes/users which only read the data
- Writers – Writers are those processes which also write, that is, they change the data .

It is allowed for 2 or more readers to access shared data, simultaneously as they are not making any change and even after the reading the file format remains the same.

But if one writer(Say w1) is editing or writing the file then it should locked and no other writer(Say w2) can make any changes until w1 has finished writing the file.

Writers are given to be exclusive access to shared database while writing to database. This is called Reader's writer problem.

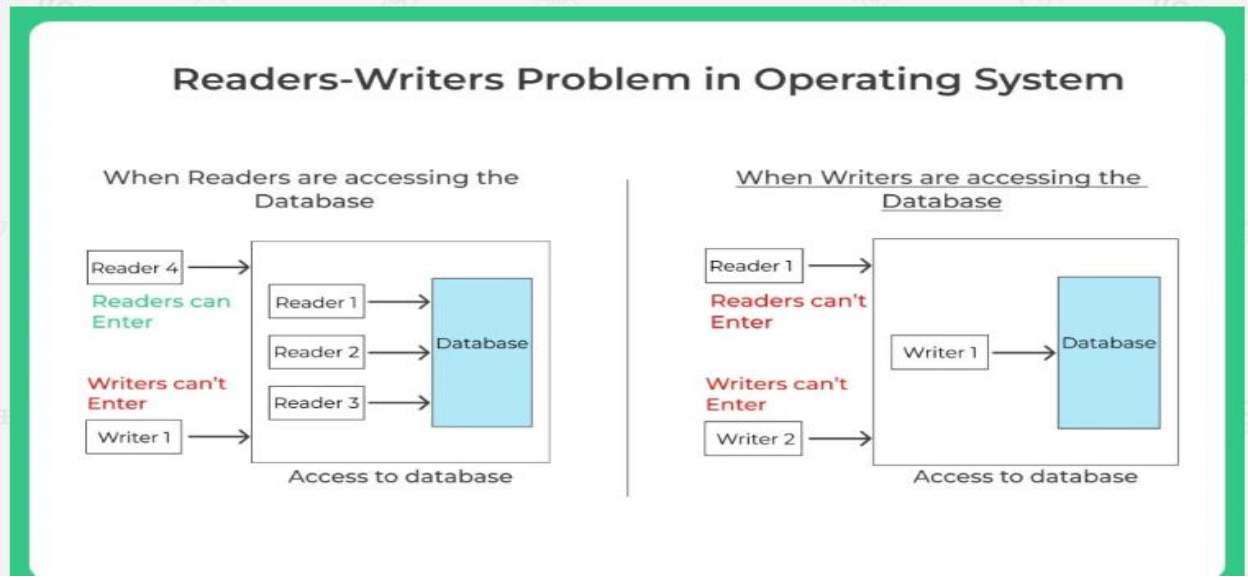
Highlights

- If writer W1 has begun writing process then
  - No additional writer can perform write function
  - No reader is allowed to read
- If 1 or more readers are reading then
  - Other readers may read as well
  - No writer may perform write function until all readers have finished reading

Explanations

In simple terms understand this as unlimited number of readers can read simultaneously. Only one writer can write at a time.

When a writer is writing no other writer can write to the file. A writer can not write when there are one or more than one readers reading. That is writer can only write when there is no readers or no writers accessing the resource.



Solution

Variables used –

1. Mutex – mutex (used for mutual exclusion, when readcount is changed)
    1. initialised as 1
2. Semaphore – wrt (used by both readers and writers)
    1. initialised as 1
3. readers_count – Counter of number of people reading the file
    1. initialised as 0

Functions –

There are two functions –

1. wait() – performs as –, which basically decrements value of semaphore
2. signal() – performs as ++. which basically increments value of semaphore

How does Wait and Signal work

To understand how readers and writers work we first need to understand that how atomic operations wait and signal work

Readers Writers Wait and Signal Implementation.

```
wait (Semaphore s)

{

  while (s == 0);

  s = s - 1;

}



signal (Semaphore s)

{
```

```
  s = s + 1;
}
```

What happened above:

semicolon after while , which results in the while loop getting executed again again. Until S value is non-zero.×Dismiss this alert.

Writers problem

```
while(TRUE)
{
   //if wait(wrt)returns true value,
   //then can enter critical section
   //if not allowed then, it keeps on waiting
       wait(wrt);


/* writer does some write operation */


   //increments w value again to 1
   //so other writers can writer
   signal(wrt);
```

}

1. Writer wants the access to critical section i.e. wants to perform the writing
2. First wait(wrt) value is checked
    1. if returns true then gets access
        1. does the write process
        2. performs signal(wrt) and increments value
    2. if returns false then
        1. doesn't get write access.

Readers Problem

```
while(TRUE)

{

    //a reader wishes to enter critical section

    //this mutex is used for mutual exclusion before readcount is changed

     wait(mutex);



    //now since he has entered increase the readers_count by 1

    readers_count++;



    // readers_count value now should be greater than or equal to 1



    // used ==1 not >=1 as we want to perform this only once.
```

```
if(readers_count == 1)
    // decrementing w value so no writer can enter writer section
    // as readers are reading
    // MCQ Question Fact -
    // Readers have more priority then writers.
        wait(wrt);

//this will ensure that now, other readers can enter critical section
signal(mutex);

/* perform the reading operation */

// a reader wants to leave after reading process
wait(mutex);
readers_count--;

if(readers_count == 0)
```

```
        // if readers_count is 0 we must restore w value to 1 so writers can write

            signal(wrt);



    // reader has now left

   signal(mutex);

}
```

Writer –

- Mutex Semaphore ensure mutual exclusion when read count is updated.
- Read_count -> tracks how many processes are currently reading the object.
- mutex -> Functions as a mutual exclusion semaphore for writers. Also used by first or last reader that enters or exits the critical section.
- If writer is in critical section and n readers are waiting, 1 reader is queued as mutex. And n-1 readers are queued or mutex.
- When signal (mutex) is executed, then one of 2 thing can happen -> Either waiting readers execute or 1 single writing process is executed. This is needed by scheduler.

The above solution to reader writer problem can be generalized to provide reader- writer lock on some system. To acquire a reader writer lock, one must specify the mode of lock- either read or write access.

If a process wanting to only read shared data , request reader-writer lock in read mode. A process waiting to only write on shared data must request the lock in write mode.

This reader writer lock is effective in –

- The application where reading and writing process are easily identifiable.
- Application having more readers than writers. As reader-writer locks require more overhead to establish than semaphore than semaphore or mutual exclusion locks. So, with the help of multiple readers, it compensates for the overhead of setting up readers-writer lock.

Dining Philosophers Problem in Operating System (OS)

Dining Philosophers in Operating System

Dining Philosophers in operating system essentially is a process synchronization example and helps understand how we can simultaneously utilise common resources of multiple processes together.

Entities –

- Noodles/Rice
- Chopsticks
- Philosophers

Problem Statement Imagine five philosophers sitting around a circular table and have a bowl of rice or noodles in the middle and there are five chopsticks on the table.

At any given instance, a philosopher will do –

- Thinking
- Eating

- Whenever the philosophers want to eat. He obviously will use two chopsticks together.
    - So to eat both chopsticks on his right and left must be free.
- Whenever he is thinking
    - He must put down both the chopsticks back at the table.

Deadlock conditionA deadlock condition may arise here, consider if all the philosophers get hungry simultaneously together and they pick up one chopstick on their left.

In this case they will wait indefinitely for their 2nd chopstick(right chopstick) and the deadlock will be there forever.

Rules and Solution

If a philosopher decides to eat

- He first will wait for the chopstick on his left to be free.
- If the left chopstick is already free he will pick that up.
- Then he will wait for the chopstick on his right
- And once the right chopstick is also free he will pick that up too and do the eating.

- Some of the ways to avoid deadlock are as follows:
- **Four Philosopher**
- There should be at most four philosophers on the table. In this way one chopstick will be extra if all decide to eat together. This one chopstick then can be used by one of our

philosopher and he will finish his eating and then two chopsticks are available then 1 more can eat and so on…

- **Even Odd**
- Solving with even and odd techniques, even philosopher must pick his right and off must pick his left.
- **Coordination**
- Only and only if both chopsticks are available at the same time for a philosopher then only he should pick them up,
- **Algorithm**

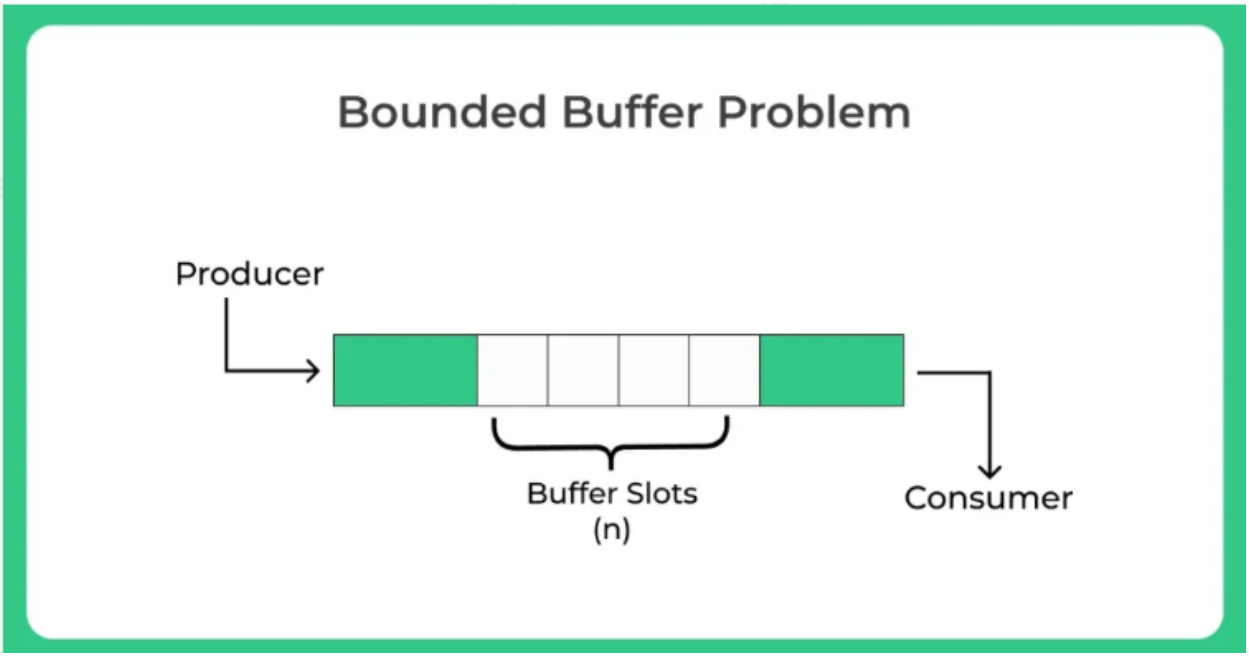
```
while(TRUE)

{

   // mod is used because if i=5,

   // next chopstick is 1 (dining table is circular)



   wait(stick[i]);

   wait(stick[(i+1) % 5]);



   /* eat */




   signal(stick[i]);

   signal(stick[(i+1) % 5]);



   /* think */

}
```

Bounded Buffer Problem in Operating System

Bounded Buffer Problem in Operating System using Semaphores

In Bounded Buffer Problem there are three entities storage buffer slots, consumer and producer. The producer tries to store data in the storage slots while the consumer tries to remove the data from the buffer storage.

It is one of the most important process synchronizing problem let us understand more about the same.



Problem

The bounded buffer problem uses Semaphore. Please read more about Semaphores here before proceeding with this post here.

We need to make sure that the access to data buffer is only either to producer or consumer, i.e. when producer is placing the item in the buffer the consumer shouldn't consume.

We do that via three entities –

- Mutex mutex – used to lock and release critical section
- empty – Keeps tab on number empty slots in the buffer at any given time
    - Initialised as n as all slots are empty.
- full – Keeps tab on number of entities in buffer at any given time.
    - Initialised as 0

Readers Writer Wait and Signal Implementation

```
wait(Semaphore s)
{
  while(s == 0);
  s = s-1;
}

signal(Semaphore s)
{
  s = s+1;
}
```

What happened above:semicolon after while , which results in the while loop getting executed again again. Until S value is non-zero.×Dismiss this alert.

Producer Buffer Solution

```
do
  {
    // wait until empty > 0 and then decrement 'empty'
    // that is there must be atleast 1 empty slot

    wait (empty);

    // acquire the lock, so consumer can't enter

    wait (mutex);
```

```
    /* perform the insert operation in a slot */

    // release lock

    signal (mutex);

    // increment 'full'

    signal (full);
  }
while (TRUE)
```

Consumer Buffer Solution

```
do
{
    // wait until full > 0 and then decrement 'full'
    // should be atleast 1 full slot in buffer

    wait(full);

    // acquire the lock

    wait(mutex);
```

```
    /* perform the remove operation in a slot */


    // release the lock


    signal(mutex);


    // increment 'empty'


    signal(empty);
}

while(TRUE);
```

Consumer Producer Problem

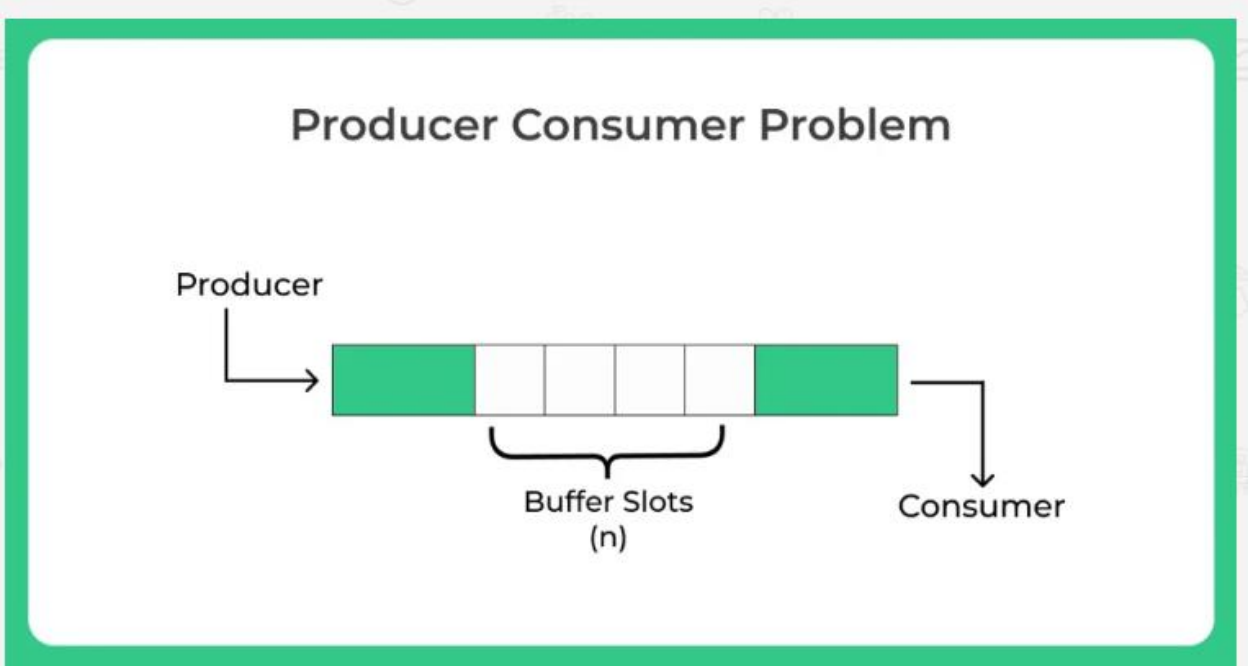
Producer Consumer Problem using Semaphores in Operating System

The producer-consumer problem is a classic multiprocess synchronization problem. In the Producer-Consumer problem there is a producer, a producer produces some items, and a consumer who consumes the items produced by the producer. Producers and consumers share the same fixed-size memory buffer.

Producer Consumer Problem using Semaphores

Producer consumer problem is one classic example of mutual exclusion, process synchronisation using semaphores. It basically synchronises data production and consumption happening simultaneously in a system where there is fixed amount of storage size. The system has the following –

1. Producer
2. Consumer
3. Storage bound of size n



Process of Producer Consumer

The processes is synchronised via a mutex which provides mutual mutual exclusion in the critical section area. There is unit called as empty which defines the number of empty slots in the storage. There is unit called as full which defines the number of full storage slots in the system.

The number of storage slots are initialised as n, the number of empty slots are initialised as n since all slots are empty at the starting similarly the full unit is initialised as 0 as there is no data initially.

It is important to understand wait and signal first from the image above to understand how the problem works.

Producer Consumer Signal and Wait

```
wait(Semaphore s)
{
  while(s == 0);
  s = s-1;
}
```

```
signal(Semaphore s)
{
  s = s+1;
}
```

What happened above:semicolon after while , which results in the while loop getting executed again again. Until S value is non-zero.

Producer

```
do
  {


// produce an item
// the producer can only produce if there is atleast 1 empty slot
// thus empty must be >0 and then it also decrements the value


    wait (empty);


// access to critical section is blocked for consumer section


    wait (mutex);


// place in buffer


// access to critical section is released
```

```
    signal (mutex);

// value of full is incremented

    signal (full);

  }
while (true)
```

Consumer

```
do
  {

// consumer should wait until there is atleast one item in storage
// full value should be greater than 0 and then decremented

    wait (full);

// access to critical section is blocked for producer section

    wait (mutex);

// remove item from buffer
```

```
// access to critical section is released for producer section

    signal (mutex);

// value of empty is incremented as item was consumer

    signal (empty);

// consumes item

  }

while (true)
```

Peterson's Algorithm for Critical Section Problem

Peterson's Algorithm For Critical Section Problem –

- This is a software based solution to Critical Section Problem.
- Doesn't work on modern architectures.
- It's for 2 processes which alternate execution between then critical section and remainder section. Say, P1 is the first process and P2 is the second process.
- The 2 processes should share 2 data items with each other.

```
int turn

Boolean flag [2]
```

- Turn – It indicates the process who should enter into its critical section.
- Flag Array – It tells whether a process is ready to enter its critical section. Let flag[0] indicate process P1. If flag[0] = true , then Process P1 is ready to execute in its critical section. flag[1] indicates process P2. If flag[1] = true, then Process P2 is ready to execute in its critical section.

Now let's take a look at peterson's Algorithm –

```
do {
    flag[i] = true;
    turn = j;

    while(flag[j]&& turn==j);

    // Critical Section

    flag[i]= false;

    // Remainder Section

} While(true);
```

- First , p1 sets flag[0] true, then sets turn to j . So that if P2 wants to enter Critical Section, it can do so.
- If P1 , P2 try to execute at same time, then turn is first changed to i, then j or it could be vice-versa. But, the important point is, only one of these 2 process is allowed to enter its critical section. The second value gets overwritten.

Features of Peterson's Solution Algorithm –

- Does not require any special hardware.
- Uses Busy waiting (Spinlock).

What is Race Around Condition ?

If many kernel processes in OS, it may lead to race around condition.

Eg – Consider a kernel data structure that maintains a list of all open files in system. List is modified if a new file is opened or closed. If 2 process simultaneously try to open files , it may separate updates to list leading to race around condition

Peterson's Algorithm for Critical Section Problem

Peterson's Algorithm For Critical Section Problem –

- This is a software based solution to Critical Section Problem.
- Doesn't work on modern architectures.
- It's for 2 processes which alternate execution between then critical section and remainder section. Say, P1 is the first process and P2 is the second process.
- The 2 processes should share 2 data items with each other.

```
int turn

Boolean flag [2]
```

- Turn – It indicates the process who should enter into its critical section.
- Flag Array – It tells whether a process is ready to enter its critical section. Let flag[0] indicate process P1. If flag[0] = true , then Process P1 is ready to execute in its critical section. flag[1] indicates process P2. If flag[1] = true, then Process P2 is ready to execute in its critical section.

Now let's take a look at peterson's Algorithm –

```
do {

    flag[i] = true;

    turn = j;


    while(flag[j]&& turn==j);


    // Critical Section
```

```
flag[i]= false;


// Remainder Section


} While(true);
```

First , p1 sets flag[0] true, then sets turn to j . So that if P2 wants to enter Critical Section, it can do so.

If P1 , P2 try to execute at same time, then turn is first changed to i, then j or it could be vice-versa. But, the important point is, only one of these 2 process is allowed to enter its critical section. The second value gets overwritten.

Features of Peterson's Solution Algorithm –

1. Does not require any special hardware.

2.Uses Busy waiting (Spinlock).

3.What is Race Around Condition ?

If many kernel processes in OS, it may lead to race around condition.

Eg – Consider a kernel data structure that maintains a list of all open files in system. List is modified if a new file is opened or closed. If 2 process simultaneously try to open files , it may separate updates to list leading to race around condition

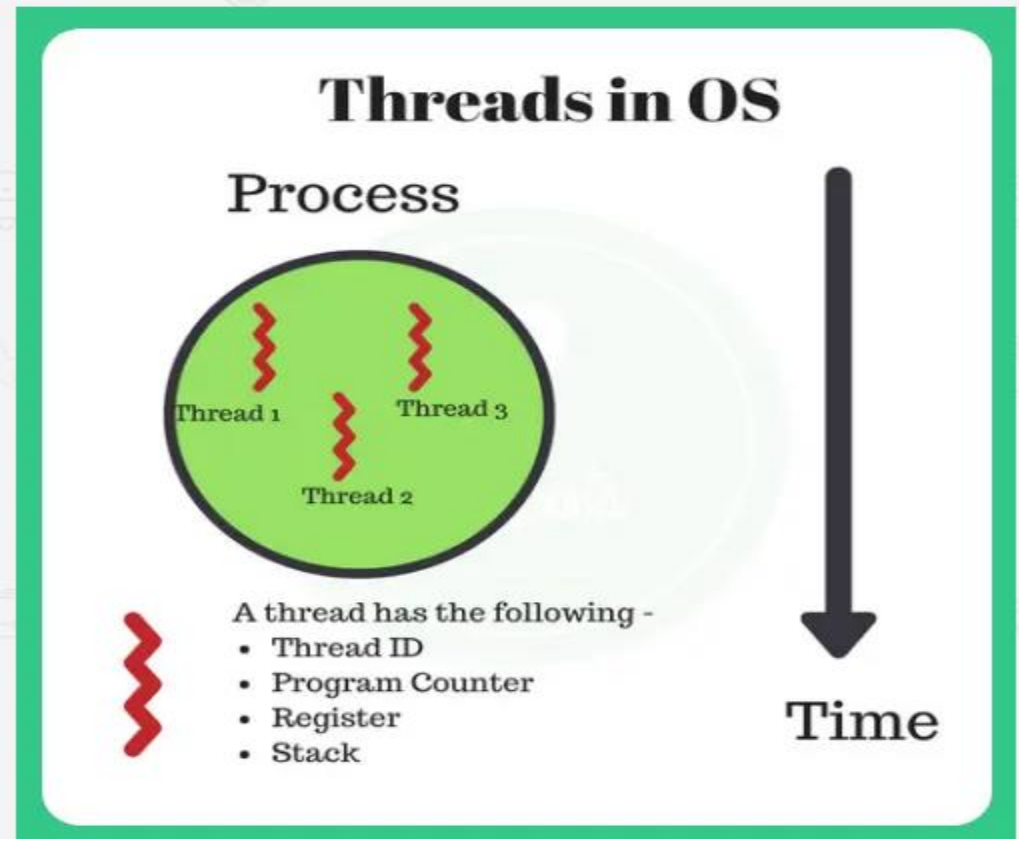
Threads in Operating System (OS)

Threads in Operating System

Threads are **Smallest sequence of programmed instruction** that can be managed independently by a scheduler.

- A thread comprises of its own
  - **Thread ID –** Unique ID for a thread in execution

- **Program counter –** Keeps track of instruction to execute
- **System Register set –** Active Variables of thread
- **Stack –** All execution history (Can be used for debugging)



Its important to know difference between thread, process and program.

Program, **Process** and **Threads** are three basic concepts of the operating systems about which every computer science engineer must be familiar with. Here, we will explain what each of them is all about and how they differ from each other.

What is a program?

Program is an executable file containing the set of instructions written to perform a specific job on your computer. For example, **notepad.exe** is an executable file containing the set of instructions which help us to edit and print the text files.

Programs are not stored on the primary memory in your computer. They are stored on a disk or a secondary memory on your computer. They are read into the primary memory and executed by the kernel. A program is sometimes referred as **passive entity** as it resides on a secondary memory.

What is a process?

Process is an executing instance of a program. For example, when you double click on a notepad icon on your computer, a process is started that will run the notepad program.

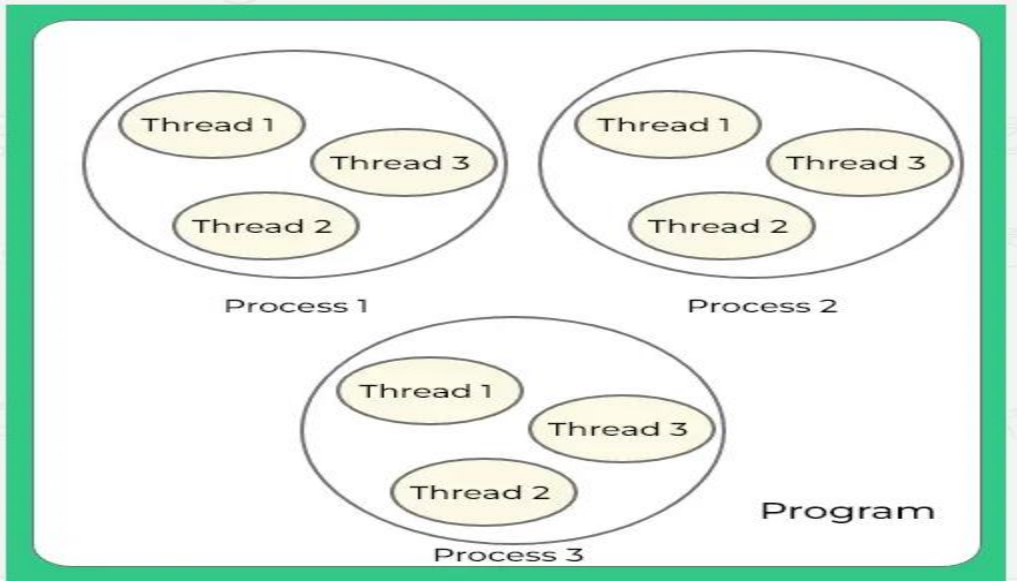
A process is sometimes referred as **active entity** as it resides on the primary memory and leaves the memory if the system is rebooted. Several processes may related to same program. For example, you can run multiple instances of a notepad program. Each instance is referred as a process.

What is a thread?

Thread is the smallest executable unit of a process. For example, when you run a notepad program, operating system creates a process and starts the execution of main thread of that process.

A process can have multiple threads. Each thread will have their own task and own path of execution in a process. For example, in a notepad program, one thread will be taking user inputs and another thread will be printing a document.

All threads of the same process share memory of that process. As threads of the same process share the same memory, communication between the threads is fast.



Facts

- Multiple threads can exist within one process , and they can also share resources such as memory. Different process do not share those resources.
- A thread shares with other thread information like code segment, data segment and open files. All other thread segments are aware of any alternation made by a thread memory.

- Threads can be scheduled preemptively or cooperatively .
- Helps to use to achieve parallelism by dividing a process into multiple threads .

Thread Versus Process –

1. Process are independent whereas threads exist as subsets of process.
2. Processes have separate address space whereas Threads within a same process same address space.
3. Processes are resource intensive whereas Threads are lightweight processes.
4. If 1 Process is blocked , no other processes can execute unless the blocked process is unblocked . whereas , if 1 thread is waiting, a 2nd thread in same task can be run.
5. No other process changes data of other process . They are independent, whereas one thread can read, write or change other tread.

Now, let's have a look at how threads are implemented

- In Single processor system, they implemented multithreading by time slicing, CPU switches between software threads. This adds a context overhead as context switching happens very often. It happens very fast, so it appears to user as if these tasks are performed in parallel.
- In multiprocessor, multiple threads execute in parallel, each processor executes 1 thread simultaneously while other processor execute other thread.

Advantages of thread –

- Context Switching is faster between threads compared to context switching between processor.
- We can effectively utilize CPU in multiprocessor as we can make 1 process with many threads run in different processor of multiprocessor , thus increasing speed.
- Sharing and communication is simplified in threads.
- Threads can implement parallelism.

Types of Threads

Threads are implemented in 2 ways –

1. User level threads
2. Kernel level threads

Read more about User level thread and Kernel level thread here on this post

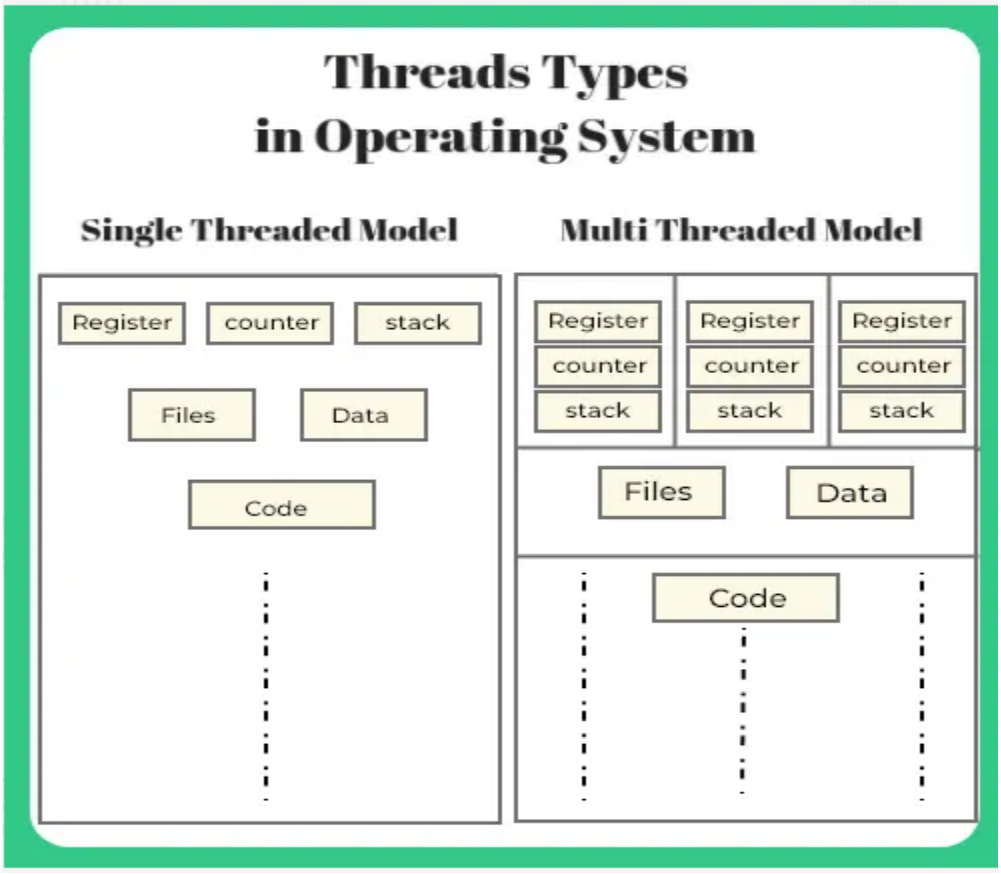
Different Types of Thread Models

Read more about various thread models here

Also, there are two different types of processes

1. Single-threaded processes

2. Multi-threaded processes



User level thread vs Kernel level thread in Operating System (OS)

Difference Between User Level and Kernel Level Threads
User Level Thread

Features of User Level Thread –

- Implemented by user and managed by run-time system (user-level library).
- OS is not aware of existence of threads.
- OS manages the user level threads as if they are single threaded process.
- Creation of thread switching between thread and synchronizing thread are all done via procedural call.

Advantage of User Level Thread-

- Are extremely fast and efficient, because the switching between thread takes almost same time as a procedural call.
- OS doesn't need to interfere ,everything can be without it.

- They do not modify OS.
- Threads are cheap and fast at user level.
- User level thread can be implemented in OS, who do not implement threads.

Disadvantage of User Level Thread-

- Since OS isn't aware of user level thread , scheduler does not schedule them properly. Eg -> Java thread, POSIX thread etc.
- If 1 user level thread performs blocking operation, entire process will be blocked.

Kernel Level Thread

Features of Kernel Level Thread –

- OS managed threads.
- OS knows about and manages the thread.
- Kernel has a thread table which keeps tracks of all threads in the system.
- System calls create and manage threads.

Advantages of Kernel Level Thread ->

- As kernel has full knowledge of threads, so scheduler handles the processes better.
- For application that frequently block, Kernel level threads are good.

Disadvantages of Kernel Level Thread ->

- Slow and inefficient than user level.
- Hardware support is needed.
- Context switching time is more. Eg – Window Solaris
- If 1 kernel thread performs blocking operation, then another thread can continue execution.

User Level Threads
VS
Multi-Threaded Model

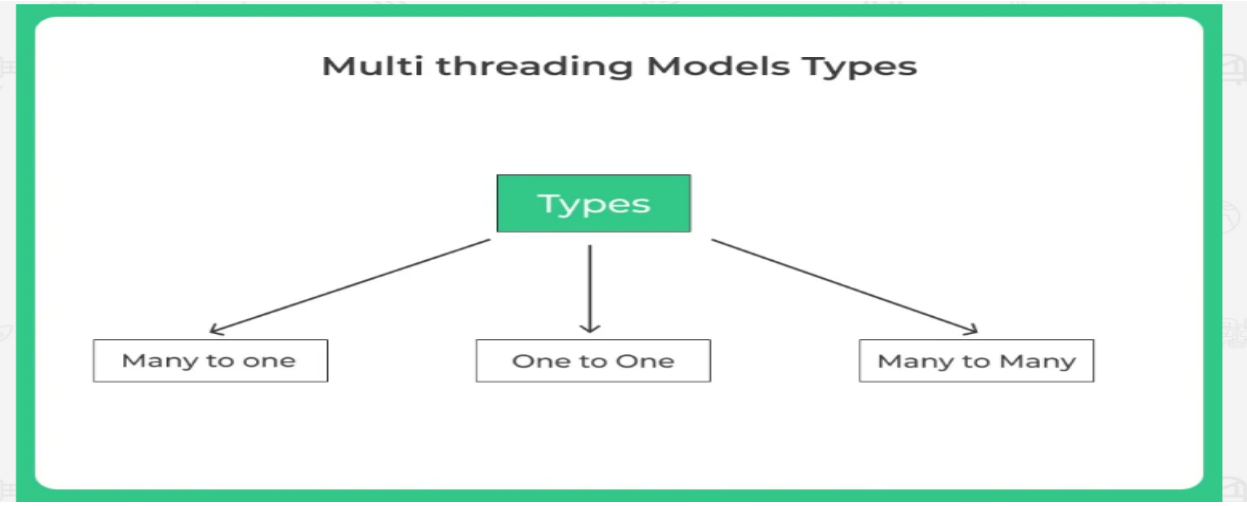
User level Threads	Multi Threaded Model
User thread are implemented by Users	Kernal threads are implemented by OS
OS doesn't recognise user level threads	Kernel threads are recognized by OS
Implementation is easy	Implementation is complicated
Context switch time is less	Context switch time is more
Context switch – no hardware support	hardware support is needed
If one user level thread perform blocking operation then entire process will be blocked	If one kernel level thread perform blocking operation then another thread can continue execution.

Multithreading models in Operating System

Multithreading Models in Operating System

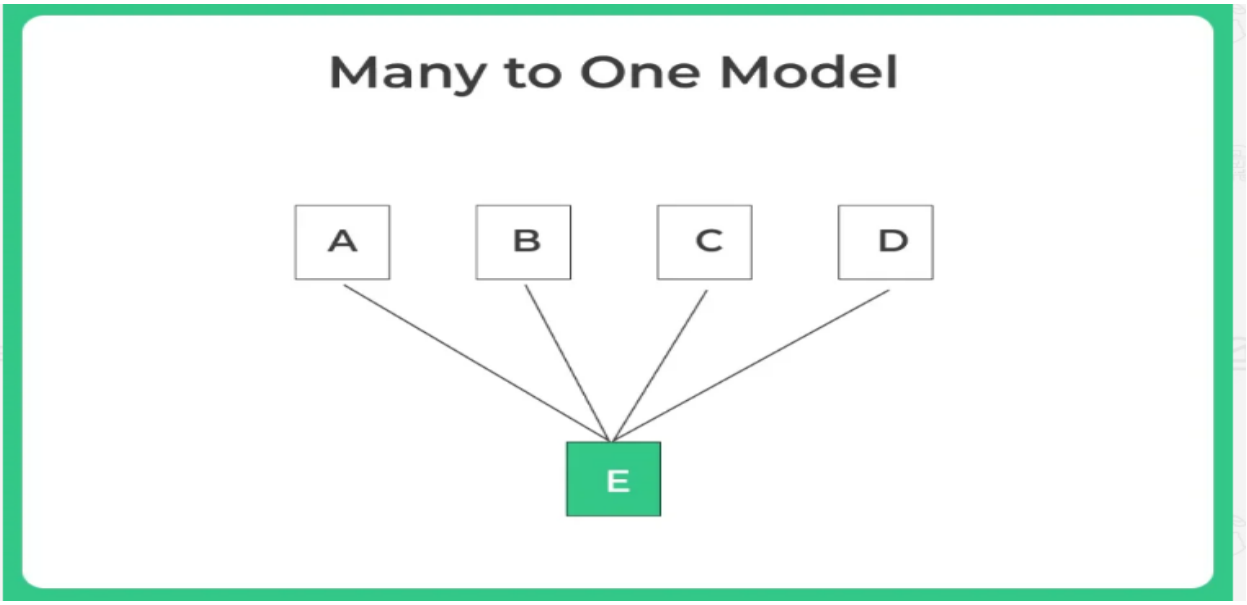
In Operating Systems, where both User Level and kernel level thread exists, there exists a relationship between the two. User Level Thread is mapped to Kernel Level Thread by one of the 3 models –

1. Many to One Model
2. One to One Model
3. Many To Many Model



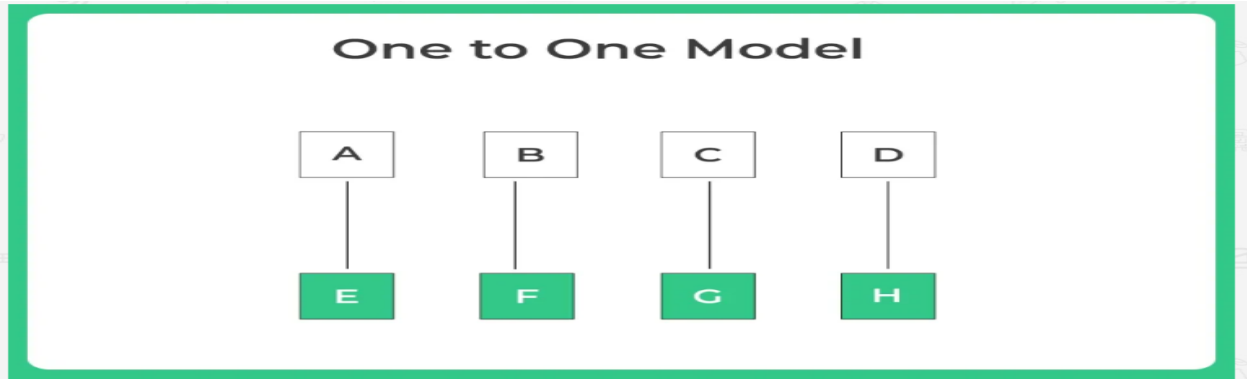
Many To One Model

- Many User Level Threads are mapped to one Kernel level Thread.
- Does Not support parallel processing of threads in Multicore systems, Because only 1 thread can access kernel at a particular time.
- If 1 thread makes a blocking call, then entire process will stop.
- This is rarely used, mainly cause it does not support Multicore system.



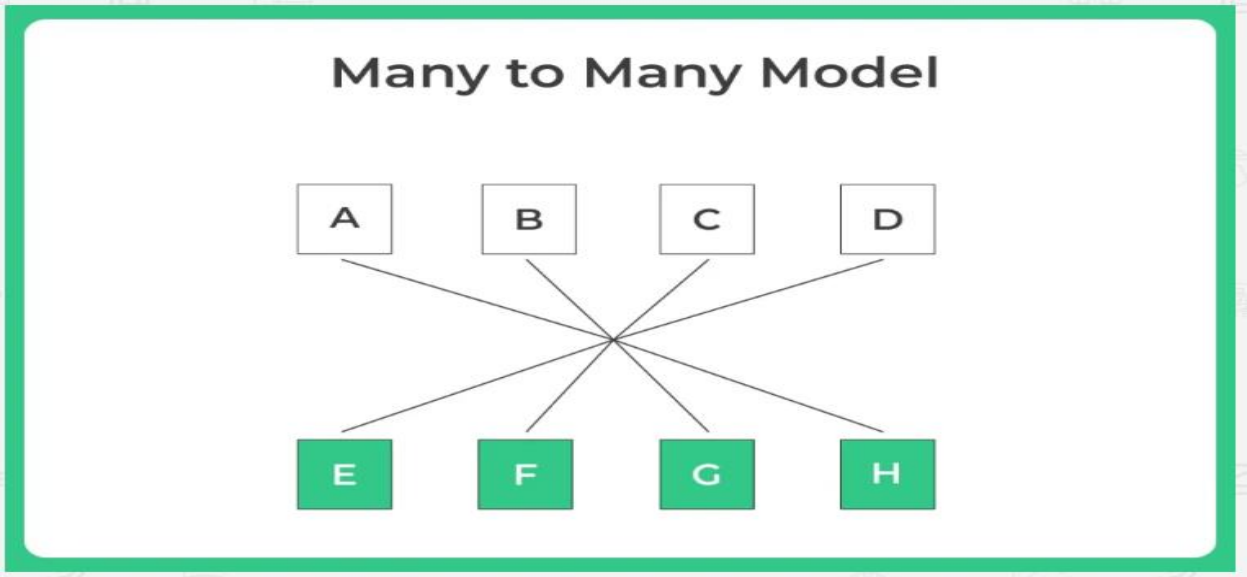
One To One Model

- Each of User thread is mapped to kernel thread.
- Provides more concurrency.
- Whole process does not gets blocked when 1 thread makes a blocking call as other threads can run.
- Creates Overhead as every User thread requires to create the corresponding Kernel Thread for it.
- Supports a particular number of thread only, cause of the overhead caused due to creation of Kernel Level Thread.
- Implemented By Linux and Windows Operating System.



Many to Many Model

- Multiple User Threads are multiplexed to same or less number of Kernel threads.
- This is the best among the three models, as it improves on shortcoming of one to one model and many to one model.
- We can create as number of User Threads as we want, unlike One to One Model.
- There is no issue of process blocking and threads can run parallelly in multiprocess system unlike Many To One Model.



Difference Between Program and Process in Operating System

Program

In simple words program is a passive entity and a set of instructions designed to perform some certain task.

Process

A process is an executing instance of a program. Its an active entity. We can also say process is an program in execution currently.

Program

- A **program is** a system activity that uses a set of instructions to perform a designated task.
- It is considered a passive entity as it resides on secondary memory.
- The resource requirement is less as it only requires memory for storage.
- The program performs tasks that are directly related to the operations of a user.

For example, notepad.exe is an executable file having a set of instructions that helps users to print and edit text files. When the user executes the program, it converts into the process.

Process

- A **process** is an executing instance of a program.
- In other words, the process is a program in execution
- A process is entirely dependent on the program
- It is considered an active entity as it resides on the primary memory. (RAM)
- It requires resources like processing, memory address, CPU, input and output resources to perform the task.

For example, a user can run multiple instances of Microsoft Word program.

Deadlock in Operating System (OS)

Operating System Deadlock

A deadlock is a situation in an operating system where two or more processes are unable to proceed because each is waiting for one of the others to do something. This can occur when multiple processes request and are allocated resources in a way that results in a cycle of dependencies, such that none of the processes can proceed. Deadlock in Operating System can be prevented or resolved by using techniques such as resource allocation algorithms, deadlock detection, and recovery schemes.

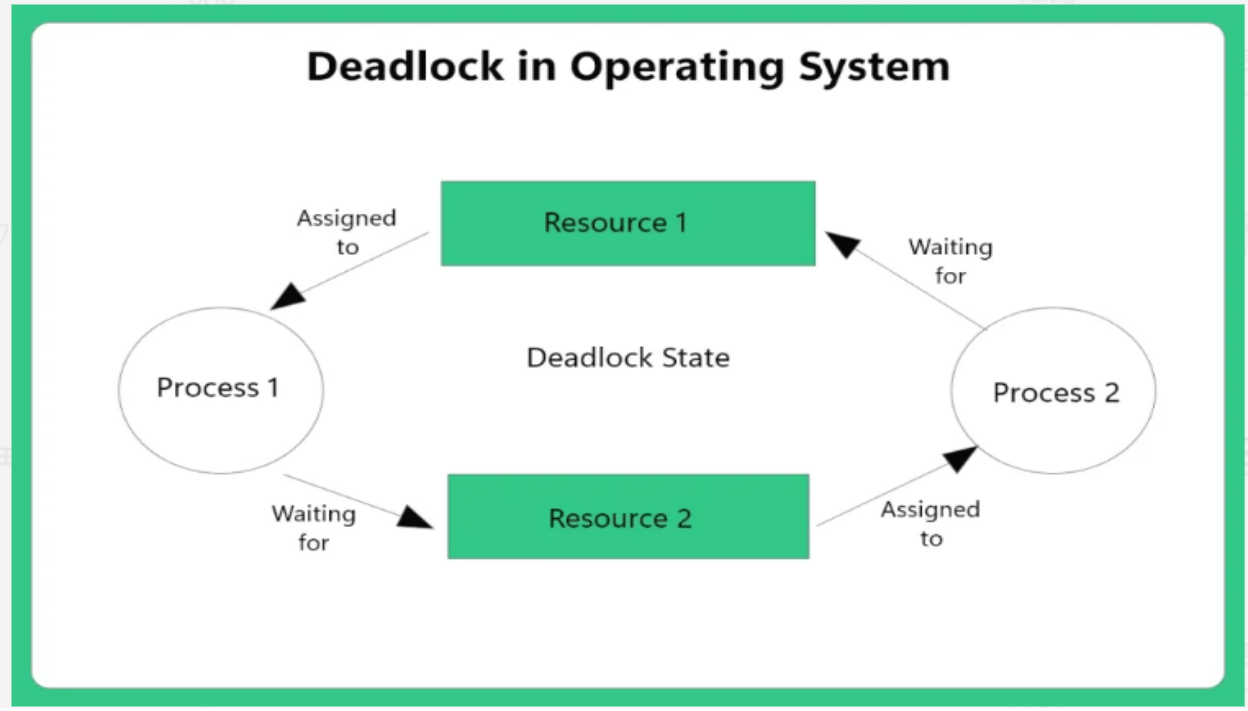
Deadlock in Operating System

Deadlock refers to the condition when 2 or more processes are waiting for each other to release a resource indefinitely. A process in nature requests a resource first and uses it and finally releases it.

But in deadlock situation, both the processes wait for the other process. Let's look at one example to understand it – Say, Process A has resource R1 , Process B has resource R2. If Process A request resource R2 and Process B requests resource R1, at the same time , then deadlock occurs.

Important Features of Deadlock –

- In deadlock, processes get blocked because each process is holding some resource and they are waiting for other resource , which is held by another process.
- Deadlock usually occurs in multiprocessing, time- sharing etc
- In Multiprocessing, many processes share a specific type of mutually exclusion resource known as soft lock.
- Time Sharing computers are equipped with a hardwired lock which gaurantees exclusive access to processes , thus preventing deadlock.
- There is no general solution to avoid soft deadlock.



Conditions for Deadlock

Deadlock can occur if all the given 4 conditions (Coffman Conditions) are satisfied, that is if all are true :

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular wait

Some important features of deadlock include:

1. Mutual exclusion: Resources are held in a non-shareable mode, meaning that only one process can use a resource at a time.
2. Hold and wait: A process is holding at least one resource and waiting for additional resources that are currently being held by other processes.
3. No preemption: Resources cannot be taken away from a process once they have been acquired.
4. Circular wait: Two or more processes are waiting for each other to release resources, resulting in a cycle of waiting.
5. Deadlock can occur in a variety of computing environments, including distributed systems, databases, and operating systems.

6. Deadlock can be resolved through various methods such as prevention, detection, and recovery.

Deadlock Detection And Recovery in OS

Detection and Recovery of Deadlock in Operating System

Deadlock detection and recovery refer to the process of identifying and resolving instances of deadlock in a system. Deadlock detection involves identifying when a deadlock has occurred and recovery involves resolving the deadlock.Deadlock detection involves identifying when a deadlock has occurred in the system. Recovery can be once a deadlock has been detected, it must be resolved.

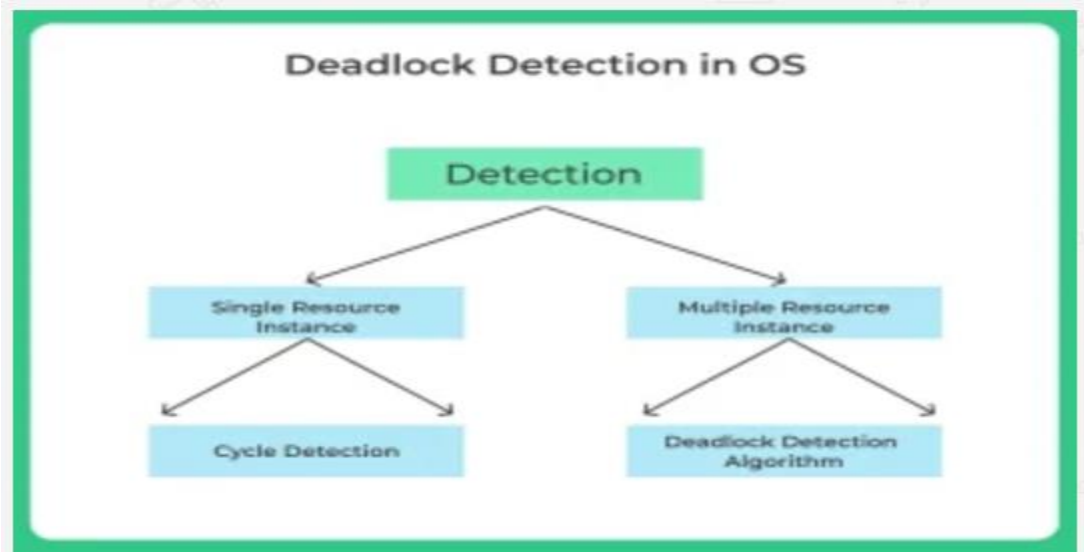
Deadlock in Operating System

Deadlock detection involves identifying when a deadlock has occurred in the system. This can be done through the use of algorithms, such as the Banker's Algorithm or the Wait-For Graph Algorithm, which check for the presence of the necessary conditions for deadlock.

Detection

There are 2 different cases in case of Deadlock detection –

1. Deadlock detection using a centralized algorithm: In this approach, a single entity, such as the operating system, is responsible for detecting and resolving deadlocks in the system. This entity periodically checks for the presence of deadlocks and takes appropriate action to resolve them.
2. Deadlock detection using a distributed algorithm: In this approach, multiple entities, such as processes or nodes in a distributed system, work together to detect and resolve deadlocks. Each entity maintains information about the resources it holds and requests, and communicates with other entities to detect and resolve deadlocks. This approach can be more efficient and scalable than a centralized approach, but it can also be more complex to implement.



Both centralized and distributed algorithms have their own advantages and disadvantages. Centralized algorithms are simpler to implement but may have scalability issues, while distributed algorithms can be more efficient and handle large systems, but they can be more complex to implement.

If Resources has Single Instance

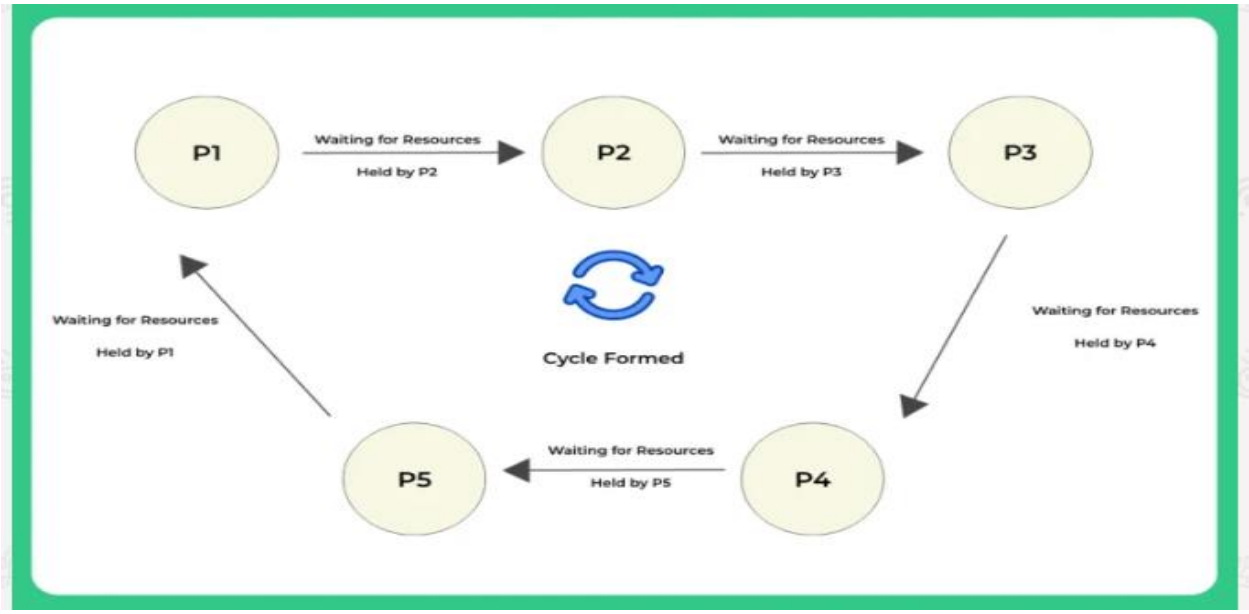
If there is only one instance of a resource in a system, deadlock detection and recovery is not necessary. This is because there can be no situation where multiple processes are competing for the same resource, and therefore no possibility of a deadlock occurring.

However, if there are multiple instances of a resource, then deadlock detection and recovery is necessary to prevent and resolve any potential deadlocks.

The **Wait-for graph** is made by looking at the resource-allocation graph. An edge between P1 to P2 exists if P1 needs some resources which P2 has.

- In the above wait-for graph P1 is waiting for resources currently in use by P2
- And P2 is waiting for resources currently in use by P3 and so on…
- P5 is waiting for a resource currently in use by P1. Which creates a cycle thus deadlock is confirmed.

Wait-for-graphA wait-for graph is made. A Wait-for graph vertex denotes process. The edge implies process waiting for Process 2 to release resources. A deadlock is detected if one wait-for graph contains a cycle.



To detect cycle, system maintains the wait state of graph and periodically invoke an algorithm to detect cycle in graph.

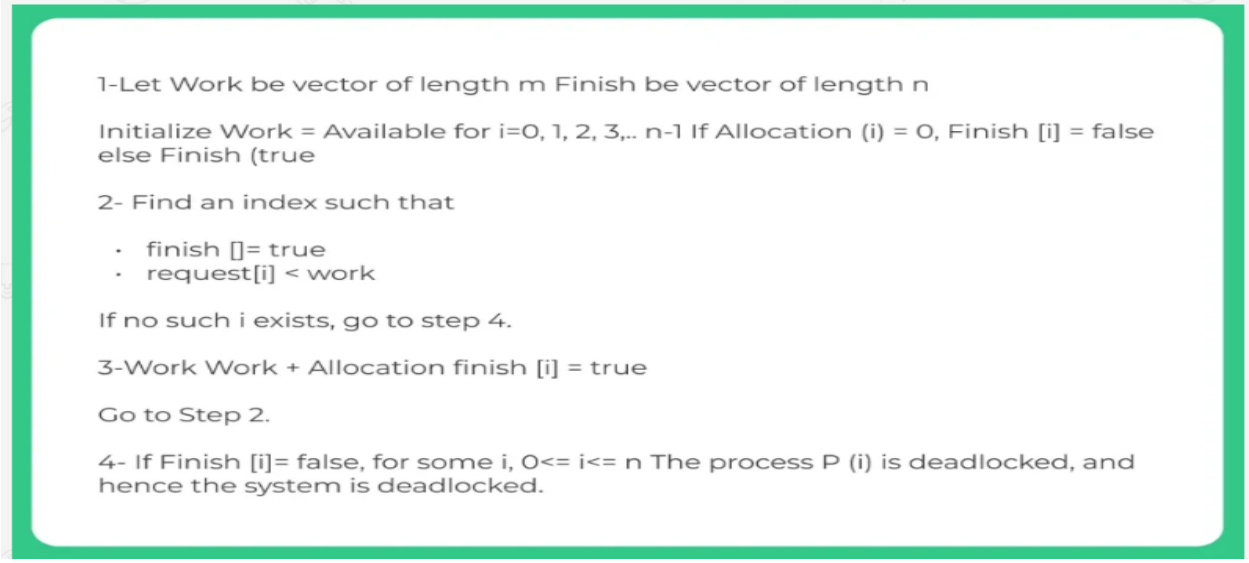
If there are multiple instances of resources –
By multiple instance we mean one resource may allow 2 or more accesses concurrently, in such cases –

- Wait for graph is not applicable.
- In this, a new algorithm is used. It's a bit similar to Banker's Algorithm, but Bankers Algorithm is different.
- It too uses 3 data structures –
    - Available
        - Vector of length m
        - Indicates number of available resources of each type.
    - Allocation
        - Matrix of size n*m
        - A[i,j] indicates the number of j th resource type allocated to i th process.
    - Request

- Matrix of size n*m
- Indicates request of each process.
- Request[i,j] tells number of instance Pi process is request of jth resource type.

The Algorithm for detection is –

Check this page here to learn how to solve Deadlock Detection Algorithm.



Time Complexity is O(m*n*n).

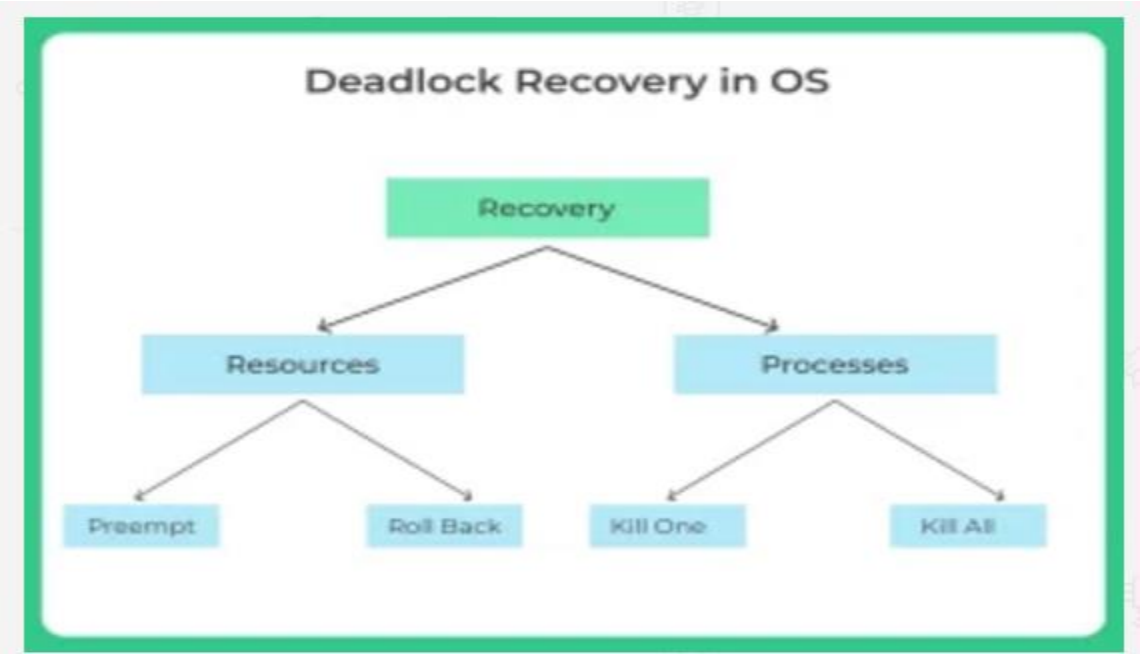
Deadlock Recovery

Once a deadlock has been detected, it must be resolved. There are several methods for deadlock recovery, including:

- Killing one or more processes: This is known as the “abort” method, where the operating system kills one or more of the processes involved in the deadlock in order to release the resources and resolve the deadlock.
- Resource preemption: This method involves forcibly taking resources from a process and giving them to another process in order to resolve the deadlock.
- Resource allocation: This method involves changing the way resources are allocated in order to prevent deadlocks from occurring in the future.

It's worth noting that, besides the above mentioned methods, there's another one called "Rollback" which involves rolling back one or more processes to a previous state in order to release resources and resolve the deadlock.

It's important to note that any chosen method should be used with care, as it may cause other issues or problems.



Deadlock Detection Algorithm in Operating System

Deadlock Detection Algorithm in Operating System

Deadlock Detection Algorithm helps decide if in scenario of multi instance resources for various processes are in deadlock or not.In cases of single resource instance we can create wait-for graph to check deadlock state. But, this we can't do for multi instance resources system.

Algorithm Example

Do not confuse the deadlock detection algorithm with Banker's algorithm which is completely different. Learn Banker's Algorithm here.

The deadlock detection algorithm uses 3 data structures –

- Available
    - Vector of length m

- Indicates number of available resources of each type.

- Allocation
  - Matrix of size n*m
  - A[i,j] indicates the number of j th resource type allocated to i th process.
- Request
  - Matrix of size n*m
  - Indicates request of each process.
  - Request[i,j] tells number of instance Pi process is request of jth resource type.

Steps

- **Step 1**

1. Let **Work(vector)** length = m
2. **Finish(vector)** length = n
3. Initialize Work= Available.
   1. For i=0, 1, …., n-1, if $Allocation_i$ = 0, then Finish[i] = true; otherwise, Finish[i]= false.

- **Step 2**

1. Find an index i such that both
   1. Finish[i] == false
   2. $Request_i$ <= Work

If no such i exists go to step 4.

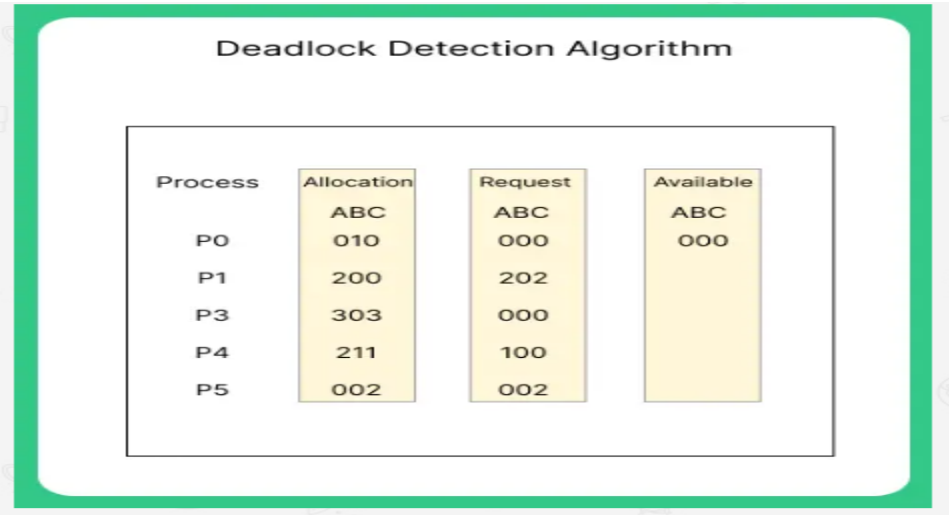
- **Step 3**

1. Work= Work+ $Allocation_i$
2. Finish[i]= true

Go to Step 2.

- **Step 4**

If Finish[i]== false for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if Finish[i]==false the process P_i is deadlocked.



Deadlock Avoidance Results

- **Step 1**

In this, Work =[0,0,0] &
Finish = [false, false, false, false, false].

- **Step 2**

i=0 is selected as both Finish[0]=false and [0,0,0] <=[0,0,0].

- **Step 3**

Work = [0,0,0]+[0,1,0]=>[0,1,0] &
Finish = [true, false, false, false, false].

- **Step 4**

i=2 is selected as both Finish[2] = false and [0,0,0]<=[0,1,0].

- **Step 5**

Work = [0,1,0]+[3,0,3] = > [3,1,3] &
Finish = [true, false, false, false, false].

- **Step 6**

i=1 is selected as both Finish[1] = false and [2,0,2] <=[3,1,3].

- **Step 7**

Work=[3,1,3]+[2,0,0] => [5,1,3] &
Finish = [true, true, true, false, false].

- **Step 8**

i=3 is selected as both Finish[3] =false and [1,0,0] <= [5,1,3].

- **Step 9**

Work = [5,1,3] + [2,1,1] =>[7,2,4] &
Finish = [true, true, true, true, false].

- **Step 10**

i=4 is selected as both Finish[4] = false and [0,0,2] <=[7,2,4].

- **Step 11**

Work = [7,2,4] + [0,0,2] =>[7,2,6] &
Finish = [true, true, true, true, true].

Since Finish is a vector of all true it means there is no deadlock in this example.

Deadlock Avoidance and Prevention in OS

OS Avoidance and Prevention Deadlock

Deadlock avoidance and prevention in OS are techniques used in operating systems to prevent or avoid situations where two or more processes are unable to proceed because each is waiting for one of the others to release a resource.

Deadlock Prevention in OS

Deadlock avoidance uses dynamic information, such as the current state of the system and the resource allocation, to ensure that a deadlock never occurs. Deadlock prevention uses static

information, such as the maximum number of resources a process can request, to prevent a deadlock from occurring. Resource allocation algorithms such as the Banker's Algorithm can be used for deadlock prevention.

Deadlock prevention is a technique used in operating systems to avoid the situation where two or more processes are unable to proceed because each is waiting for one of the others to release a resource. This can be achieved by imposing a set of constraints on the way resources are requested and released by the processes.

Some common methods of deadlock prevention include:

1. **Resource allocation:** By allocating resources in a way that ensures that a process can never enter a deadlock state, such as using the Banker's Algorithm, which checks for safe states before allocating resources.
2. **Limit on resources:** By limiting the number of resources a process can request, a deadlock state can be prevented from occurring.
3. **Timeout:** By setting a timeout for resource requests, a process will release a resource if it is unable to acquire it within a certain time period.
4. **Priority-based resource allocation:** By allocating resources based on the priority of the process, a deadlock state can be prevented from occurring.
5. **Preemptive resource allocation:** By preempting resources from a process that has been holding them for a long time, a deadlock state can be prevented from occurring.

These are some of the common methods used in operating systems to prevent deadlocks. The best method to use depends on the specific requirements of the system and the resources being used.

Deadlock avoidance in OS

Deadlock avoidance is a technique used in operating systems to prevent the situation where two or more processes are unable to proceed because each is waiting for one of the others to release a resource. Unlike deadlock prevention, which uses static information to prevent a deadlock from occurring, deadlock avoidance uses dynamic information, such as the current state of the system and the resource allocation, to ensure that a deadlock never occurs. Some common methods of deadlock avoidance include:

1. **Wait/Die:** In this method, a process that requests a resource that is not available will wait until the resource becomes available. If a process that holds the resource is also requesting a resource that is not available, the process that has been waiting the longest will be allowed to proceed.

2. **Wound/Wait:** In this method, a process that requests a resource that is not available will be killed (wounded) if a process that holds the resource is also requesting a resource that is not available. The process that was killed will have to request the resource again later.
3. **Resource allocation graph:** By representing the resources and the processes as nodes in a graph, the operating system can use algorithms to check for safe states before allocating resources.
4. **Banker's Algorithm:** This algorithm is a variant of Resource Allocation Graph algorithm, it uses the available and maximum resource information for each process, the algorithm checks for safe state before allocating resources.
5. **Time-stamp ordering:** In this method, a process is only allowed to request a resource if its time stamp is greater than the time stamp of the process that holds the resource.

These are some of the common methods used in operating systems for deadlock avoidance. The best method to use depends on the specific requirements of the system and the resources being used.

Bankers Algorithm in Operating System (OS)

Operating System – Bankers Algorithm

The Banker's Algorithm in Operating System is a method used in operating systems to ensure that a system does not experience deadlock or starvation. The algorithm is used to determine the maximum number of resources that can be allocated to each process while still ensuring that the system remains in a safe state. The algorithm keeps track of the maximum number of resources that each process may need and the number of resources that are currently available.

What is Bankers Algorithm?

If a process requests a resource, the algorithm checks to see if the resources are available and, if so, allocates them to the process. If the resources are not available, the algorithm checks to see if they can be freed up by other processes and, if so, allocates them. The algorithm continues to check and allocate resources until the system is in a safe state or until all processes have been allocated the resources they need.

Now, Let's have a look at how these are implemented . They are **implemented using 4 data structure.**

Let n be the number of process in system.

Let m be no of resources in system.

- **Available**
  - 1 D Array of Size m.
  - Each element Available[i] indicates the number of resources of i type available. Denoted by R_i
- **Maximum**
  - 2 D Array of size n*m
  - Determines maximum demand of each process.
  - Maximum[i,j] tells the maximum demand an ith process will request of jth type resource.
- **Allocation**
  - 2 D Array of size n*m
  - Defines number of resources of each type currently allocated to each process.
  - Allocation[i,j] means i th process has current k instance of j th type resource
- **Need**
  - 2D Array of size n*m
  - Tells about remaining resources type which are required by each process.
  - Need[i,j] means i th process needs k instance of j th resource type.
  - Need[i,j] = Maximum[i,j] – Allocation[i,j]

What is Bankers Algorithm in Operating System?

Banker's Algorithm is a resource allocation and deadlock avoidance algorithm. This name has been given since it is one of most problem in Banking Systems these days.

- In this, as a new process P1 enters, it declares the maximum number of resource it needs.
  - The system looks at those and checks if allocating those resources to P1 will leave system in safe state or not.
  - If after allocation, it will be in safe state , the resources are allocated to process P1.
- Otherwise, P1 should wait till other process release some resource.

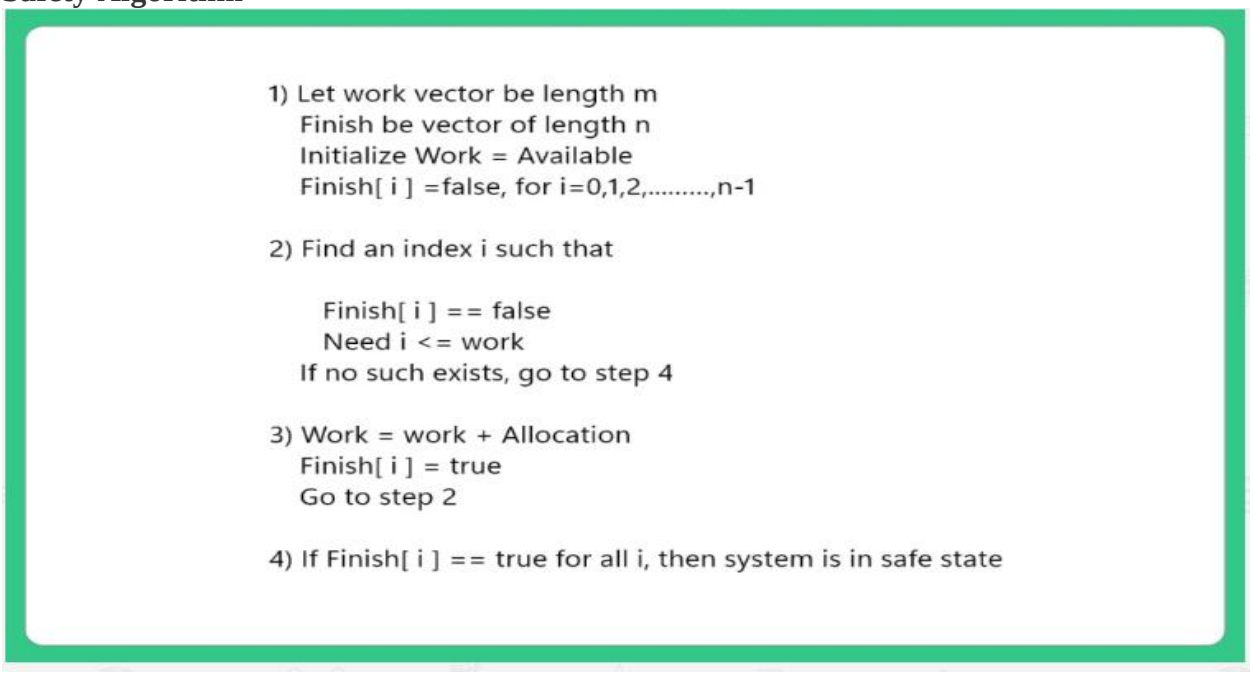
This is the basic idea of Banker's Algorithm.

A state is ***safe*** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state**.**

The above data Structures Size and value vary over time. There are 2 Algorithm on which banker's Algorithm is build upon

- Safety Algorithm : It finds out whether system is in safe state or not.
- Resource Request Algorithm : It finds out if request can be safely granted.

Safety Algorithm –

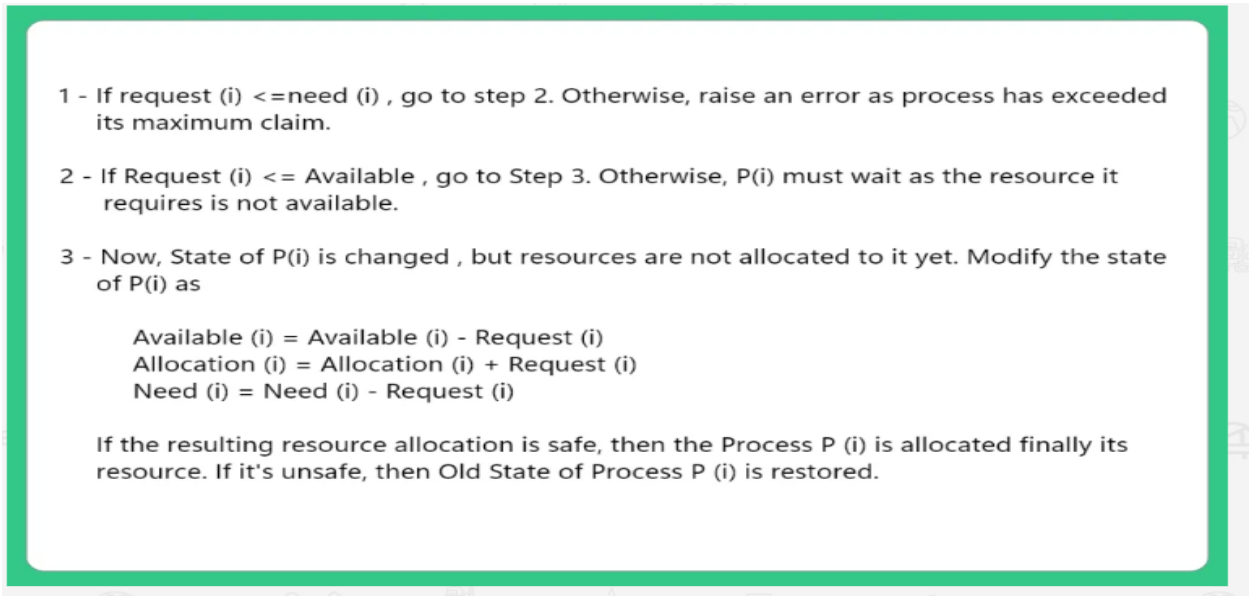


Time Complexity of Above Algorithm = O(m*n*n)

Resource Request Algorithm

- Tells if Resource can be safely Granted.
- Request i is Request for Process i.

After a process makes a request, The following steps are followed-



Memory Management in Operating System (OS)

Memory Management in Operating System

Memory management is the process of intelligently handling all the memory related operations and resources in the primary memory or storage disk when there are multiple processes that are using memory and resources. It keeps track of all the memory, it may be idle in the system or can be allocated to different processes at a given time.

- It decides which processes will get memory resources and when
- It keeps track of all memory status, when allocated or when idle

Various Methods Involved

There are various methods through which the memory management system in OS intelligently manage the system –

We will discuss all these in detail in the upcoming posts of the series.

- Partition Allocation Method
    - Create page
    - First Fit
    - Best fit
    - Worst fit
    - Next fit
- Fragmentation
    - Internal
    - External
- Paging
- Segmentation
- Buddy-System Allocator
- Page Replacement Algorithms
    - LRU

  - FIFO
  - Optimal Page Replacement algorithm
- Thrashing
- Translation lookaside buffer

Entities involved –

- Static vs Dynamic Loading
- Static vs Dynamic Linking
- Swapping
- Process Address Space
- Mapping Virtual address to Physical Address
- Virtual Memory

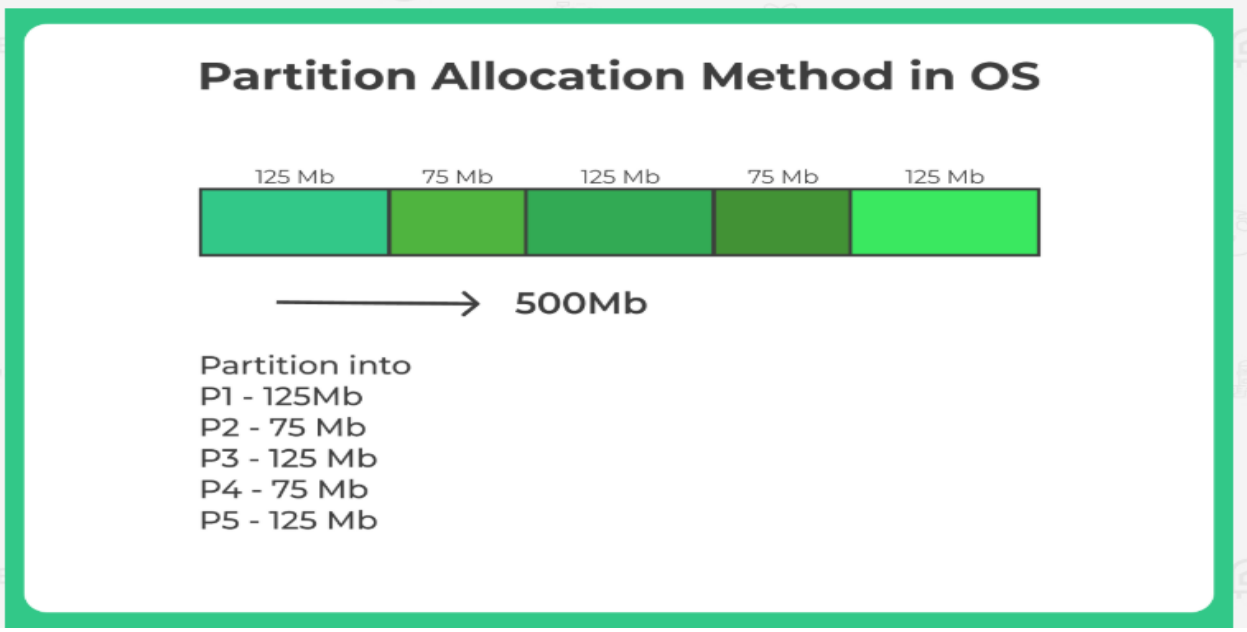
Partition Allocation Method in Operating System

Partition Allocation Methods in OS

The memory systems developed in early 1990 had a **contiguous(continuous) memory available** for storing different files one example of such traditional OSes is MS-Dos.

But, new systems that have been developed **do not have a contiguous** memory space available for files. In fact they are divided into blocks of different sizes.

This was done to avoid complete memory corruption if a file is infected. If memory is divided into blocks corrupted file will only infect that memory block in which its stored.



Example of Memory Partitioning

For example, a 500Mb memory may be (equally or unequally) divided to blocks 5 blocks of 100Mb each.

Or unequally as in 5 blocks of 125mb, 75mb, 125mb, 75mb, 125mb respectively.

The main memory must accommodate –

1. Operating system
2. User processes
3. Operating System Processes

For a partitioned memory available in modern systems, we need different algorithms to decide most which processes must use which memory block while maintaining the most efficient use of these memory block.

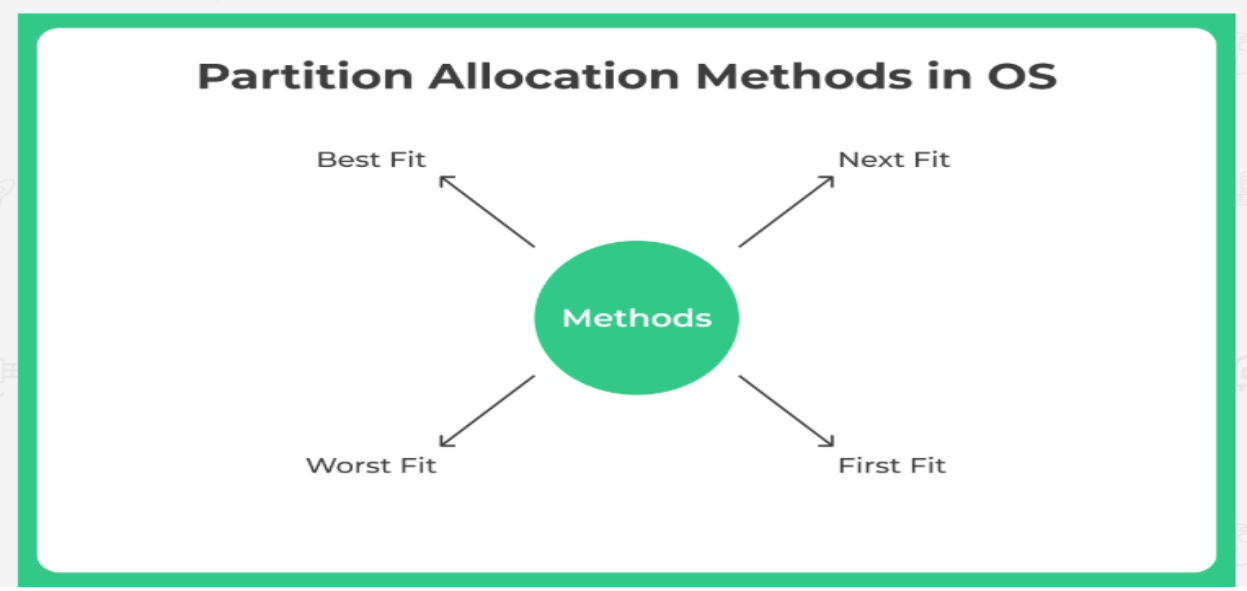
We therefore need to allocate different parts of the main memory in the most efficient way possible.

Memory Partitioning Algorithms

The main four partition allocation schemes are listed below:-

- First Fit
- Best Fit
- Worst Fit

- Next Fit



Overview

There are two major divisions one for operating system residing and other for allocating processes.

- OS memory can't be altered in a direct by way by user.
- Process memory is managed by OS to perform processes.

One of the simplest methods for memory allocation

1. Divide memory into several fixed-sized partitions.
2. Each partition may contain exactly one process.
3. When a partition is free, a process is selected from the input queue and is loaded into the free partition.
4. When the process terminates, the partition becomes available for another process.
5. The operating system keeps a table indicating which parts of memory are available and which are occupied using a table.
6. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided.

The following are used to choose process memory allocation –

- First Fit
- Best Fit
- Worst Fit
- Next Fit

1. First Fit

- **Mechanism**: The system scans the memory from the beginning and allocates the first available block that is large enough for the process.
- **Advantages**:
  - Simple and fast as it stops searching as soon as a suitable block is found.
  - Reduces search time compared to other strategies.
- **Disadvantages**:
  - Can lead to fragmentation since smaller unused blocks may accumulate near the beginning of memory.
- **Example**:
  - Available blocks: [100 KB, 500 KB, 200 KB, 300 KB]
  - Process size: 150 KB
  - Allocates the **200 KB block** (first suitable block).

---

2. Best Fit

- **Mechanism**: The system searches the entire memory to find the smallest block that is large enough for the process, minimizing leftover space.
- **Advantages**:
  - Minimizes memory wastage by utilizing blocks more efficiently.
- **Disadvantages**:
  - Slower than First Fit because it must search the entire memory.
  - May increase fragmentation as small gaps accumulate.
- **Example**:
  - Available blocks: [100 KB, 500 KB, 200 KB, 300 KB]
  - Process size: 150 KB
  - Allocates the **200 KB block** (smallest suitable block).

---

3. Worst Fit

- **Mechanism**: The system allocates the largest available memory block to the process, ensuring that the remaining block is large enough for future allocations.
- **Advantages**:
  - May reduce fragmentation in some cases by leaving larger free blocks available.
- **Disadvantages**:
  - Wastes large memory blocks, leading to inefficient utilization.
  - Slower than First Fit due to searching for the largest block.
- **Example**:
  - Available blocks: [100 KB, 500 KB, 200 KB, 300 KB]
  - Process size: 150 KB
  - Allocates the **500 KB block** (largest block).

4. Next Fit

- **Mechanism**: Similar to First Fit, but instead of starting the search from the beginning of memory, it resumes from the last allocated block and wraps around to the beginning if necessary.
- **Advantages**:
  - Distributes allocation across the memory, potentially reducing fragmentation in specific areas.
  - Faster than Best Fit and Worst Fit.
- **Disadvantages**:
  - Can still lead to fragmentation, especially if memory usage patterns are uneven.
- **Example**:
  - Available blocks: [100 KB, 500 KB, 200 KB, 300 KB]
  - Process size: 150 KB
  - If the previous allocation ended at the 500 KB block, it starts scanning from the **200 KB block**.

First Fit Algorithm in OS

First Fit in Operating System

The operating system uses different memory management schemes to optimize memory/resource block allocation to different processes. We will look at one of such memory allocation processes in OS called First Fit in Operating System. In this approach we allot the first available space that is large enough to accommodate the process.

Different types of Memory Allocations in OS?

The four most commonly used allocation schemes are

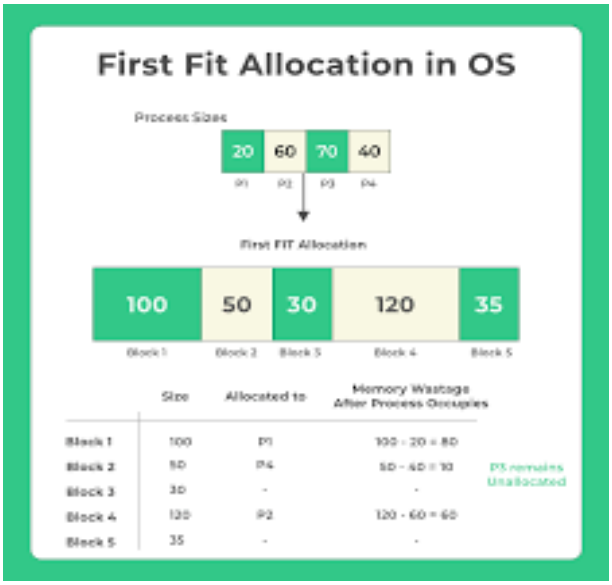
- First Fit
- Best Fit
- Worst Fit
- Next Fit

How first fit works?

Whenever a process (p1) comes with memory allocation request the following happens –

- OS sequentially searches available memory blocks from the first index
- Assigns the first memory block large enough to accommodate process

Whenever a new process P2 comes, it does the same thing. Search from the first index again.



Working Example for First Fit

Example: As shown on the right/below image

Memory blocks available are : {100, 50, 30, 120, 35}

Process P1, size: 20

- OS Searches memory from sequentially from starting
- Block 1 fits, P1 gets assigned to block 1

Process P2, size: 60

- OS Searches memory sequentially from block 1 again
- Block 1 is unavailable, Block 2 and 3 can’t fit
- Block 4 fits, p2 assigned to block 4

Process P3, size: 70

- OS Searches memory sequentially from block 1 again
- Block 1 is unavailable, Block 2, 3 can’t fit. Block 4 unavailable, Block 5 can’t fit
- P3 remains unallocated

Similarly, P4 is assigned to block 2

Advantages

- Easy to implement
- OS can allocate processes quickly as algorithm to allocate processes will be quick as compared to other methods (Best Fit, Worst Fit, Next Fit etc)

Disadvantages

- Causes huge internal fragmentation
- Smarter allocation may be done by best-fit algorithm
- High chances of unallocated for some processes due to poor algorithm
- More overhead as compared to next fit

Best Fit Algorithm in Operating System

Best Fit Algorithm in Operating System

In the case of the best fit memory allocation scheme, the operating system searches for the empty memory block. When the operating system finds the memory block with minimum wastage of memory, it is allocated to the process this is known as Best Fit Algorithm in Operating System.

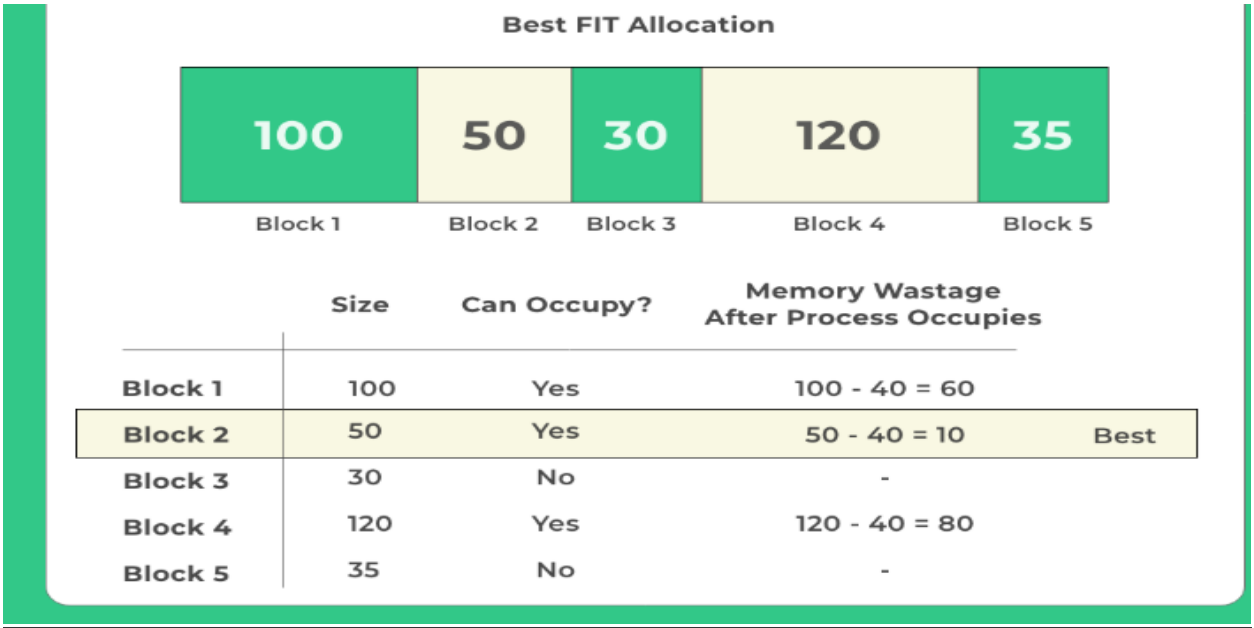
How does it work?

In Best fit memory allocation scheme, the operating system searches that can –

- Accommodate the process
- Also, leaves the minimum memory wastage

Example –

If you see the image below/right you will see that the process size is 40.While blocks 1, 2 and 4 can accommodate the process. **Block 2 is chosen as it leaves the lowest memory wastage**



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100	Yes	100 - 40 = 60	
Block 2	50	Yes	50 - 40 = 10	Best
Block 3	30	No	-	
Block 4	120	Yes	120 - 40 = 80	
Block 5	35	No	-	

Advantage:

It is allocated to the process. This scheme is considered the best approach as it results in the most optimized memory allocation.

Also reduces internal fragmentation.

Disadvantage:

However, finding the best fit memory allocation may be time-consuming.

Worst Fit Algorithm in Operating System

Worst Fit Algorithm in Operating System

The processes need empty memory slots during processing time. This memory is allocated to the processes by the operating system which decides depending on the free memory and the demanded memory by the process in execution. The three most common memory allocation schemes are first fit, best fit, and worst fit algorithm in Operating System.

How Worst Fit Works?

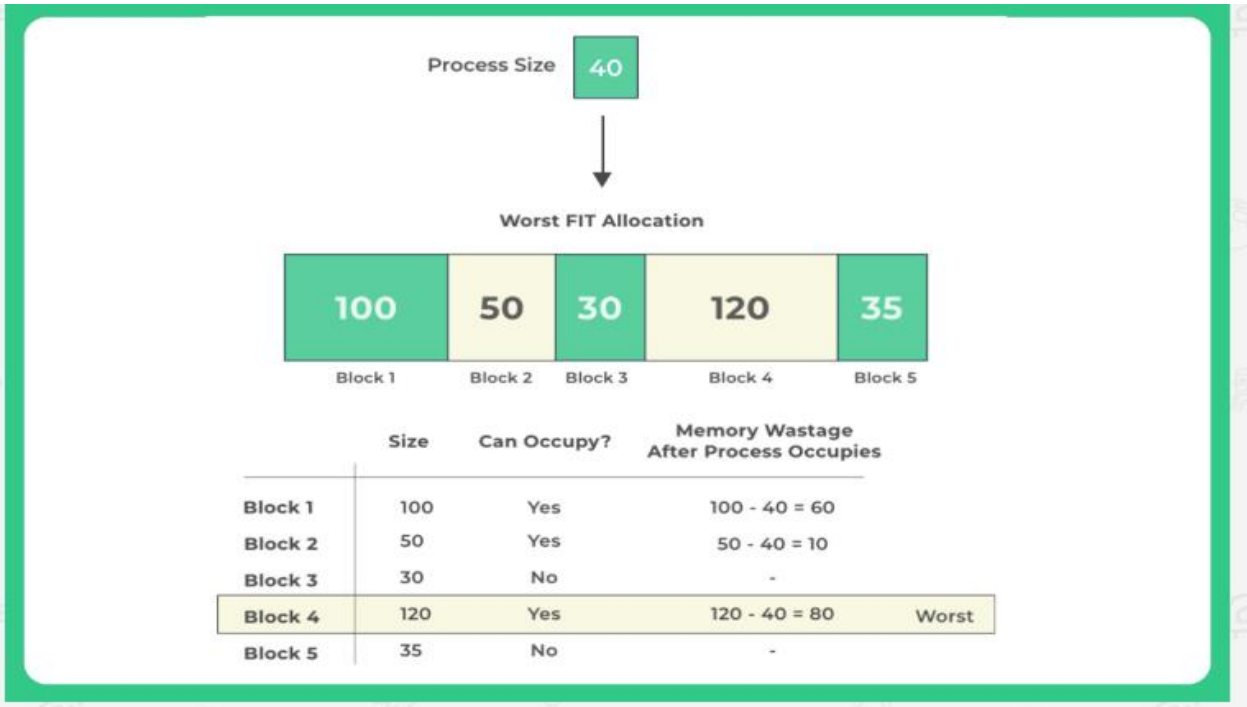
Worst fit works in the following way, for any given process P_n.

The algorithms searches sequentially starting from first memory block and searches for the memory block that fulfills the following condition –

- Can accommodate the process size

- Leaves the largest wasted space (fragmentation) after the process is allocated to given memory block

Worst Fit Allocation in OS



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100	Yes	100 - 40 = 60	
Block 2	50	Yes	50 - 40 = 10	
Block 3	30	No	-	
Block 4	120	Yes	120 - 40 = 80	Worst
Block 5	35	No	-	

Algorithm for Worst Fit Memory Management Scheme

- **Step 1:** Input memory block with a size.
- **Step 2:** Input process with size.
- **Step 3:** Initialize by selecting each process to find the maximum block size that can be assigned to the current process.
- **Step 4:** If the condition does not fulfill, they leave the process.
- **Step 5:** If the condition is not fulfilled, then leave the process and check for the next process.
- **Step 6:** Stop.

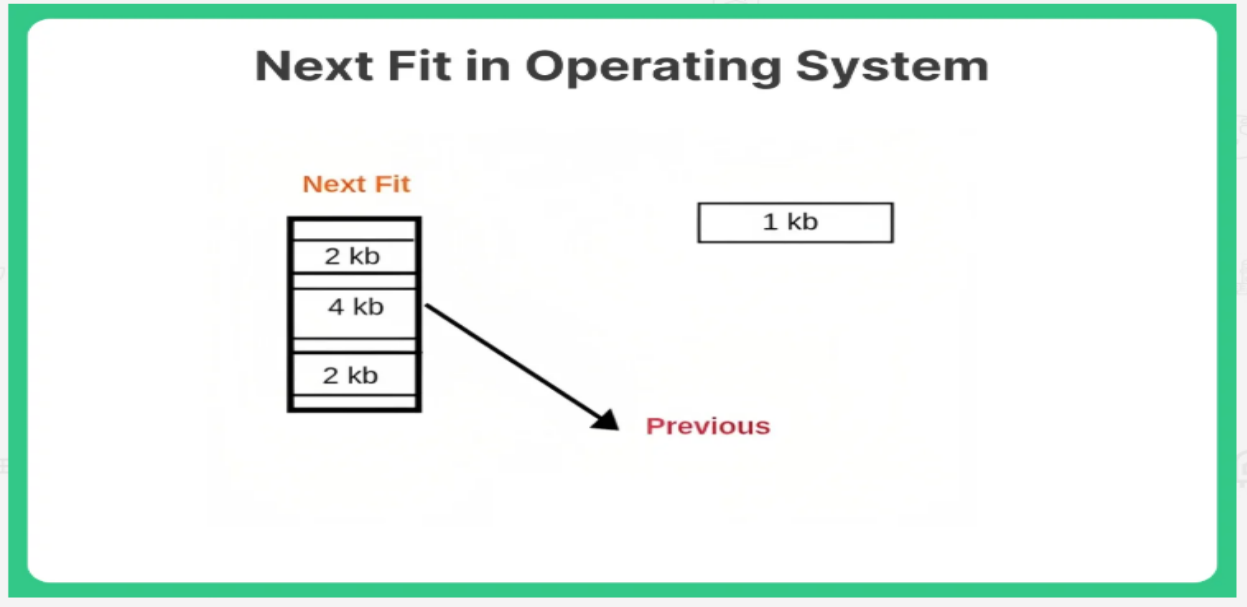
Next Fit Algorithm in Operating System (OS)

Next Fit Algorithm in Operating System

Next fit is another version of First Fit in which memory is searched for empty spaces similar to the first fit memory allocation scheme. Unlike first-fit memory allocation, the only difference between the two is, in the case of next fit, if the search is interrupted in between, the new search is carried out from the last location.

Next fit is another version of First Fit in which memory is searched for empty spaces similar to the first fit memory allocation scheme. Unlike first-fit memory allocation, the only difference between the two is, in the case of next fit, if the search is interrupted in between, the new search is carried out from the last location.

Next fit can also be said as the modified version of the first fit as it starts searching for a free memory as following the first-fit memory allocation scheme. This memory allocation scheme uses a moving pointer which moves along the empty memory slots to search memory for the next fit. The next fit memory allocation method avoids memory allocation always from the beginning of the memory space. The operating system uses a memory allocation algorithm, also known as the scheduling algorithm for this purpose.



Algorithm for memory allocation using Next Fit

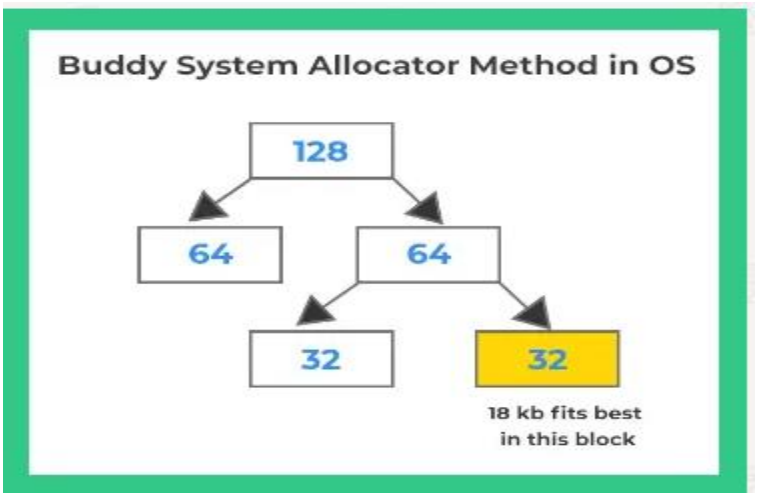
- **Step 1**. Enter the number of memory blocks.
- **Step 2**. Enter the size of each memory block.
- **Step 3**. Enter the number of processes with their sizes.

- **Step 4**. Start by selecting each process to check if it can be assigned to the current memory block.
- **Step 5**. If the condition in step 4 is true, then allocate the process with the required memory and check for the next process from the memory block where the searching was halted, not from the starting.
- **Step 6**. If the current memory size is smaller, then continue to check the next blocks.
- **Step 7**. Stop

Buddy System Allocator in Operating System

Why Buddy System Allocator?

A poor match between available partition sizes and process sizes. A dynamic partitioning scheme is more complex to maintain and includes the overhead of compaction. An interesting solution is the buddy system.



Buddy System: How it works

In a buddy system, the entire memory space available for allocation is treated as a single block whose size is a power of 2.

Suppose that the size of memory is 2U and the requirement is of size S.

Allocation can happen if –

1. **If $2^{U-1}<S<=2^{U}$:** is satisfied else

2. Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block and get out the loop.

Advantage

1. Faster allocation of memory and even faster deallocation
2. More useful.

Using the above method itself, if we assume that physical address space is of 128KB and we need the allocation for 18KB partition. Then it will fall into the 32KB partition. Check the image above to understand the same.

Linux also uses buddy system to allocate the memory.

Disadvantage

The disadvantage of this approach is that it wastes a lot of memory in internal fragmentation since all the blocks are rounded of to the size of two.

Paging in Operating System

What is Paging?

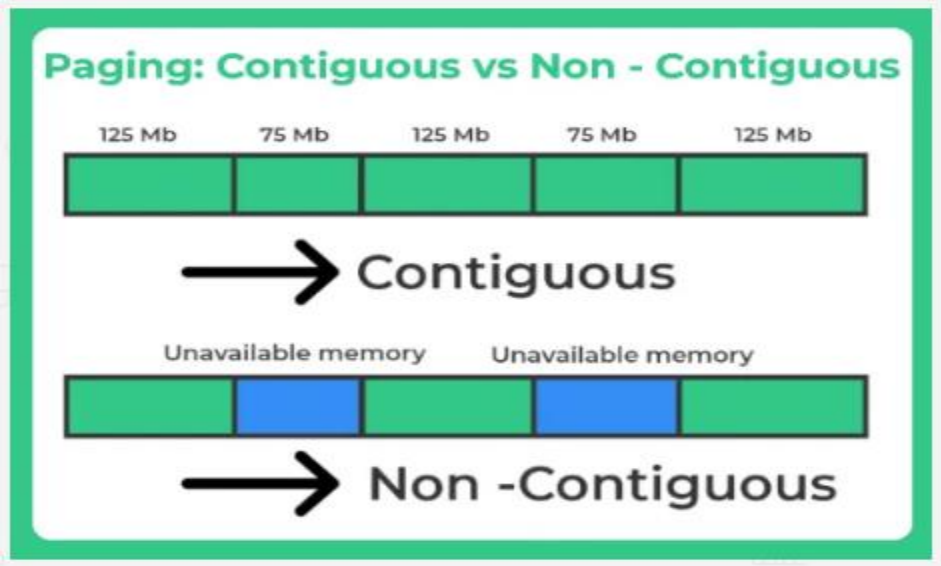
Paging is a memory-management scheme that permits the physical address space of a process to be noncontiguous or in other words eliminates the need for contiguous allocation of physical memory.

That is we can have logically use memory spaces that physically lie at different locations in the memory

This allows viewing memory spaces that physically lie at different locations in the hardware to be logically viewed as contiguous.

This is possible via mapping of physical and virtual memory that is done by hardware component called memory management unit (MMU). The mapping process that is done by MMU is basically paging process.

Let’s see how paging process is implemented.



Prerequisite
Before this its important to know a few terminologies –

This will also be useful –

1. 1 byte = 2^3 bits
2. 1KB = 2^10 bytes
3. 1MB = 2^20 bytes
4. 1GB = 2^30 bytes

Logical View

- Logical Space Available – The total space available in logical system
- Logical Address – This is also known as Virtual Address and is directly generated by the CPU of the system.
- **Conversion Examples –**
    - If Logical address is 21 bit then -> Logical Address space = 2^{21} which is $2 * 2^{20}$. Thus 2M words
    - If Logical address is 23 bit then -> Logical Address space = 2^{23} which is $2^3 * 2^{20}$. Thus 8M words

Physical View

- Physical Space Available – All possible set of physical store locations
- Physical Address – This is true address as seen where information is stored in the hardware.
- **Conversion Examples –**

- If Logical address is 34 bit then -> Logical Address space = 2^{34} which is $2^4 * 2^{30}$. Thus 16 G words

Frames vs pages

Frames are basically the sliced up physical memory blocks of equal size. Example : 512kb of memory can be divided into 4 parts for 128kb each

While, **pages** are sliced up logical memory blocks of equal size. While solving any problem always the page size should be equal to frame size.

Some Formulas –

- No. of Frames = Physical address space/ Frame size
- No. of pages = Logical address space/ Page size

Logical Address

Page address is called logical address and is a combination of by page number (denoted as 'p') and offset (denoted as 'd').

Page number sometimes is also called as VPN(Virtual Page Number).

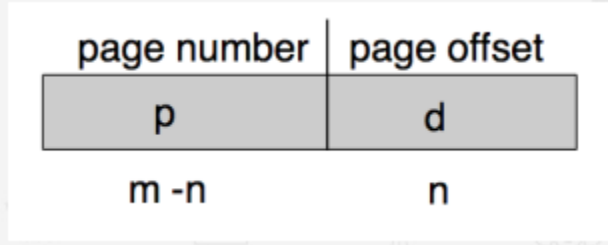
Logical Address = Page number + page offset

Logical Address – p + d

● For given logical address space 2m and page size 2n

Page Number(p)- Number of bits required to represent the pages in Logical Address Space

Page offset(d) – Number of bits required to represent particular word in a page or page size of Logical Address Space



Physical Address

Frame address is called physical address and is a combination of a frame number (denoted as 'f') and the offset (denoted as 'd').

Frame number is also sometimes referred as PFN (Physical Frame Number.

Physical Address = Frame number + page offset

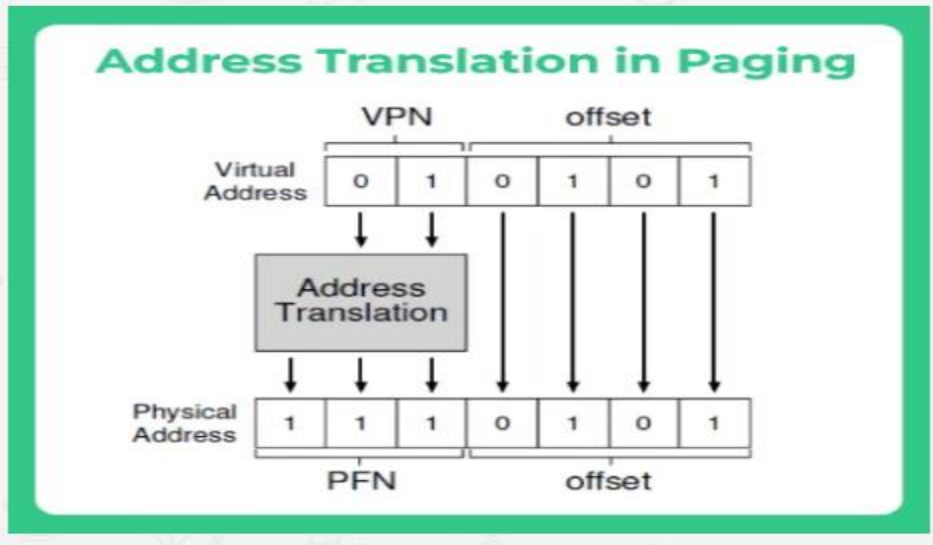
Physical Address – f + d

Frame Number(f) – Number of bits required to represent the pages in Physical Address Space

Frame offset(d) – Number of bits required to represent particular word in a page or page size of Physical Address Space

Note – We have seen students getting confused in if frame belongs to physical or logical and page belonging to physical or logical.

- Frame is Physical
- Page is Logical



How Paging Happens in System?
There are two ways paging can happen in the system

- Using Page Table
- Using translation look-aside buffers or TLB's

Using Page Table
Lets take an example to understand, where physical address is 14 bits and Logical address is 16 bits and let's assume a frame size(page size as both are equal) of 2 bit.

1. For 14 bits of Physical address the physical address space would be 16K.

1. Thus number of frame = physical address space/frame size = 16/2 = 8
2. f value would be 3 as 2 power 3 is 8 (need help? – Check frame number definition above)

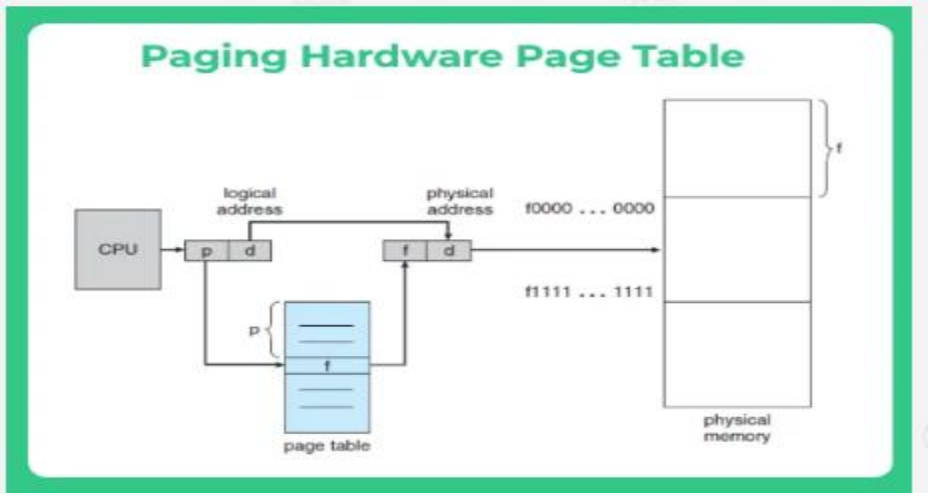
2. For 16 bits Logical address the logical address space would be 64 bits.
    1. Thus number of page = physical address space/page size = 64/2 = 32
    2. p value would be 5 as 2 power 5 is 32 (need help? – Check page number definition above)

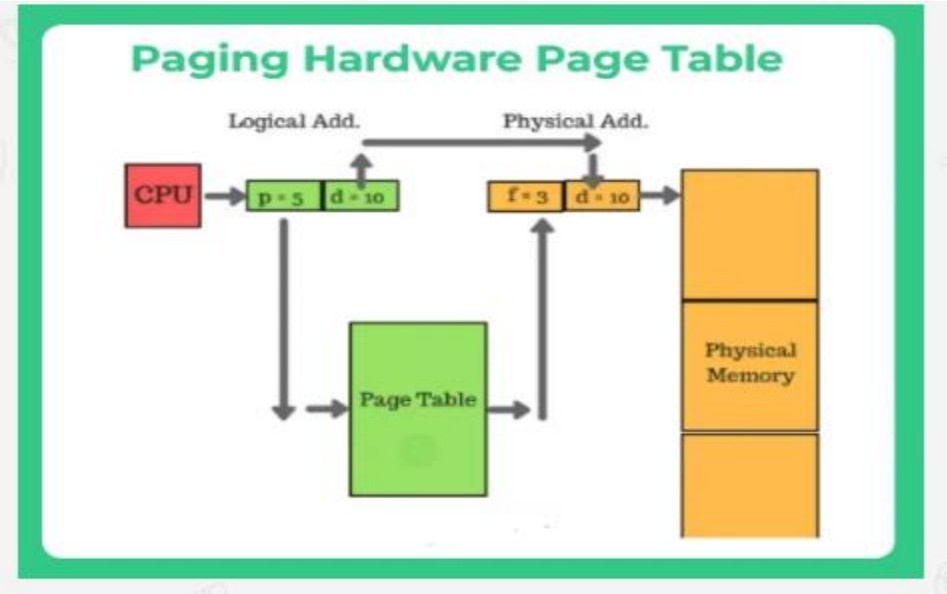
Help – Check conversion example given above on this post to understand how the above happened.

Additional Formula for Page table –

Size of page table = (total number of page table entries) *(size of a page table entry)

Assume a offset value of 10





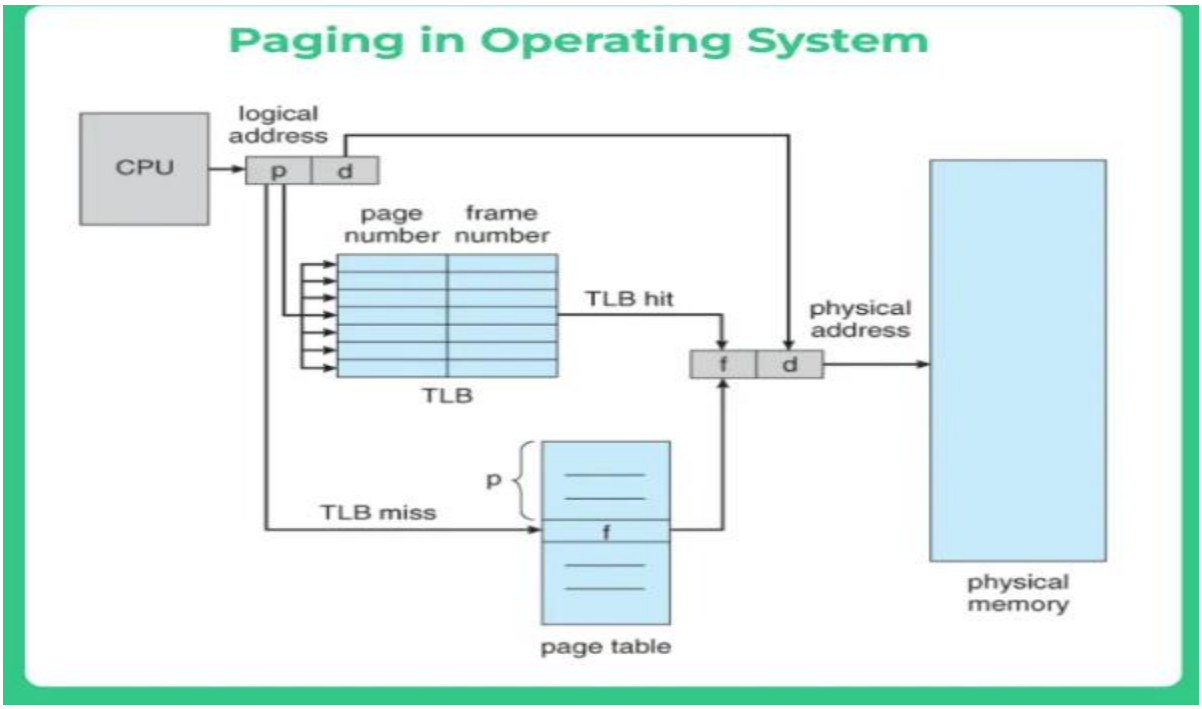
Additional Facts about Page Table

1. Page table is kept in main memory
2. Page-table base register (PTBR) points to the page table
3. Page-table length register (PTLR) indicates size of the page table
4. In this scheme every data/instruction access requires two memory accesses
    1. One for the page table
    2. one for the data / instruction
5. The two memory access problem can be solved – by the use of a special fast-lookup hardware cache – called associative memory or translation look-aside buffers (TLBs)

Using Translational Look-Aside Buffers or TLB's

We all know registers are faster than normal storage options in system, which is why in page tables registers are used. But, registers can cause poor performance if the the page table is very large. To solve this problem we use a special faster hardware called TLB i.e. translational look aside buffers.

These are fast, and they have a tag and value.



Types of Paging

What Is Paging?

Paging is a memory management scheme used in operating systems to efficiently manage memory resources. It allows the operating system to divide the logical address space of a process into fixed-sized blocks called pages. These pages are then mapped to physical memory, enabling efficient memory allocation and retrieval.

Demand Paging

How Demand Paging Works?

- **Demand paging** is a memory management technique where pages are loaded into memory only when they are needed.
- Instead of loading the entire program into memory at once, the operating system loads pages on-demand as the program accesses specific memory locations. This approach minimizes the initial loading time and conserves memory resources.

Advantages of Demand Paging

- **Reduced memory usage:** Demand paging allows programs to use less memory since only the required pages are loaded into memory.
- **Faster startup times:** By loading only the necessary pages, the initial startup time of programs is significantly reduced.
- **Improved system responsiveness:** Demand paging improves system responsiveness by loading pages on an as-needed basis, ensuring that the most frequently accessed pages are always available in memory.

Limitations of Demand Paging

- **Page faults:** If a program tries to access a page that is not present in memory, a page fault occurs. This can cause a slight delay as the operating system fetches the required page from secondary storage.
- **Increased I/O overhead:** Demand paging involves frequent disk I/O operations to retrieve pages from secondary storage, which can impact system performance if not managed efficiently.
- **Thrashing:** In some cases, demand paging can lead to thrashing, where the system spends more time swapping pages in and out of memory than executing useful work.

Prepaging

Understanding Prepaging

- Prepaging is a technique where the operating system anticipates future memory requirements and loads additional pages into memory before they are actually requested.
- This proactive approach aims to reduce page faults and improve overall system performance by prefetching pages that are likely to be needed soon.

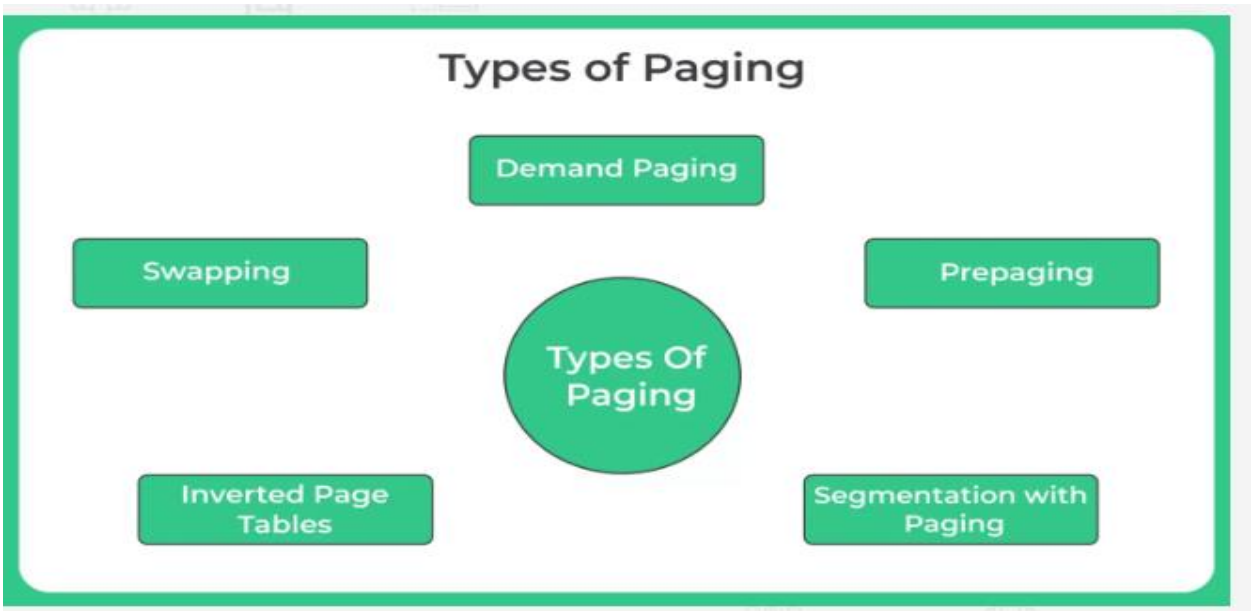
Benefits of Prepaging:

- **Reduced page faults:** By loading additional pages in advance, prepaging minimizes the occurrence of page faults during program execution.

- **Enhanced responsiveness:** By prefetching data that is likely to be accessed, prepaging ensures that the necessary pages are readily available, improving system responsiveness.
- **Efficient memory utilization:** Prepaging can optimize memory usage by identifying and loading related pages, reducing unnecessary disk I/O operations.

Drawbacks of Prepaging

- **Increased memory overhead:** Prepaging requires additional memory resources to store the prefetched pages, which can impact systems with limited memory capacity. **Uncertain prediction accuracy:** Predicting future memory requirements accurately can be challenging, and incorrect predictions may lead to wasted resources and decreased performance.



Swapping

The Concept of Swapping

Swapping is a paging technique where entire processes or parts of processes are temporarily moved out of main memory and into secondary storage, typically a hard disk. This allows the operating system to free up memory for other processes when the available physical memory becomes insufficient.

Pros of Swapping

- **Efficient memory utilization:** Swapping enables the operating system to utilize secondary storage as an extension of physical memory, allowing more processes to run concurrently.
- **Flexibility in process management:** Swapping facilitates the execution of processes with memory requirements larger than the available physical memory, preventing resource constraints.
- **Multiprogramming support:** Swapping enables the execution of a greater number of programs simultaneously, enhancing overall system efficiency.

Cons of Swapping

- **Increased I/O overhead:** Swapping involves frequent disk I/O operations to move processes between main memory and secondary storage, potentially affecting system performance.
- **Delayed process execution:** Swapping can introduce delays in process execution as processes need to be swapped in and out of memory, resulting in increased response times.
- **Data integrity risks:** Swapping poses a risk of data loss or corruption if a system failure occurs while a process is swapped out, but its modified data is not yet written back to disk.

Segmentation with Paging

Overview of Segmentation with Paging

- **Segmentation with paging** combines the benefits of both segmentation and paging techniques. In this approach, a program's logical address space is divided into segments, which are further divided into fixed-sized pages.
- This hybrid scheme offers advantages in terms of **memory management flexibility** and **address space organization**.

Advantages of Segmentation with Paging

- **Improved memory management:** Segmentation with paging provides greater flexibility in managing memory by allowing dynamic growth and sharing of segments.
- **Enhanced protection and security:** Segment-level protection mechanisms can be applied, providing finer-grained access control to program segments.
- **Efficient address space allocation:** Combining segmentation and paging allows for more efficient allocation of address space, reducing internal and external fragmentation.

Limitations of Segmentation with Paging

- **Increased complexity:** Implementing segmentation with paging requires additional hardware support and more sophisticated memory management algorithms, increasing system complexity.
- **Overhead in address translation**: The combined address translation process of segmentation and paging can introduce additional overhead compared to other memory management schemes.
- **Potential for fragmentation:** While segmentation with paging reduces external fragmentation, it may still be susceptible to internal fragmentation within individual segments.

Inverted Page Tables

Inverted Page Tables

Inverted page tables are an alternative approach to traditional page tables used for virtual-to-physical address translation. Unlike conventional page tables, where each process has its own page table, inverted page tables maintain a single global table that maps physical frames to the corresponding virtual pages.

Benefits of Inverted Page Tables

- **Reduced memory overhead:** Inverted page tables require less memory since they store mappings only for the pages currently in use, rather than the entire address space of each process.

- **Improved TLB efficiency:** The smaller size of inverted page tables enables better utilization of the Translation Lookaside Buffer (TLB), resulting in faster address translation.
- **Scalability:** Inverted page tables can efficiently handle systems with a large number of processes, as they eliminate the need for duplicating page tables for each process.

Challenges with Inverted Page Tables

- **Slower page table lookup:** Inverted page tables require a linear search to find the mapping for a given physical frame, which can be slower compared to direct lookups in traditional page tables.
- **Increased complexity:** Maintaining coherence and managing updates in a shared global page table can be more complex, especially in systems with concurrent access from multiple processes.
- **Limited support for sharing:** Inverted page tables may have limitations in supporting efficient memory sharing among processes due to the shared nature of the tables.

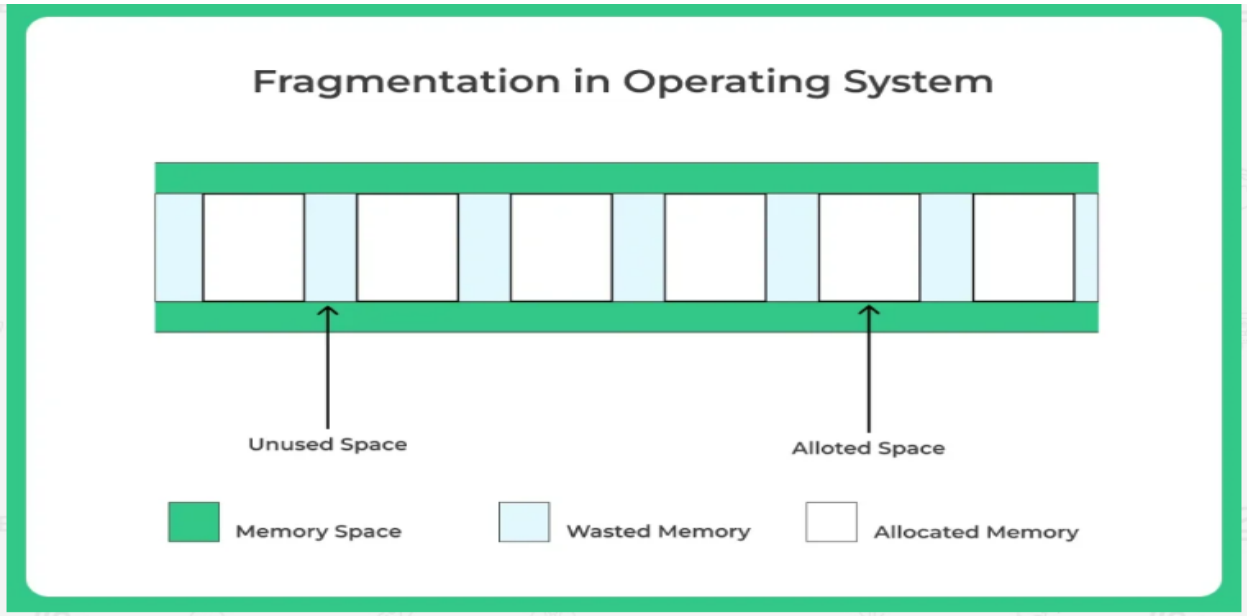
Fragmentation in OS

What is Fragmentation in OS

Memory space in the system constantly goes through loading and releasing processes and their resources because of which the total memory spaces gets broken into a lot of small pieces, this causes creation small non utilised fragmented memory spaces, which are so small that normal processes can not fit into those small fragments, causing those memory spaces not getting utilised at all, this is called memory Fragmentation in operating system.

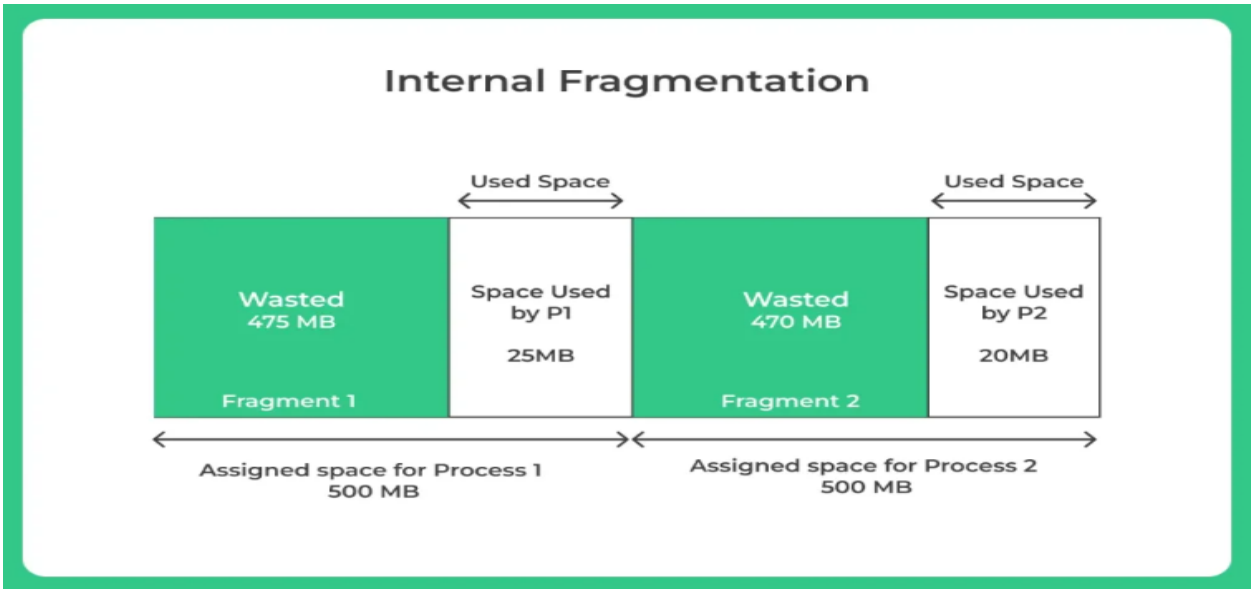
Fragmentation is of the following two types –

1. Internal Fragmentation
2. External Fragmentation



Internal Fragmentation

Memory block that is given to the process can comfortably use the block but the size of the block announced, but it is way bigger than the memory process will require thus causing a that memory remaining unused for further processes.



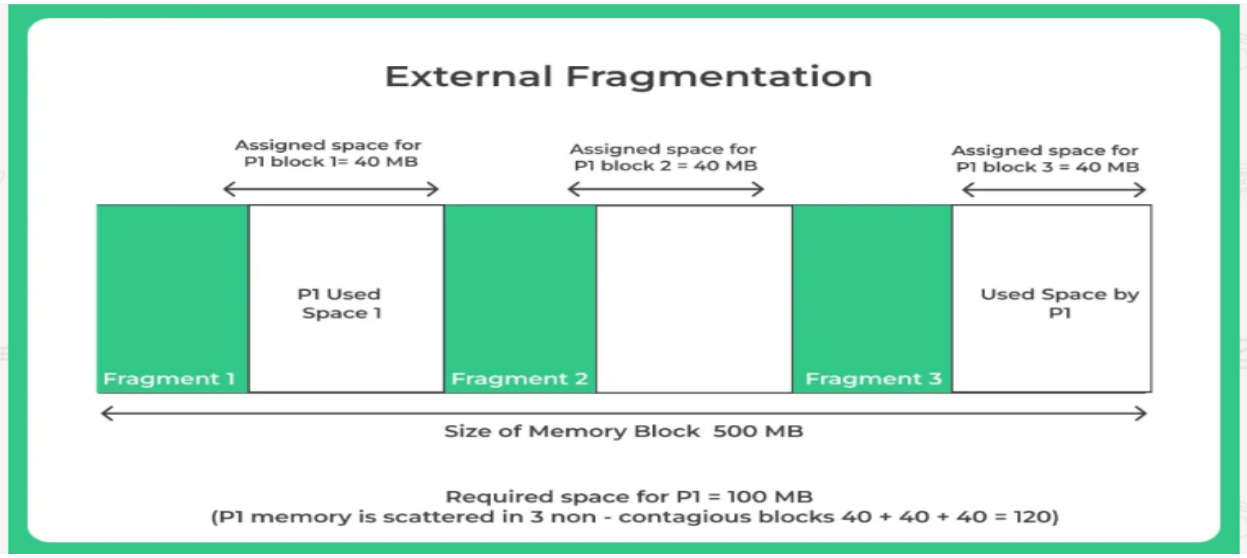
External Fragmentation

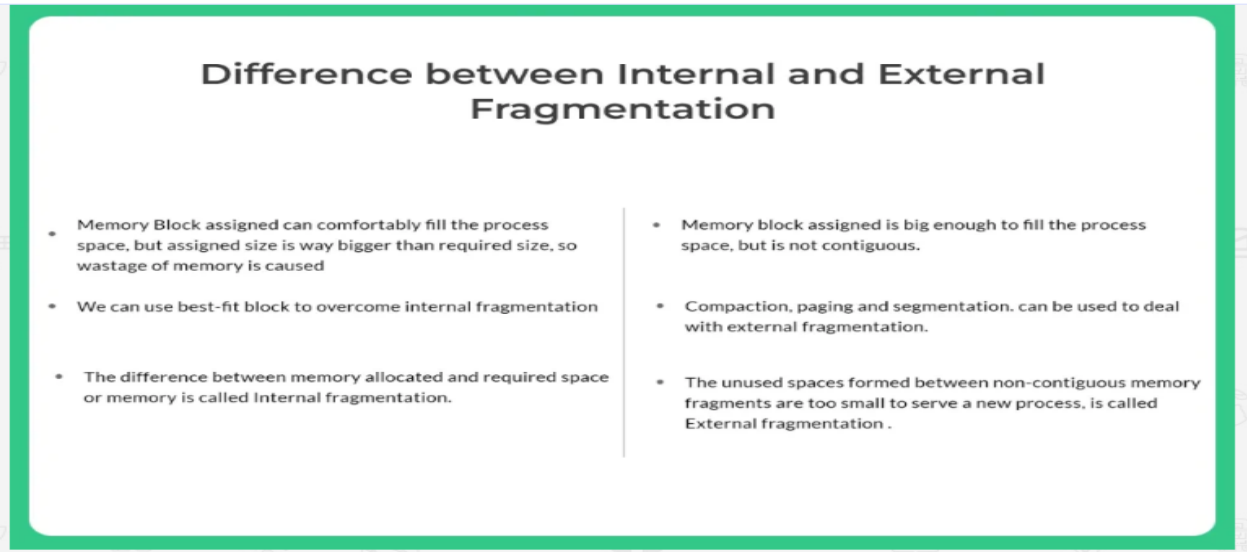
The memory space in the system can comfortably satisfy for the processes, but the available memory space is non-contiguous, thus it further can't be utilised.

How to solve the fragmentation problem

Re-shuffling the utilised contents of the memory and placing all them together contiguously. Thus compacting the same and free spaces occur together and allocated spaces occur together, **this is called compaction for External Fragmentation**.

Internal Fragmentation has a few to effectively be reduced by intelligently assigning smaller partitions which though are large enough for the processes.

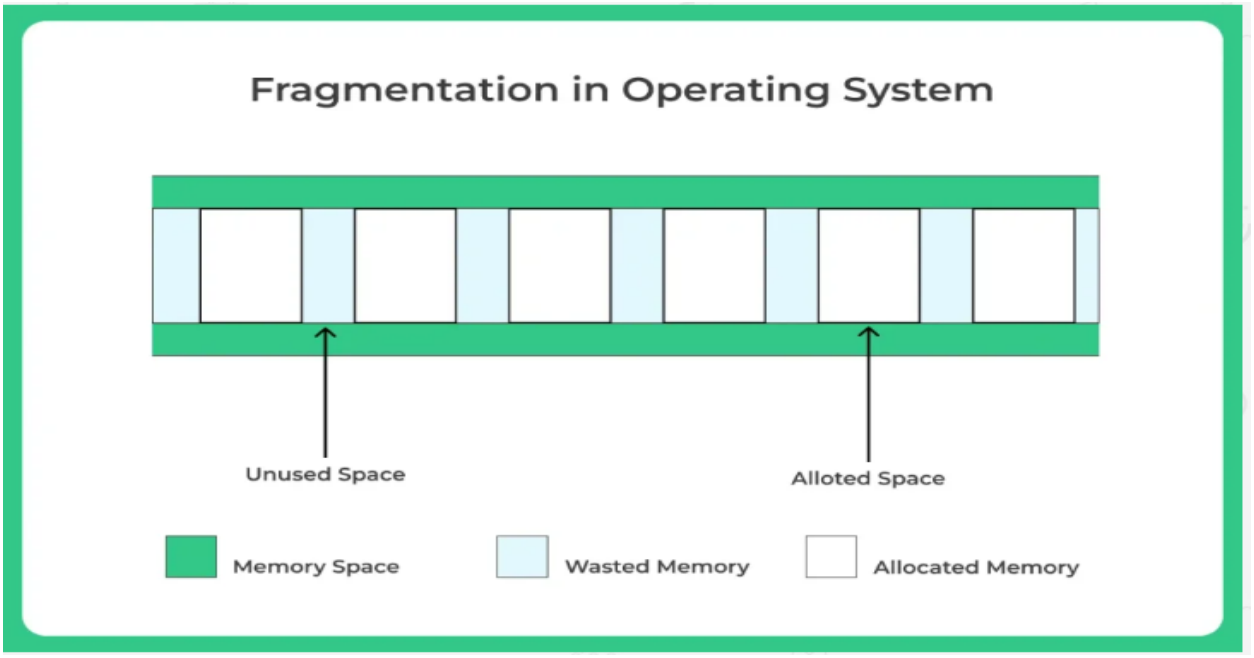




Internal Fragmentation

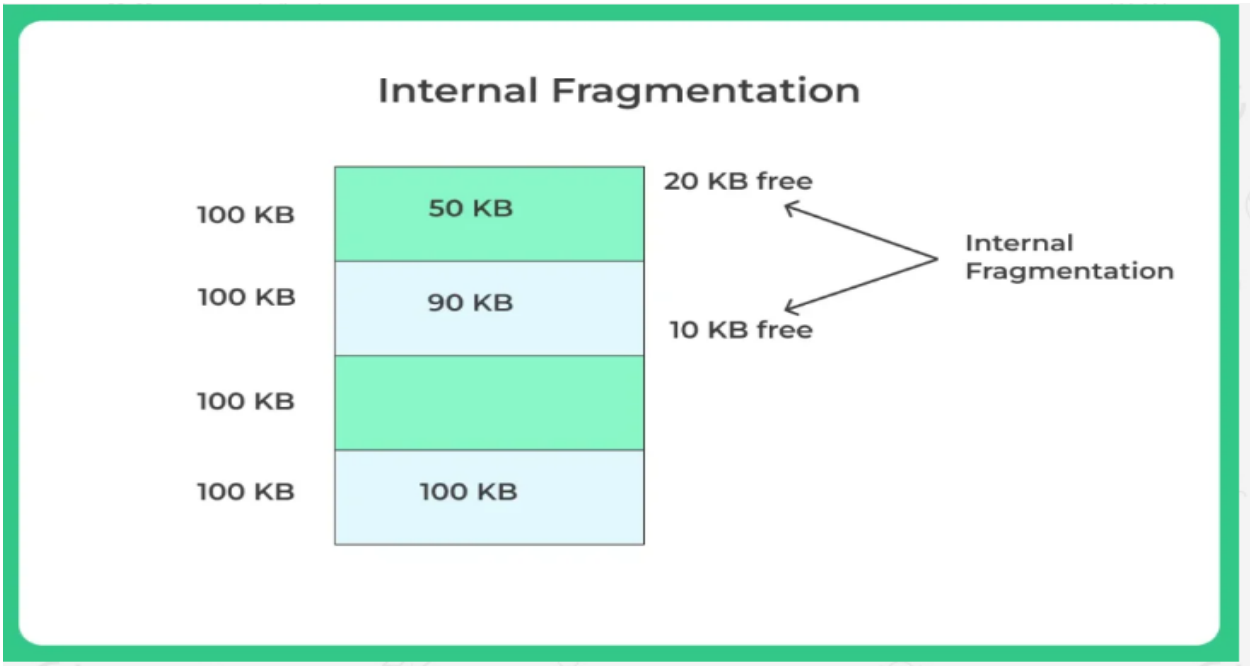
Fragmentation

Users keep loading and unloading processes from the main memory. Usually, processes are stored in blocks in the memory. Sometimes the available memory blocks are not sufficient to store a process that requires contiguous memory spaces. This condition is known as fragmentation. In other words, fragmentation occurs when users dynamically allocate RAM to processes and despite much sufficient space available in the memory, the process cannot be loaded as continuous memory allocation is not possible.



Internal Fragmentation

When a process is assigned to a memory block and if that process is smaller than the memory requested, it creates a free space in the assigned memory block. Then the difference between assigned and requested memory is called internal fragmentation. Usually, memory is divided into fixed size blocs.



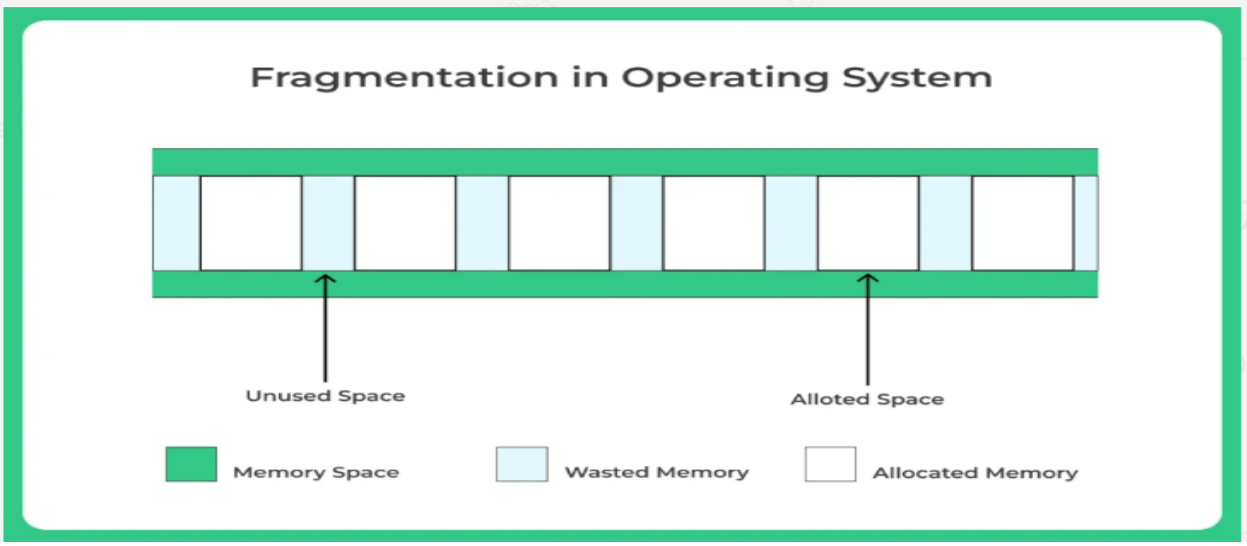
When it occurs?
When the memory assigned to the process is larger than the memory requested by the process.

Example
The memory space is divided into the fixed-sized blocks of 32 bytes. Let us consider that program 'A' request for 24 bytes and a fixed-sized block of 32 bytes is assigned to the program. Now, the difference between assigned and requested memory is 8 bytes which is internal fragmentation.

What is the solution?
The memory must be divided into variable sized blocks and assign the best-fit block to the process requesting for the memory. In simple words, internal fragmentation can be reduced effectively by allocating the smallest partition but large enough for the process. However, the problem will not be resolved entirely but can be mitigated to some extent.

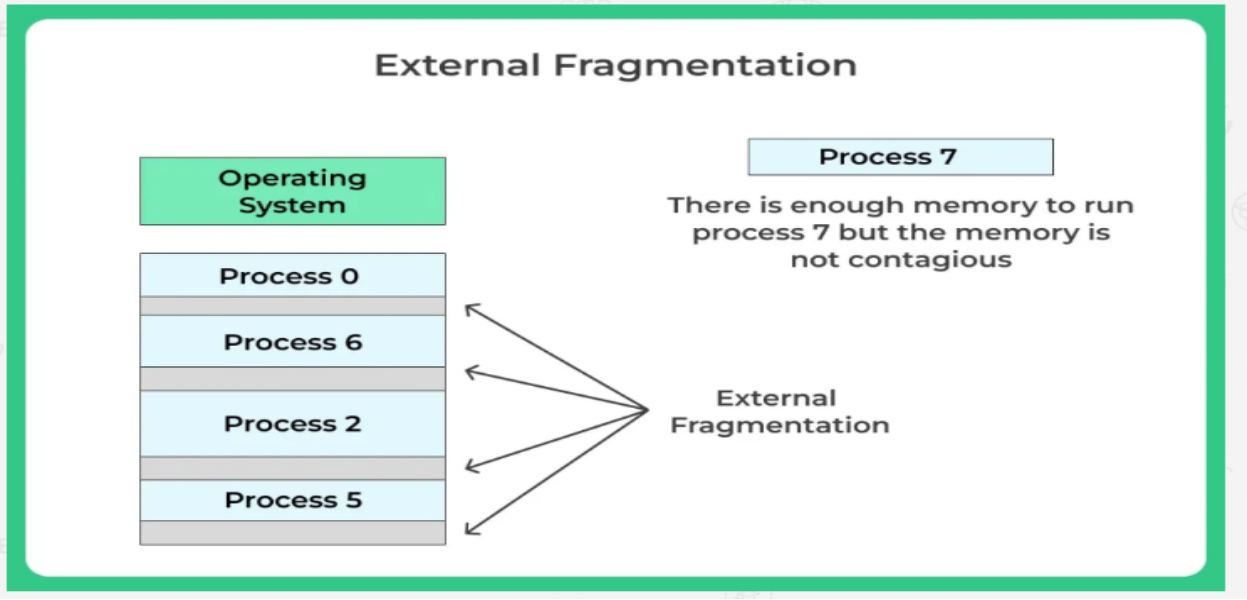
External Fragmentation

Fragmentation
Whenever a process is loaded or removed from the physical memory block, it creates free spaces in the memory, which are commonly known as fragments. These small fragments in the memory cannot be allocated in a contiguous manner to any process as these spaces are not continuous. Consequently, the memory is wasted and cannot be used by another process. This problem is called fragmentation.



External Fragmentation

When the process is loaded or removed from the memory, a free space is created. This free space creates an empty space in the memory which is called **external fragmentation**.



When it occurs?
When portions of allocated memory are too small to hold any process.

Example
The RAM has a total of 10 kb free space , but it is not contiguous, or is fragmented. If a process with 10 kb size wants to loads on the RAM, then it cannot load because space is not contiguously free.

What is the solution?
The solution for external fragmentation is compaction or shuffle memory contents. In this techniques all the memory contents of memory are shuffled and all free memory is put together in one large block. In order to make compaction feasible, relocation should be dynamic. Also, external fragmentation can be resolved by paging or segmentation mechanisms, which will allow a process to acquire physical memory in a non-contiguous manner.

Mapping Virtual Address to Physical Address

Mapping Virtual Address to Physical Address in OS

The physical memory is basically a large array of words (storage spaces), with each having its own address. The CPU allocates logical memory address to any given physical address where actual data is stored. Since, in real case scenario related data may be stored at different contiguous locations. Logical address generated by CPU may help in viewing the data stored as contiguous.

CPU also, performs the job of fetching the memory / instructions based on the program counter. It may load or store data in those physical address based array of words.

Address Binding
Address binding is essentially the process of binding (mapping) different address spaces to another.

Different Types of Addresses
There are two different types of addresses in the system –

- Logical or Virtual
- Physical

The physical address is where the actual data is stored, and the logical address generated by CPU is how the system will see the addresses.

Physical memory (RAM) is divided into pages contiguous sequences of memory typically in the 4KB-16KB range. Physical addresses are provided by the hardware.

- One physical address space per machine
- Valid addresses are usually between 0 and some machine-specific
- Not all addresses have to belong to the machine's main memory.
- Other hardware devices can be mapped into the address space.

Virtual (or logical) addresses are provided by the OS kernel:

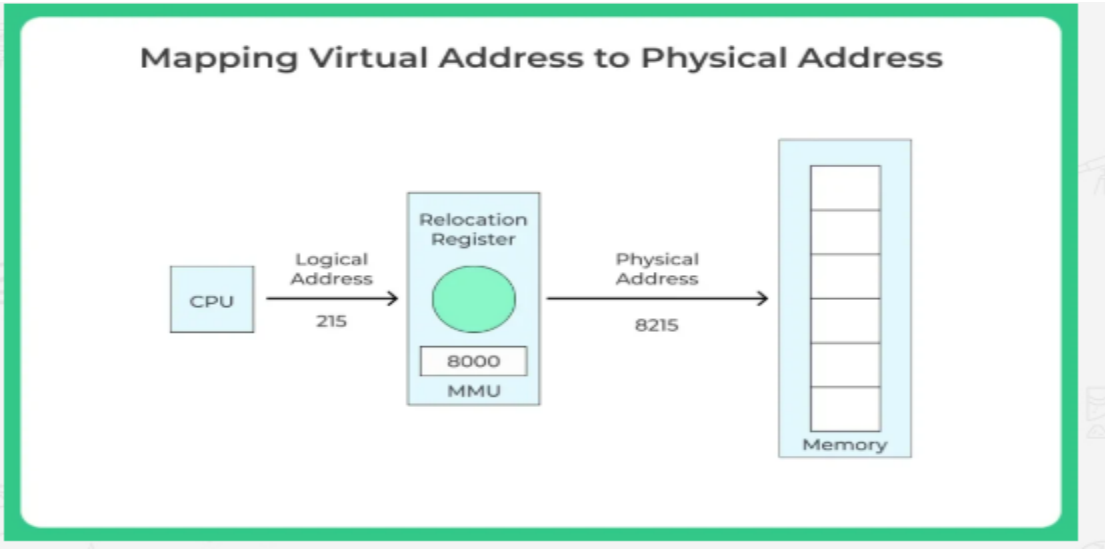
- One virtual address space per process
- Addresses may start at zero, but not necessarily
- Space may consist of several segments

Memory Management Unit (MMU)

The user only thinks that he is accessing the data from the logical address. The translation from the logical to the physical address is done by special equipment in the CPU that is called the Memory Management Unit (MMU), in some systems it is also called as address translator unit.

MMU Additional does –

- Checks for protection violations
- Raises exceptions when necessary (e.g., write operation on readonly memory region).



The address translation can be done in the three ways or at three different stages –

Compile Time

In some cases it is pre-known at the compile time itself about the final address (physical) of the memory for a running process.

This helps the system overall as –

- It is faster to load an existing executable format process

Note – There are chances for the program itself to crash as the defined address space (physical) may already be preoccupied by another process. In this case crash may happen and the system itself or with user intervention has to re-compile the whole program.

Load Time

The above crashing issue can be solved here, or in cases where it is not known that where the process memory will finally be residing. Here the relocatable address will be generated.

Loader does –

- Job of the loader is to translate the above relocatable address to an absolute address
- Adds : Base address to all, logical addresses, in turn to generate absolute address.

Note – Reloading of the whole process is necessary if in case the base address itself has changed in the mean time,

Execution Time

We all know that at the time of execution of a process we may need dynamic memory to save intermediate results or remove the results, i.e. allocation and de-allocation is inadverent at execution time.

Adding addresses as follows –

- CPU generated logical address
    - 245
- MMU generated relocation register, which is also called as base register
    - 8000
- So finally physical memory location will be
    - 8000 + 245 = 8245

Virtual Memory in Operating System

Virtual Memory in Operating System

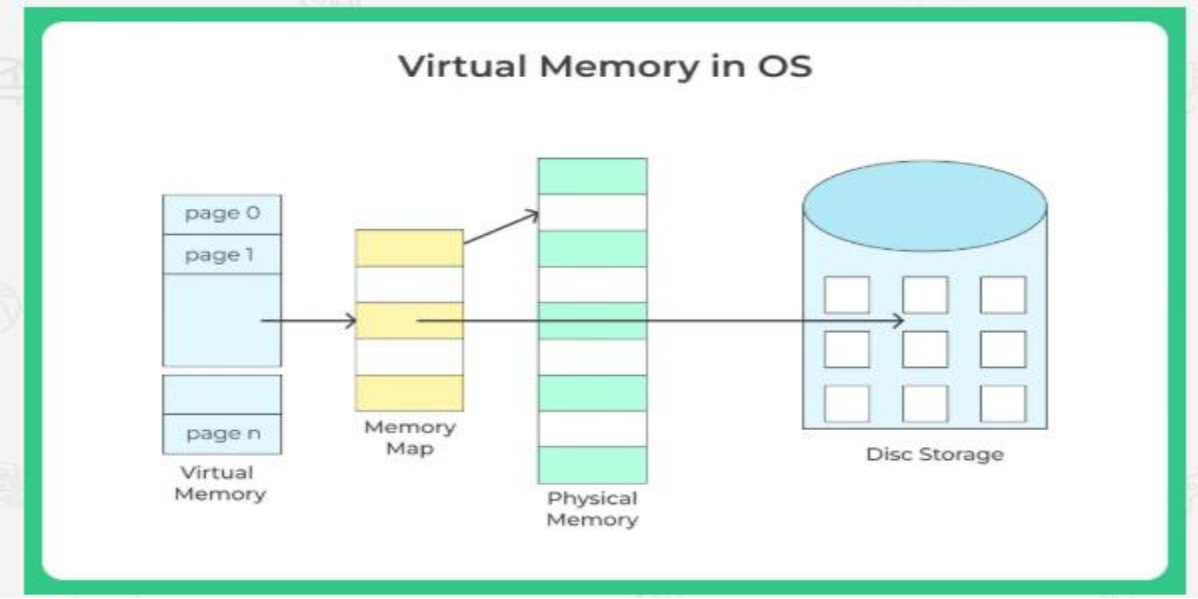
The Primary Memory (such as RAM) is expensive and limited in the system, however, the secondary memory is cheaper and large and can be extended easily. Virtual memory concept offers unique solution for apparent primary memory expansion, in this we secondary memory

can be accessed as if it were the part of the main memory. One of the techniques to extend this is to transfer the inactive RAM storage to secondary storage temporarily.

Virtual Memory

It essentially with the help of Operating System, and the software and hardware installed on the system, helps to access more memory than what actually is installed on the system, by transferring data from the RAM to the to disk.

- Primary Memory – RAM
    - The entry application or program in the system first loads into the RAM (Random Access Memory), this makes the application faster to load and access, than the secondary memory and is readily available
- Secondary Memory – Hard Disk and Storage devices



In what Form Virtual Memory is stored?

The memory is stored in form of units that we call as pages in Operating System, these are atomic units where large programs can be stored.

For a program there may a large set of storage in physical addresses. All of those are not required to be loaded as only certain programs are used, for an application.

Imagine playing a game like FIFA on your desktop. The game itself maybe stored in hard disk (secondary device), but while playing all the installed space would not be loaded into RAM. But only certain programs that are required for the application (Game) to run .

Benefits of having Virtual Memory

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

How it works?

The Addressing scheme that is provided by the system and total available space in the secondary storage device is what limits the size of the virtual storage, not the size of Primary storage as RAM. If the storage is unlimited (as in the case of cloud systems), some applications can behave as if they have unlimited primary memory available to them.

We use both of these of the system –

- Hardware
- Software

Both work together to map virtual Addresses to physical addresses, with the help of MMU, i.e. Memory Management Units.

1. What system or uses sees are essentially logical addresses not the actual physical address. These logical addresses may be translated dynamically at the load, execute or compile time. Depending on the requirement or definition of the system.
2. In actual scenario, the processes and resources stored in the memory are not continuous, however are broken down and are scattered and stored at various places in the physical memory. With the help of dynamic run-time address translation and page or segment table we can easily do this. That even though they actually may be scatterly stored but to the user it looks like they are stored contiguously.

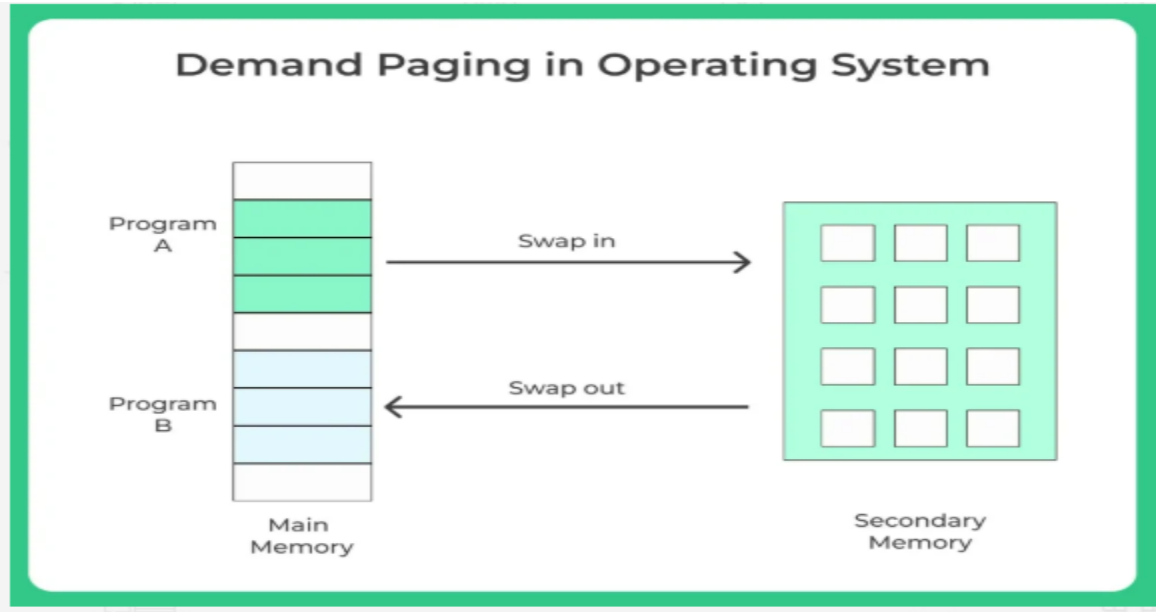
Demand Paging in Operating System

The basic purpose of the OS is reduce load on the system and become more efficient. Consider that a process enters the system i.e. swapped in. Every process as we discussed in the virtual memory post here has a lot of pages.

It might not be suitable and efficient to swap or load all the pages for the process at once. As the program may only need only a certain pages for the application to run.

- Infact, in cases a 1GB application may need as little as 20MB pages to be swapped so there is no need to swap all at once.

Demand paging is the process that solves, this by only swapping pages on Demand. This is also known as lazy swapper.



How Demand Paging Works?

The main components involved in the process are –

- CPU
- Main Memory
- Interrupt
- Secondary Memory
- Logical Address space
- Physical Address space
- Page Table
- Operating System

1. A request may be made to the CPU for a page that may not be available in the main/primary memory in active state. Thus, it has to generate an interrupt.
2. This is when the Operating system moves the process to a blocked state as an interrupt has occurred.
3. The Operating system then searches the given page in the Logical address space
4. Finally with the help of page replacement algorithms, replacements are made in the physical address space. Simultaneously, page tables are updated.
5. CPU is informed about the update and asked to go ahead with the execution and process gets back into the ready state.

Page Fault Service Time

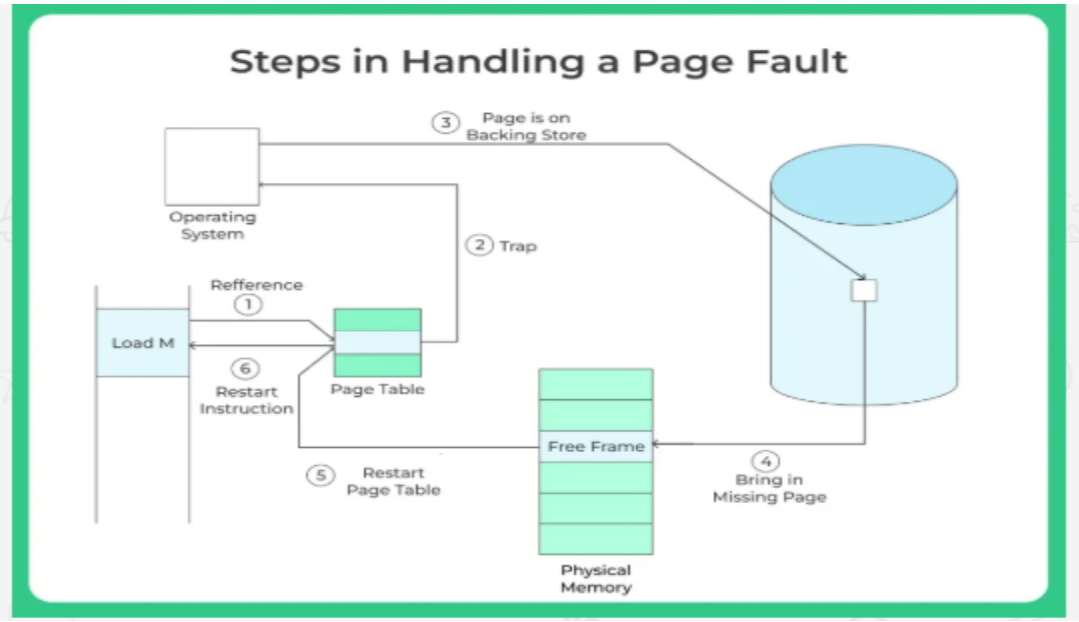
Whenever a page fault occurs, the CPU takes considerable amount of time to perform the above mentioned steps. This time taken by the CPU is called as page fault service time.

- Access time for Main memory : m
- Service time for Page fault: s
- Rate for Page fault Page fault rate is : p
- Effective memory access time = (p*s) + (1-p)*m
    - This formula is important for both placements and gate.

Implementation of Demand Paging in OS

The main steps involved in demand paging that is in between the page is requested and it is loaded into main memory are as follows:-

1. CPU refers to the page it needs.
2. The referred page is checked in page table whether that is present in main memory or not.If not an interrupt page fault is generated.
3. OS puts the interrupts process in blocking state and starts the process of fetching the page from memory so that process can be executed.
4. OS will search for it in logical address space.
5. Required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
6. Page table will be updated.
7. Interrupted process will be restarted.



These are steps when a page required is missing from the memory.

Performance of Demand Paging

Demand paging can significantly affect the performance of a computer system. To see why, let's compute the effective access time for a demand-paged memory. the memory-access time, denoted ma,ranges from 10 to 200 nanoseconds.

effective access time = (1 − p) × ma + p × page fault time.

where p is probability of page fault.

Three major activities

– Service the interrupt – careful coding means just several hundred instructions needed
– Read the page – lots of time
– Restart the process – again just a small amount of time

Segmentation in Operating System

How Segmentation works?

Just like page table or TLB, for segmentation in operating system all the information necessary is stored into the segment table. The relationship between the logical and physical address is a bit more complex here.

The unit involved here are –

- Segment table
- Base Address
- Limit
- Segment Number (s)
- Segment Offset (d)

Segment Table

The correlation and mapping for the logical address which is 2 dimensional address and the physical address is done with the help of segment table.

Base Address

The starting address of the memory where the data is stored in its logical format in the system.

Say Base address is 8000, then the data will be available in addresses from 8000 onwards.

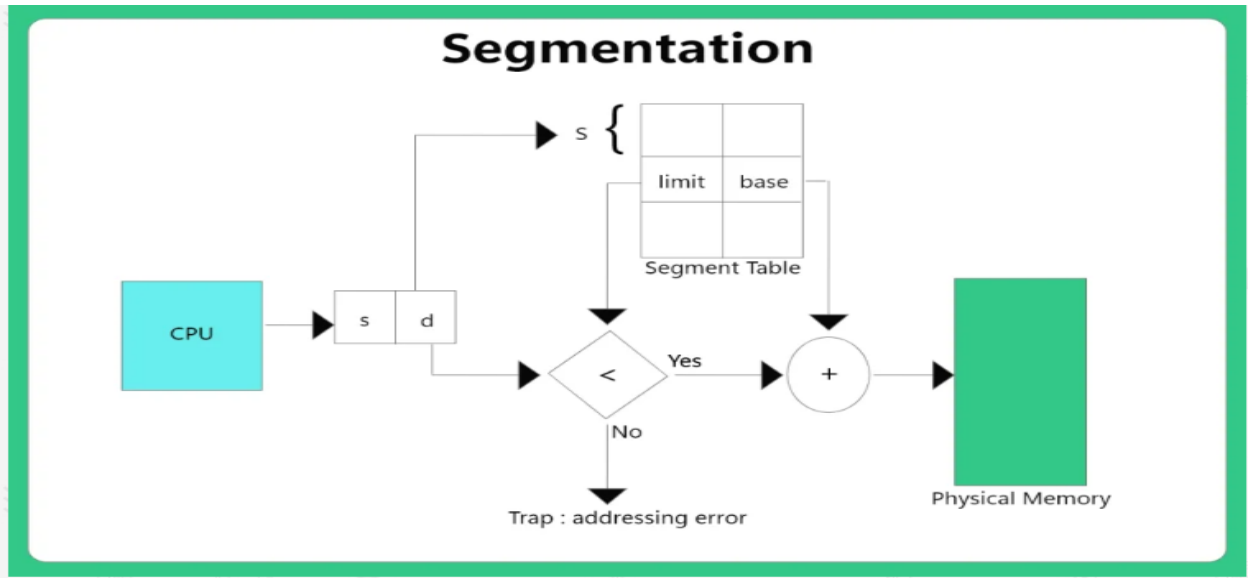
Limit

Since the size of each segment is different, thus each segment has its own limit defined and it tells the system about the size or length of any particular segment.

Further for mapping to physical form for the CPU the following are required –

1. Segment Number (s) – It is basically the total number of bits that are required to store any particular address of the segment in consideration
2. Segment Offset (d) – Number of bits required for the size of the segment.

The process can be easily visualised from the image below-



Contrast with Paging

Paging

- Page size is fixed
- Can lead to internal fragmentation
- Size is decided by the Hardware
- Address mapping is simple and page table is sufficient

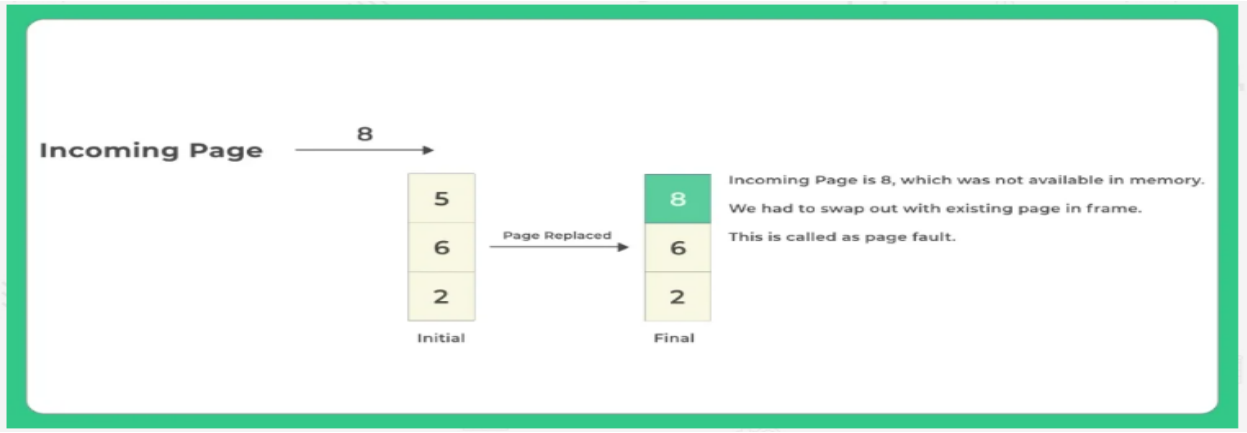
Segmentation

- Segment size is variable
- Can lead to external fragmentation
- Size is decided by the user
- Address mapping is complex even with the help of segment table, segment number and offset.

Page Replacement Algorithms in OS

Page Fault – In simple terms, if an incoming stream item (page) is not available in any of the frames. Then page fault occurs.

When does page fault Occur
A Page Fault occurs when a program running on the CPU tries to access a page that is in the address space of that program, but the requested page is currently not loaded into the main physical memory, the RAM of the system



FIFO-First In First Out

A FIFO replacement algorithm associates with each page the time when that page was brought into memory.

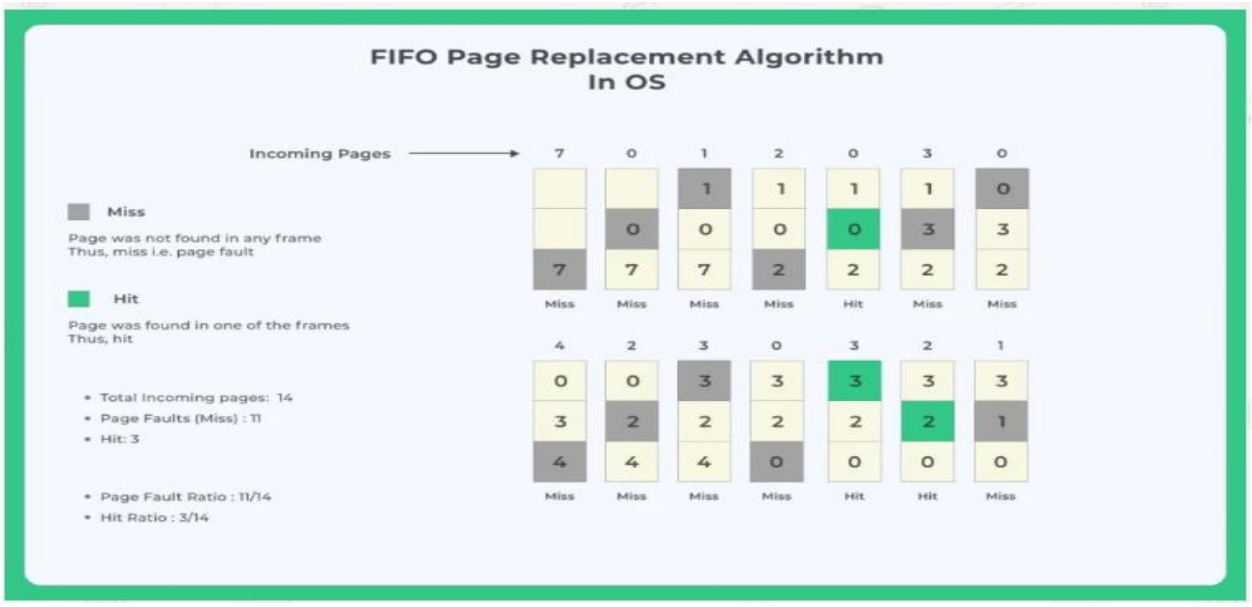
This is how FIFO works –

- If an incoming page is not available in any of the frames. Replacement shall be done.
- Page replaced is according to FIFO (First in First Out)
- The page that entered first must be swapped out first
- In other words, the oldest frame must be swapped out

Example
It might be a confusing theory, lets look with an example –

- Incoming page Steam – **7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1**
- Page Size/ Frame Size = 3

Table will look like(Explanation after the table)



Explanation –

- Step n [Incoming Stream] – Current Stack [x,y,z]
- Current Oldest – In Red
- After Replacing – In Green

Goes as

- Step 1 [7] – Current Stack [nil, nil, nil] (Page Fault Occurs)
    - Result – [7, nil, nil]
- Step 2 [0] – Current Stack [7, nil, nil]
    - Result – [7,0,nil] (Page Fault Occurs)
- Step 3 [1] – Current Stack [7, 0, nil]
    - Result – [7,0,1] (Page Fault Occurs)
- Step 4 [2] – Current Stack -[7,0,1]
    - Result – [2,0,1] (Page Fault Occurs)
- Step 5 [0] – Current Stack [2,0,1]

    - Result – [2,0,1] as 0 is already Present
- Step 6 [3] – Current Stack [2,0,1]
    - Result – [2,3,1]
- Step 7 [0] – Current Stack [2,3,1]
    - Result – [2,3,0] (Page Fault Occurs)
- Step 8 [4] – Current Stack [2,3,0]
    - Result – [4,3,0]
- Step 9 [2] – Current Stack [4,3,0]
    - Result – [4,2,0] (Page Fault Occurs)
- Step 10 [3] – Current Stack [4,2,0]
    - Result – [4,2,3] (Page Fault Occurs)
- Step 11 [0] – Current Stack [4,2,3]
    - Result – [0,2,3] (Page Fault Occurs)
- Step 12 [3] – Current Stack [0,2,3]
    - Result – [0,2,3] (3 already there)
- Step 13 [Incoming Stream] – Current Stack [0,2,3]
    - Result – [0,2,3] (2 already there
- Step 14 [1] – Current Stack [0,2,3]
    - Result – [0,1,3] (Page Fault Occurs)

If you have any questions ask in the comments section, and we will help.

Now page fault occurs 11 times out of 14.

So page fault ratio is 11/14.

Belady's Anomaly
Think – Now, one should think that increase the page size(frame size as page size = frame size) will lead to less page fault, right!! since more chances of element to be present in the queue.

But, that's not the case always. Sometimes by increasing the page size page fault rather increases, this type of anomaly is called belady's Anomaly.(asked in AMCAT, CoCubes, Oracle)

For example, consider the following(solve yourself for practice)

- 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4
    - Using 3 slots gives 9-page faults
    - Using 4 slots gives 10-page faults

Least Recently Used

In this algorithm, we replace the element which is the current least recently used element in the current stack.

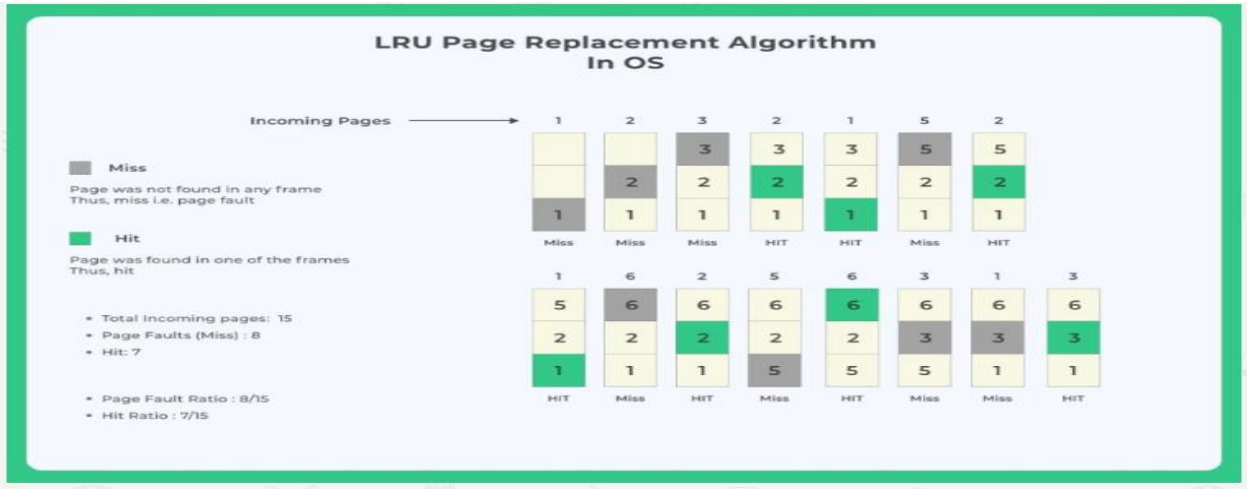
That is, when we look to the left of the table, that we have created we choose the further most page to get replaced.

ExampleLet's say d is the incoming page and the current stack is [a,b,c].

Let's assume we are at the 10th iteration
1. a was last entered in current stack at 4th iteration.
2. b was entered at 2nd iteration
3. c was entered at 7th iteration

In this case d will replace b as b is the least recently used as was last seen in 2nd iteration

TrickThe trick is to look for the most recent occurrence of a page towards the left of the table and whichever is the furthermost. The incoming page should replace that.



Optimal Page Replacement

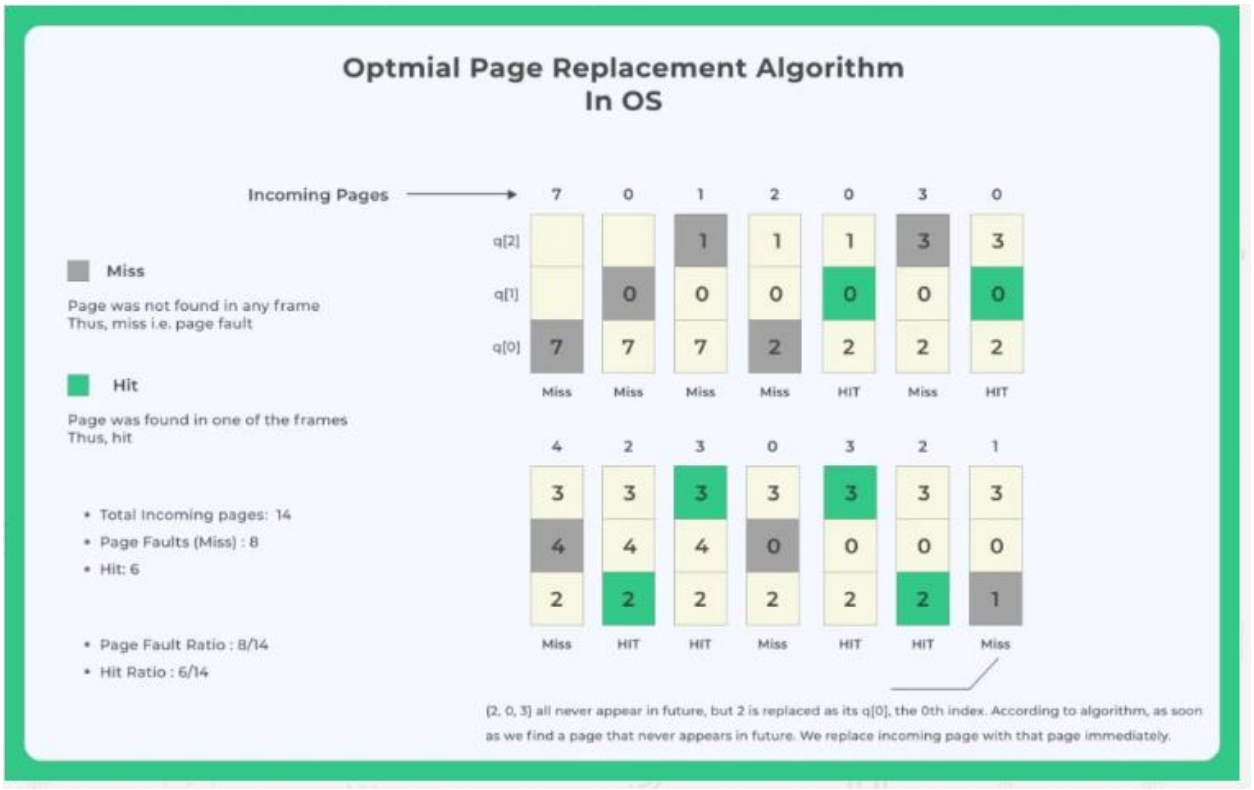
In LRU we looked for the left furthermost page to replace. In optimal we do the opposite and look for right furthermost.

Now, this is done so that there are lesser page faults as the element will not used for the longest duration of time in the future.

Optimal Page Replacement Quick Facts

- The result of the discovery of Belady's Anamoly
- Lowest page fault rate of all algorithm's and will never suffer from belady's Anamoly.
- Simply it replaces the pages that won't be used for longest period of time.
- Optimal page replacement is perfect, but not possible in practice as operating system cannot know future requests.
- The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

The image below shows the implementation of the Optimal page replacement Algorithm.



FIFO Page Replacement Algorithm

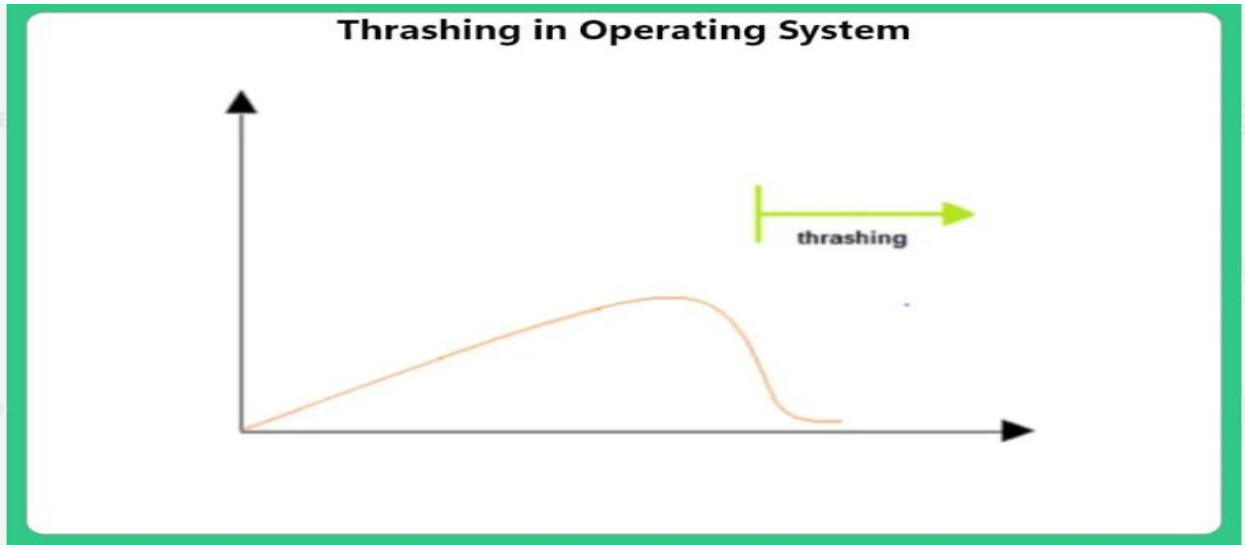
Definition

FIFO is an acronym for First in First out approach. The concept is based on the fact that the elements present in the stack are removed following the same order in which they were filled. Precisely, the element present at the bottom of the stack will be removed first. This also means the element present on the top of the stack will be removed last.

Thrashing in Operating System (OS)

Page Fault and Swapping

A page fault occurs when the memory access requested (from the virtual address space) does not map to something that is in RAM. A page must then be sent from RAM to swap, so that the requested new page can be brought from swap to RAM. This results in 2 disk I/Os. Now you might know that disk I/Os are very slow as compared to memory access.



Thrashing in Operating System

Now if it happens that your system has to swap pages at such a higher rate that **major chunk of CPU time is spent in swapping** then this state is known as thrashing. So effectively during thrashing, the CPU spends less time in some actual productive work and more time in swapping.

While processing page faults it is necessary to be in order to appreciate the benefits of virtual memory, thrashing has a negative effect on the system.As the page fault rate increases, more transactions need processing from the paging device.

While the transactions in the system are waiting for the paging device, CPU Utilization, system throughput and system response timedecrease, resulting in below optimal performance of a system.

Cause of Thrashing

1. **High degree of multiprogramming**:
   The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more

2. **Lack of frame**s:-
   If a process has less number of frames then less pages of that process will be able to reside in memory and hence it would result in more frequent swaping. This may lead to thrashing. Hence sufficient amount of frames must be allocated .

Recovery

- Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
- If the system is already in thrashing then instruct the mid term schedular to suspend some of the processes so that we can recover the system from thrashing.

Belady's Anomaly in Operating System (OS)

Virtual Memory and Belady's Anomaly

- In the Virtual memory system, all the processes are divided into fixed-sized pages. These pages are loaded into the physical memory using the method of **demand paging**.
- Under demand paging, during the execution of a particular process, whenever a page is required, a page fault occurs, and then the required page gets loaded into the memory replacing some other page.
- The page replacement algorithm specifies the choice of the page which is to be replaced.
- Now, Belady's Anomaly is said to occur when the number of page faults increases significantly.

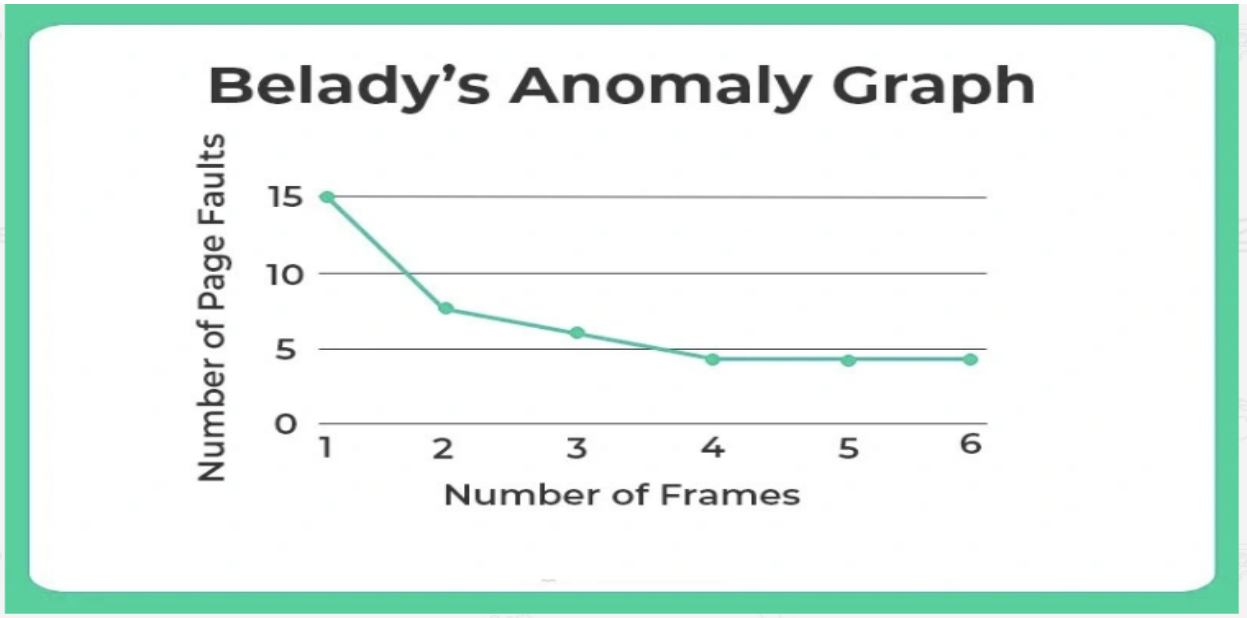
Belady's Anomaly

The rate of page faults varies directly with the number of frames allocated to the individual process. The increase in the number of frames considerably decreases the number of page faults. However, sometimes reverse action occurs when the increased number of frames results in increased page faults. This exception is known as the **Belady's Anomaly**. The occurrence of this exception depends on the **page replacement algorithm,** which governs the **demand paging** process. The page replacement algorithms in which it occurs the most includes:

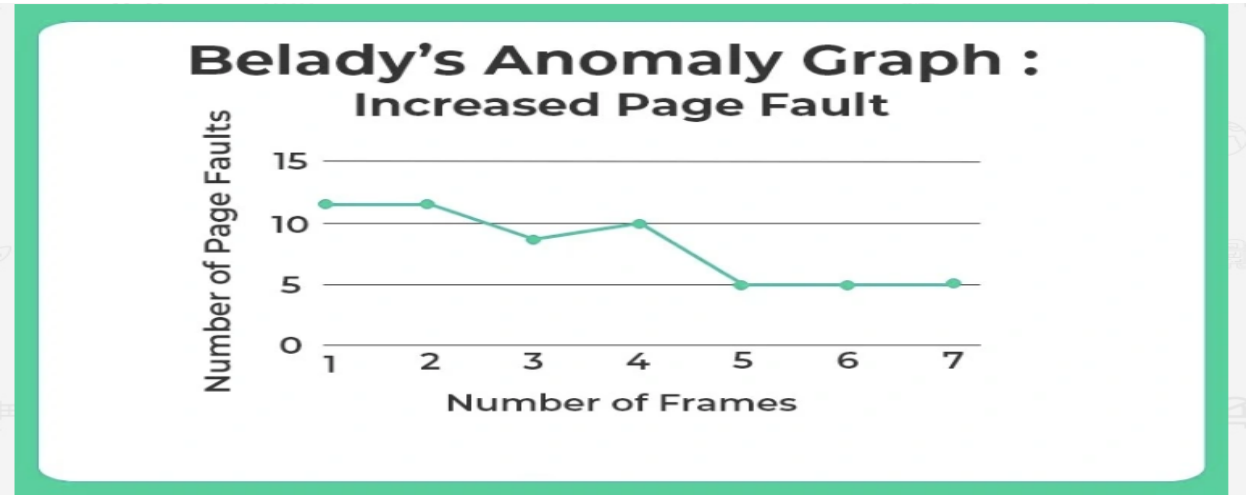
1. **First In First Out (FIFO)**
2. **Second Chance Algorithm**
3. **Random Page Replacement Algorithm**

Belady's Graph

The graph in Fig.1 symbolizes that the number of page faults is inversely proportional to the number of memory frames. In simple terms, when the number of page frames increases, the number of page faults decreases.



However, when FIFO, second chance algorithm, and Random Page Replacement Algorithm are utilized in demand paging, then Belady's anomaly is suffered by increased page fault while the expected result should be decrease in the number of page faults. Fig. 2 depicts the graph for Belady's Anomaly.



Static Vs. Dynamic Loading in Operating System (OS)

Static Loading vs Dynamic Loading in Operating System

- The process can vary depending on the loading method where the loading can take place at once, or at random time. In the case of static loading, the load process doesn't change over time, on the other hand, in the case of dynamic loading, it changes with time.
- The selection between static and dynamic loading is made by the developer who develops the operating system.

The following table represents the difference between the static and the dynamic loading.

Static Loading

In static loading the complete program is loaded to main memory before it is executed.

Dynamic Loading

To execute a process, the entire programme and all process data must be in physical memory. As a result, the size of the process is limited by the quantity of physical memory accessible.

Head-to-head Comparison between the Static and Dynamic Loading in Operating System

Static Loading	Dynamic Loading
The complete program is linked and complied without dependency of an external program.	All the modules are loaded dynamically. The developer provides a reference to all of them and the rest of the work is done at execution time.
Absolute data and program are loaded into the memory to start execution.	Loading of data and information takes bit by bit in run time.
The linker combines the object program with other object modules to make a a single program.	The linking process takes place dynamically in relocatable form. Data is loaded into the memory only when it is needed in the program.
The processing speed is faster as no files are updated during the processing time.	The processing speed is slower as files are uploaded at the time of processing.
The code may or may not be executed once it is loaded into the memory.	Execution takes place only when it is required.
Static loading is done only in the case of structured programming languages such as C.	Dynamic loading takes place in the case of object-oriented programming languages such as C++, Java, etc.

Static vs. Dynamic Linking in Operating System (OS)

Static Linking vs Dynamic Linking in Operating System

- The process of linking can take place at both compile time and load time. During compilation source code is translated into machine code, and during loading the program gets loaded into the memory.
- The process involves copying the library modules required by the program into a final executable image. The following section discusses the difference between static and dynamic linking process.

Static Linking

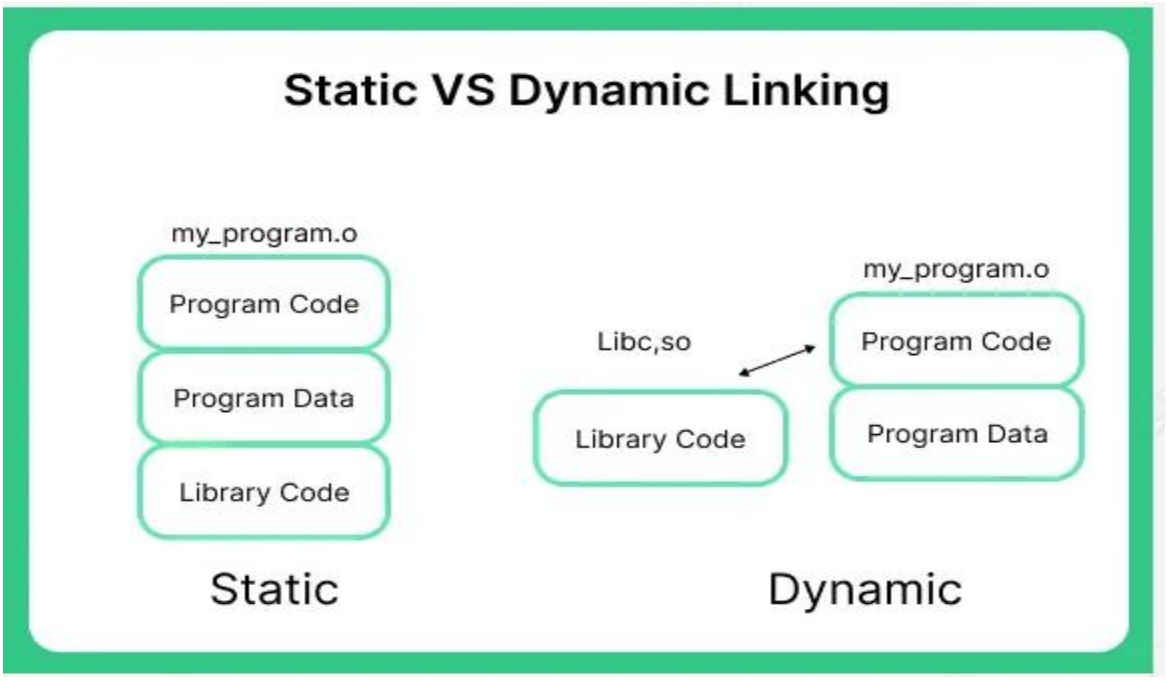
Static Linking is the process of copying all library modules used in the program into the final executable code.

Dynamic Linking

Dynamic Linking is the process of using only the name of the shared library module in the program.

Static Linking	Dynamic Linking
Files that are statically linked are larger in size as they contain external programs with their details.	Dynamically linked files are smaller.
The process is conducted by programs known as linkers. It is the last step in the compilation of a program.	Dynamic linking is performed at run time by the operating system.
If any external program changes, then it require recompilation, and relinking, else the changes will not be reflected in the existing executable files.	Individual shared modules can be updated and recompiled. The programs can be changed as many times as needed.
Programs under static linking take constant load time each time they are loaded into the memory for execution.	The load time must be minimized if the shared library code is already present in the memory.
Programs that use this method are comparatively faster as they contain all the data and information necessary for processing.	These programs are usually slower than those using the static linking library.

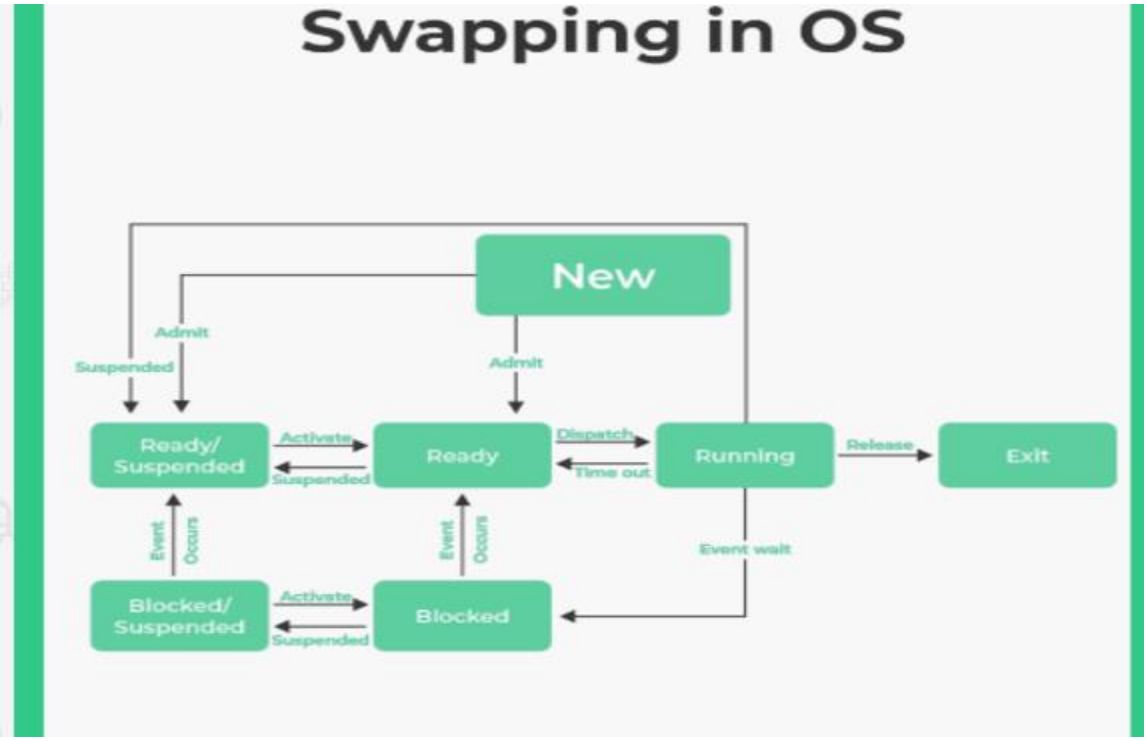
The difference can be seen in the diagram below which describes the association of library codes in the case of static as well as dynamic linking.



Swapping in Operating System (OS)

Overview

- Swapping is a simple memory or process management method used by the operating system (O.S).
- This method is dedicated to increasing the utilization of the processor by moving blocks of data in and out of the main memory. This data and information are swapped between the main and the secondary memory to optimize memory utilization and increase the processing power.
- The process of swapping also forms a queue of temporarily suspended processes as the execution carries out with the newly arrived process. The concept of swapping consists of two more concepts, swap in and swap out.
- Swap Out is the method of removing a process from the RAM and adding it to the Hard Disk. On the other hand, swap in means, removing the program from the hard disk and placing it back into the main memory or the RAM.



Advantages of Swapping

- The process helps the CPU to manage multiple processes within the same main memory.
- The method helps to create and use Virtual Memory.
- The method is economical.
- Swapping makes a CPU perform several tasks simultaneously. Hence, processes do not have to wait for too long before they are executed.

Disadvantages of Swapping

Although the process has no significant or effective disadvantage, the overall method depends heavily on the virtual memory. Such dependability results in a significant performance drop.

- In the case of heavy swapping activity, if the computer system loses power, the user might lose all the information related to the program.
- If the swapping algorithm is not good, the overall method can increase the number of page faults and decline the overall processing performance.
- Inefficiency may arise in the case where a resources or a variable is commonly used by the processes which are participating in the swapping process.

Operating System which uses Swapping

- UNIX systems use a swapping mechanism if the main memory gets The system may discontinue swapping when the load on the processor drops.
- Windows 3.1 is another operating system which uses swapping when the load on the processor is heavy.

Mobile Systems using Swapping

Mobile phone usually does not use the concept of swapping during their operation. The most popular mechanism used by them is flash memory which is more efficient than the hard drives used for persistent storage. The flash memory can only be written a limited number of times before it becomes unreliable. Thus, the method is avoided to be used in the case of mobile devices.

Translation Lookaside Buffer (TLB) in Operating System (OS)

Translation Look aside Buffer in Operating System

Translation look aside buffer in operating system overcomes the problem the occurs in paging. So, to understand the concept of translation look aside buffer which should have a knowledge about paging and it’s disadvantages that TLB overcomes.

Paging in Operating System

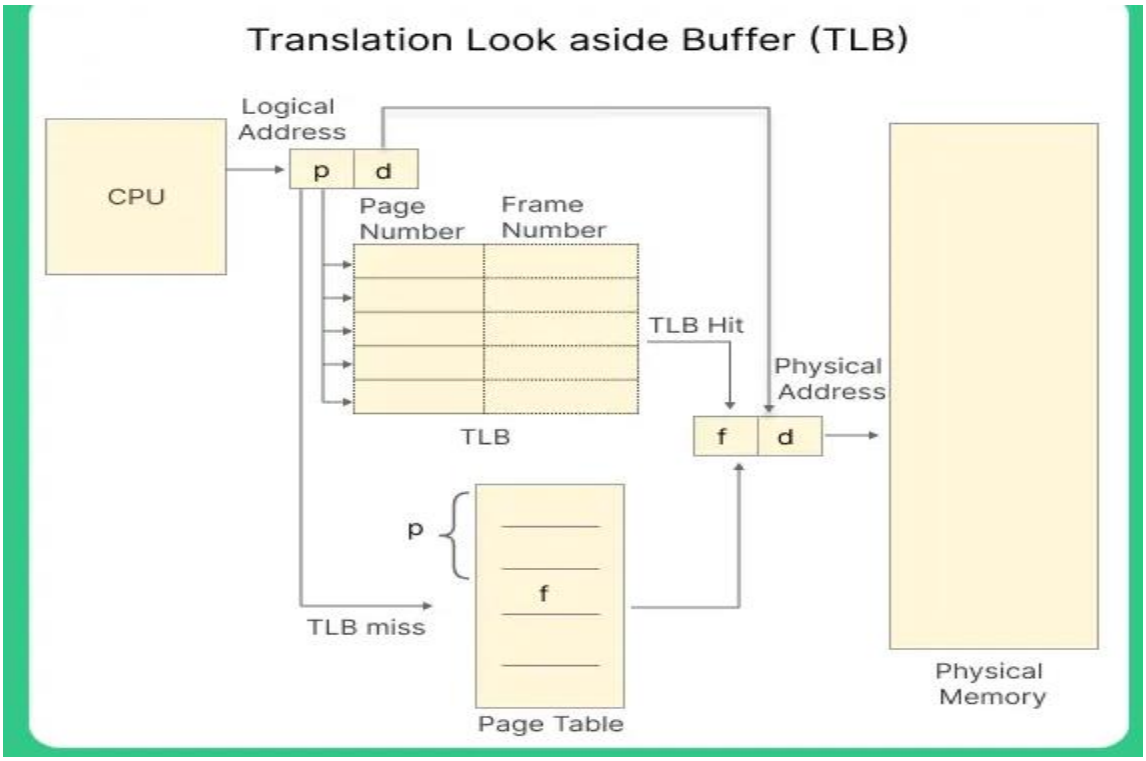
- Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous.
- Paging avoids external fragmentation and the need for compaction.
- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of same size called pages.
- When a process is executed, its pages are loaded into any available memory frames from their source file (a file system or the backing store).
- The backing store is divided into fixed-sized blocks that are the same size as the memory frames or clusters of multiple frames.

Disadvantages of Paging

- Longer memory access time.
- Internal fragmentation to every last page of the process.
- Large memory space is required as page table pre stored in main memory as well.

Translation Look aside Buffer(TLB)

1. **Translation look aside buffer (TLB)** is a special, small, fast-lookup hardware cache.
2. The **TLB** is associative, high-speed, memory.
3. Each entry in the TLB consists of two parts: **a key** (or tag) and **a value**.
4. when the **associative memory** is presented with an item, the item is compared with all keys simultaneously.
5. If the item is found, the corresponding **value field** is returned.
6. Typically, the number of entries in a **TLB is small**, often numbering between 64 and 1,0241.
7. The TLB is used with pages tables in the following way :



8. The TLB contains only a few of the **page-table** entries. When a logical address is generated by the CPU, its page number is presented to the TLB.
9. if the page number is found (known as **TLB hit**), its frame number is immediately available and is used to access memory.
10. If the page number is not found in the TLB (known as **TLB miss**) a memory reference to the page table must be made.
11. when the frame number is obtained, we can use it to access memory.
12. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the **next reference**.
13. If the TLB is already full of entries, the operating system must **select one** for replacement.
14. Replacement policies range from **least recently used** (LRU) to random.

Process Address Space in Operating System (OS)

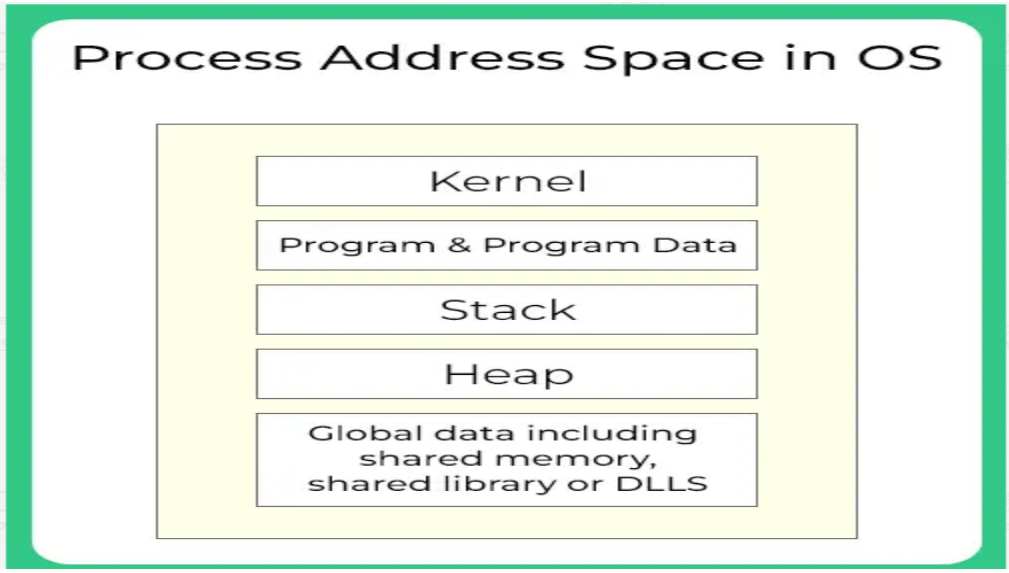
Some Background

We need memory management in an OS since, we may need to move processes back and forth in and out for primary memory (RAM) and disk memory for process execution.

While memory management will keep track of all the memory locations even the unallocated memory and especially the allocated memory.

It also has additional tasks to –

- Decide when and which process gets memory
- Decides how much memory should be allocated to a process
- Updating the status for whenever a memory gets free and updating its status



Definition of Process Address Space

Address space may also denote a range of physical or virtual addresses which can be accessed by a processor.

While a Process Address space is set of logical addresses that a process references in its code. For example, for a 32-bit address allowed, the addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers

The OS here also has an additional job to map the logical addresses to the actual physical addresses too.

We have generally three different types of addressing systems used –

Types of Addresses	Description
Physical addresses	After the compilation when the program is loaded into main memory the loader will generate mapping to physical addresses
Relative addresses	Where we compile anything, the compiler will convert symbolic addresses into some relative address
Symbolic addresses	There are mostly addresses referenced in a source code. The components can be - variable names, constants, and instruction labels.

Components of a Process Address Space

- The total amount of shared memory a system can allocate depends on several factors. The overall space may include sections such as stack space, program size required, memory mapped files, shared libraries, as well as memory allocated from the heap.
- Memory allocation policies and address spaces used by the varied operating systems are complicated. They may also differ from one operating system to another. The figure below gives an overall layout of a generic process address space for a 32-bit operating system.

More information (May not be relevant for exams)

- Each section in the given address space is dedicated to performing a specific task and is present to perform an explicit task. The most common areas are the Kernel, program and program data, stack value, heap value, and global data including shared libraries, shared memory, or Dynamic Link Libraries.

Learn about swapping here.

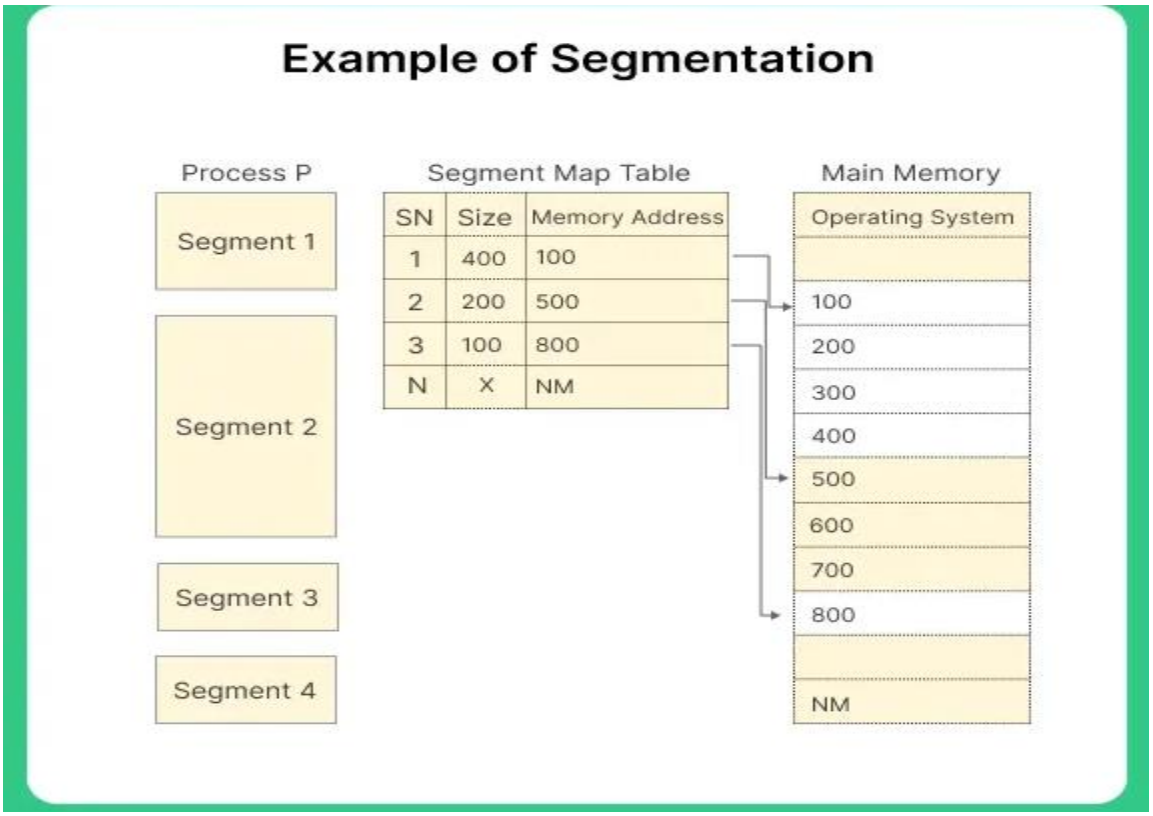
- The magnitude of each section in the address space depends on the type and power of the operating system. These sizes also affect the size of allocated segments of the shared memory. In the case of a 64-bit operating system, most of the computational limits are theoretical and may change depending on the available hardware.
- Address spaces can be further differentiated as either flat or segmented. In the case of flat address spaces, the addresses are expressed as incremental values which start from zero. In the case of segmented address spaces, the addresses are represented as distinct segments augmented by offsets. Offsets are values that are added to generate secondary addresses.
- These address spaces can be transformed in some systems from one format to another following a process known as Thunking.
- Addresses spaces have also been linked with the IP address spaces. It is said that the developers of IPv4 had not initially anticipated the exponential expansion of the internet.

For this reason, the IPv4 address spaces were updated with the IPv6 address spaces with higher bandwidth and address storage capacity.

Difference Between Segmentation and Paging in Operating System (OS)

Segmentation

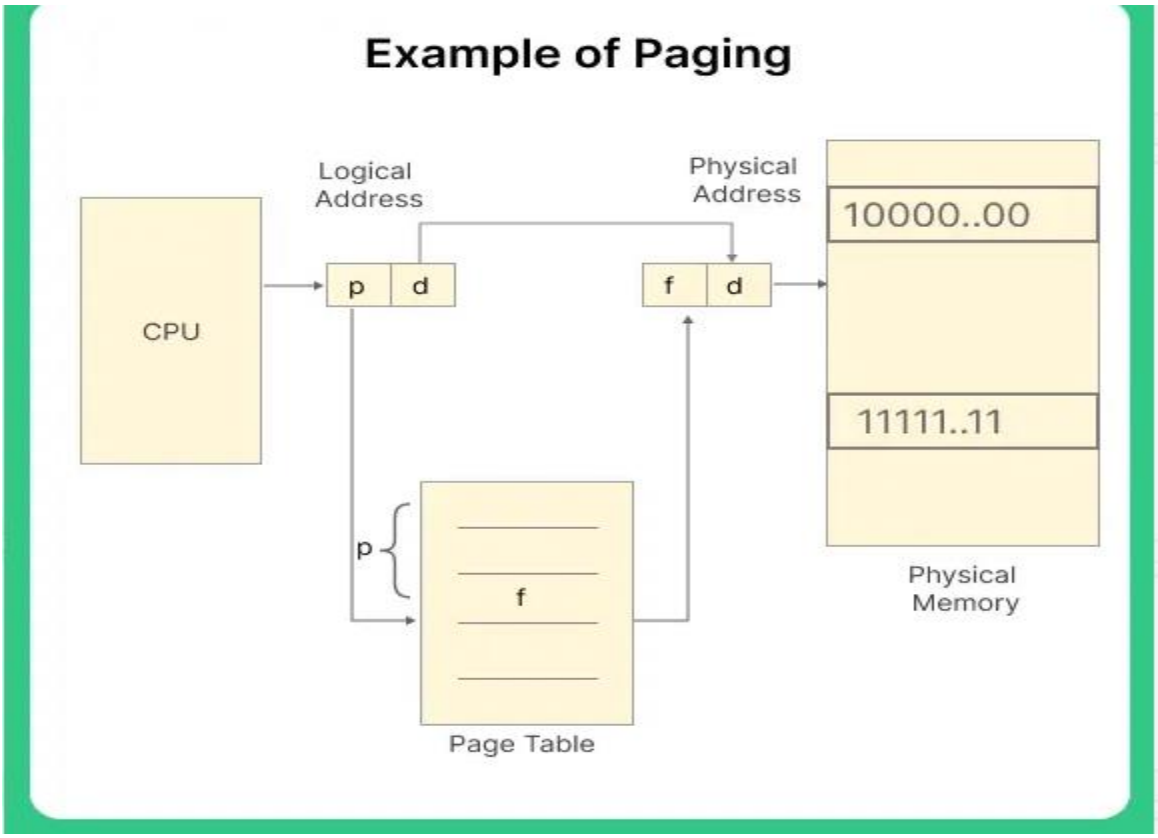
- Segmentation is a memory management technique which divides the computer's primary memory into segments or sections.
- It is a scheme in which page is of the variable block size.
- In segmentation, the process is divided into two variable size segments, and these segments are loaded into logical memory address space.
- The process is segmented module by module. However, in the main memory, the only segment of the process is a store.
- The segment table is used to keep the records of the segment, its size, and its memory address.



Paging

- Paging is also a memory management scheme.
- It allows a process to be stored in the memory in a non-contiguous manner.

- It resolves the issue of external fragmentation. In paging, the physical and logical memory spaces are divided into the same fixed-sized blocks.
- The blocks of physical memory are called frames, and the blocks of logical memory are called pages.
- When a process is executed the process pages from logical memory space are loaded into the frames of physical memory address space.
- Then the address generated by CPU for accessing the frame is divided into two parts:- page number and page offset. Page table maps logical address to the physical address.



Paging vs Segmentation

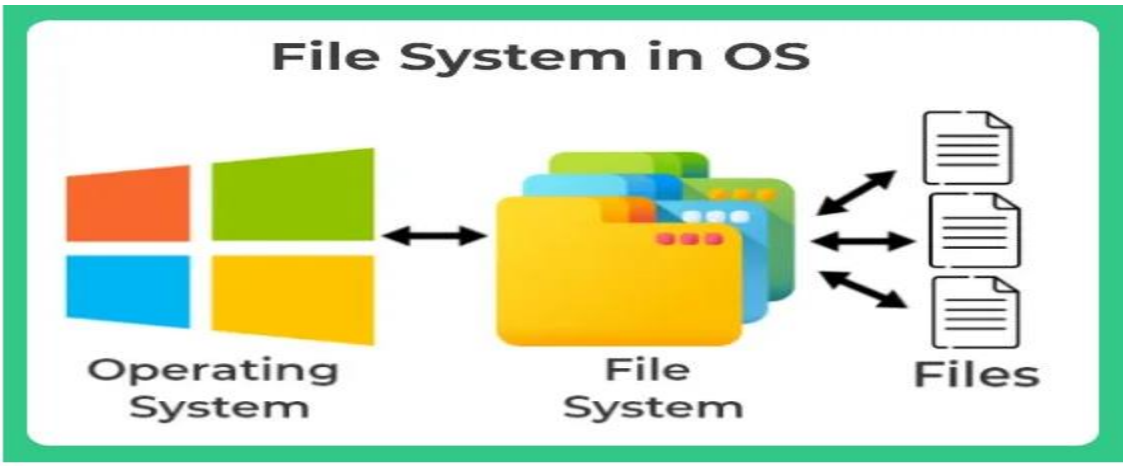
Paging	Segmentation
A page is a physical unit of information.	A segment is a logical unit of information.
Frames on main memory are required	No frames are required
The page is of the fixed block size	The page is of the variable block size
It leads to internal fragmentation	It leads to external fragmentation
The page size is decided by hardware in paging	Segment size is decided by the user in segmentation
It does not allow logical partitioning and protection of application components	It allows logical partitioning and protection of application components
Paging involves a page table that contains the base address of each page	Segmentation involves the segment table that contains the segment number and offset

File System in Operating System

File System

The file manager administers the file system by:

1. Storing the information on a device
2. Mapping the block storage to a logical view
3. Allocating/deallocating storage
4. Providing directories



File Operations:-

- Creating a file
- Writing a File:-To write a file, we make a system call specifying both the name of the file and the information to be written to the file
- Reading a File
- Repositing withing a file
- Deleting a File
- Truncating a File

Types of File

File System in OS	General Extension	Function
Archive	arc, zip, tar	Related files grouped into one compressed file
Batch	bat, sh	Commands to the command interpreter
Executable	exe, com, bin	Read to run machine language program
Multimedia	mpeg, mov, rm	For containing audio/video information
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats

File Attributes

File System in OS	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Last Modified	Date	Truncate
Protection	class	Delete

- Name – only information kept in human-readable form
- Identifier – unique tag (number) identifies file within file system
- Type – needed for systems that support different types
- Location – pointer to file location on device
- Size – current file size
- Protection – controls who can do reading, writing, executing
- Time, date, and user identification – data for protection, security, and usage monitoring

➢ Files Directories in Operating System

File Directories in Operating System

Group of files combined is known as directory. A directory contains all information about file, its attributes. The directory can be viewed as a symbol table that translates file names into their directory entries. Directory itself can be organized in many ways.

The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory. In this section, we examine several schemes for defining the logical structure of the directory system.

Types of File Directories in OS

- Single Level Directory
- Two Level Directory
- Tree Structured Directory

Single Level Directory

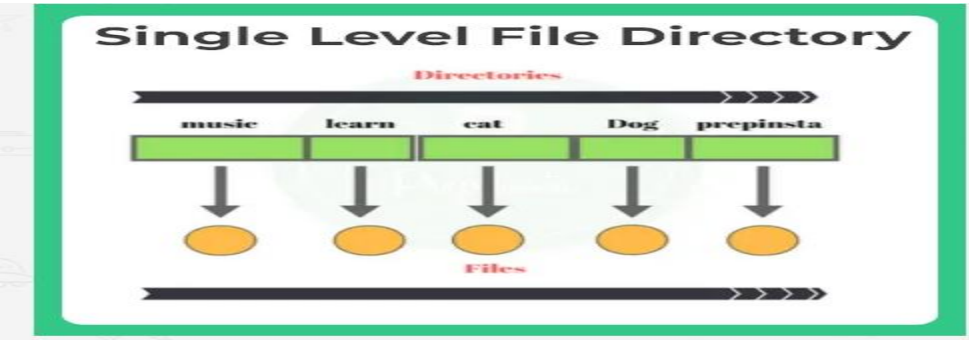
All the files belong to a single directory

Advantages

1. Easy to implement
2. Reduced Redundancy

Disadvantages

1. All files belong to one same directory
2. User can't have 2 files with same names
3. User doesn't have option to group files according to his/her needs.



Two Level Directory

In this type of directory each user has its own directory.

In the two-level directory structure, each user has his own user file directory (UFD). The UFDs have similar structures, but each lists only the files of a single user. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.

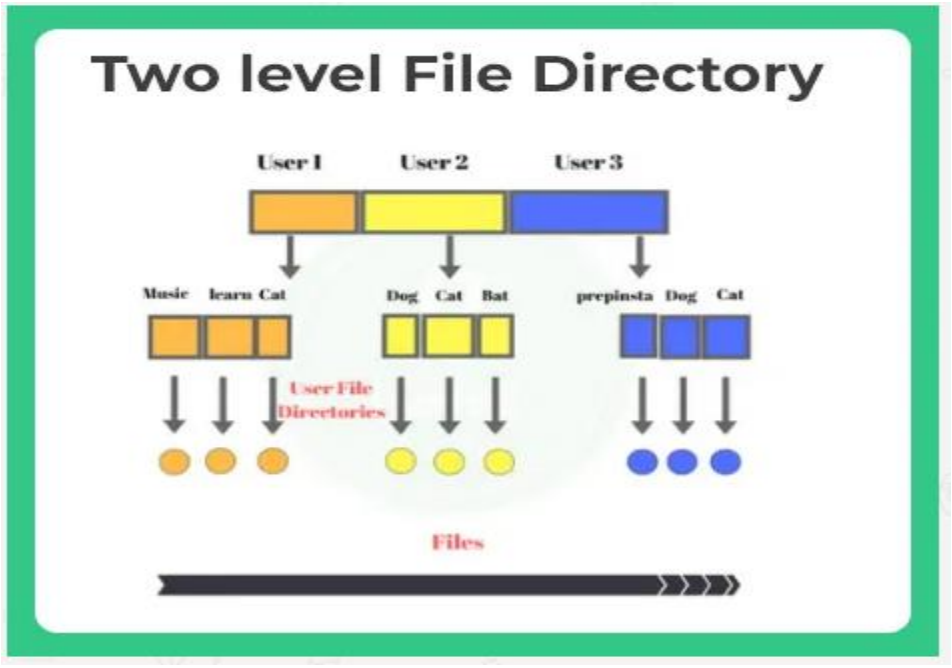
Advantages

- Sharing Directory by users problem is solved
- Efficienty
- More secure then 1 level directory
- Two users can have file with same name in their own directories.

Disadvantages

- Grouping problem is still not solved, for example if user wants to create a directory group for all files that are movies or music., he cant do that
- No two files for a single user can have same names.

Due to two levels there is a path name for every file to locate that file.

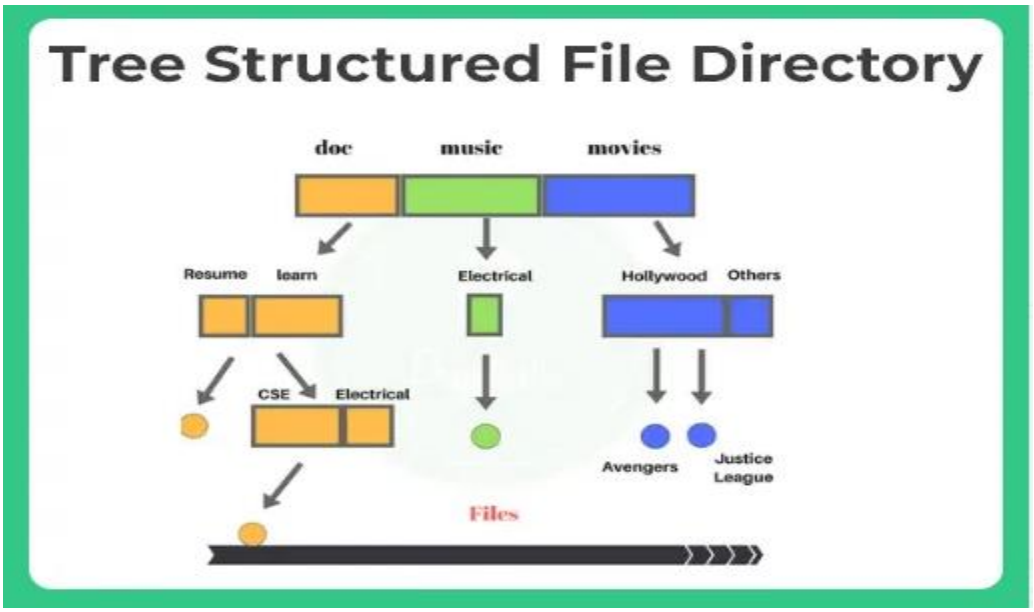


Tree-structured directory

Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.

Advantages

- Two files can have same names now, if they are in different sub directories.
- Grouping is also possible now



File Allocation Methods in OS

File Allocation Methods in OS

File allocation methods are basically ways in which any file is stored in the memory block of the system. There are three types for file allocation methods –

1.Contiguous Allocation

2.Linked List Allocation

3.Indexed Allocation

The main idea behind these methods is to provide:

1.Efficient disk space utilization.

2.Fast access to the file blocks.

File Allocation Methods

Contagious Allocation

In this method for any file in the memory block, it occupies a Contagious i.e. continuous set of memory blocks.

For example: for file F1, if the starting address is 1 and the memory blocks required by it is 3. Then it will be stored at b[1], b[2], b[3].

Each file will have a directory entry, that will have the following information –

1. Address of starting block
2. Space required by file in terms of the memory block

For example: Consider the following structure for four files – file 1, file 2, file 3.

Contagious Allocation	Size or Memory Blocks Req	Starting Address	Color in Diagram
File 1	5	1	Green
File 2	3	11	Blue
File 3	3	20	Red

Every file must occupy contiguous blocks on the storage device. The word contiguous means continuous.

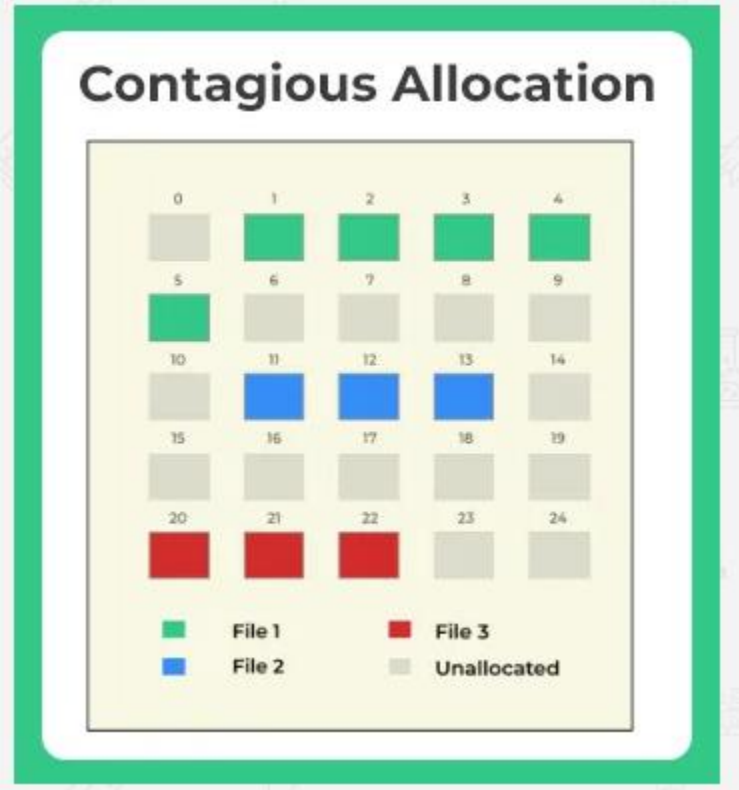
Advantages

- Since items are stored in a contiguous fashion, there is limited or no disk-head movement which reading/writing the blocks.
- Also seek time is less. Consequently, there is a great improvement in the access time of a file.
- We just need to know the starting index and length of the file to access the contiguously stored files in memory.
- For any file the kth block which starts at index i can easily be obtained as (i+k)

Disadvantages

- Memory utilization can be inefficient as such type of file allocation suffers from internal and external fragmentation.
- For large size blocks, allocation may get difficult as contiguous blocks may be scarce in the memory.

- Growing file problem may occur, initial space allocated may be sufficient but the file size may grow over time, example for word file where we are typing. Memory block currently allocated may become insufficient.
- Compaction (which is a solution for fragmentation) can take up a lot of time and may need a system to be down, wherein normal operation will not be permitted.



NoteMemory allocation for Contagious Allocation allocation usually is done with First fit and Best Fit Algorithms

Linked List Allocation

As the name suggests the use of linked lists is there –

Rather than the memory being continuous, it is scattered on available space at the disk. These are also called **chained file allocation methods in OS.**

Every directory entry will necessarily contain the pointer to the next address of the memory block the file is stored at.

There are two modifications of this –

Modification 1

The Directory entry will have the following information –

- First memory block address
- Last memory block

Modification 2

The Directory entry will have the following information –

- First memory block address
- Last memory block
- Total blocks required by file

The total block information is used to cross-check if there is any corruption in the system or not.

Modification 3

- First memory block address

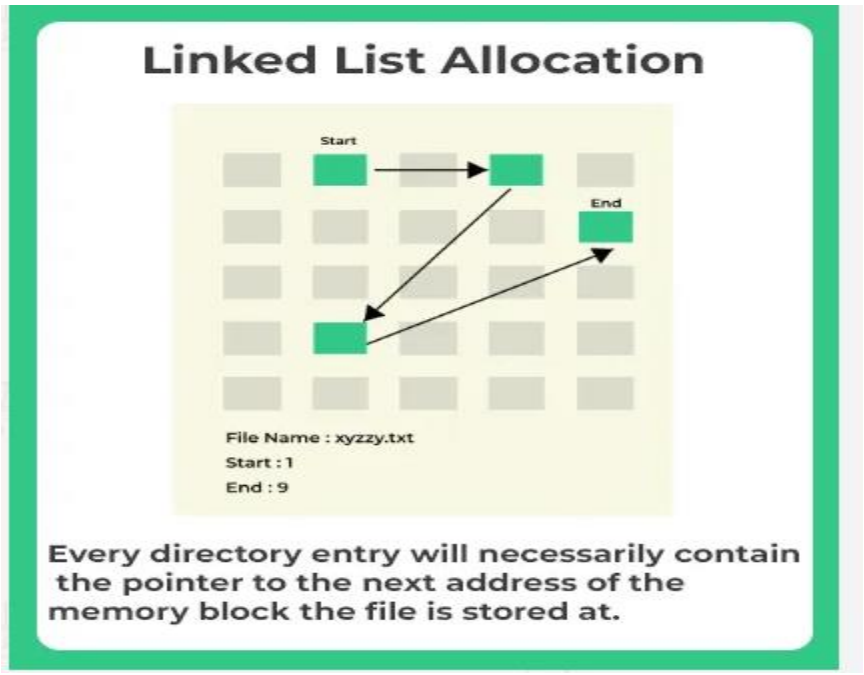
The last memory block will not have any pointer, thus indicating the end of the file.

Advantage

- Doesn’t suffer from external fragmentation as the increasing file size can be handled with adding more scatter linked memory
- File size growth is not a problem and can be handled

Disadvantage

- Extra space for a pointer
- Loss of one pointer may lead to whole file corruption
- A large number of disk seeks operations may be required and may slow down access as file blocks are scattered all over the memory
- Direct access is not supported, example: Starting index i, we can’t access kth block by doing (i+k) since it’s not stored in a contiguous manner



Indexed Allocation

ImportantThe file allocation method of choice in both Unix and Windows is the indexed allocation method. This method was championed by the Multics operating system in 1966.

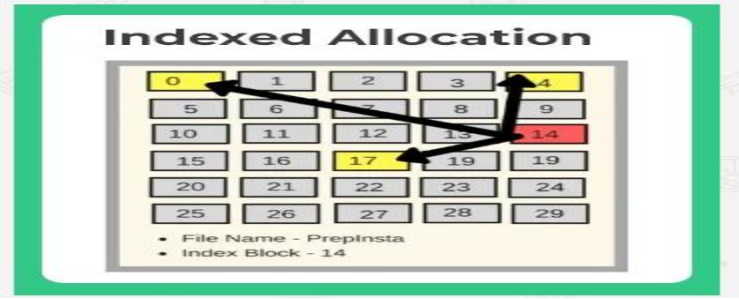
In indexed allocation method, all the pointers (pointing to the next block in the Linked list) are gathered together into one location known as Index Block.

- The file-allocation table contains a multi-level index for each file.
- Indirection blocks are introduced each time the total number of blocks “overflows” the previous index allocation.

You get all block locations in one index file. Blocks are stored in the index file in order of access.

Example –

- In the image below the index block is 14 and contains all block addresses of file jeep
- The first block storage is 0 then 4 then 17 in order.
- A negative number denotes index block list being empty. That is the file has not grown large enough to fill more blocks.



ResolutionIn Linked List Allocation method the pointers as well as the blocks were scattered across the memory and the retrieval was done by visiting each block in sequential order.

This gets resolved in Index allocation method

Highlights

For Linked Allocation the pointers along with the blocks were scattered across the disk and needed to be retrieved in order by visiting each block to access the file

Advantages

- The kth entry of the index-block is a pointer to the kth block of the file.
- All addresses can be found in one index block file
- Doesn't suffer from external fragmentation

Disadvantages

- Seek time may still be high as memory is scattered all over the disk
- Corruption in index files may lead to loss of file access locations.

Magnetic Disk Structure in OS

Magnetic Disk Structure in OS

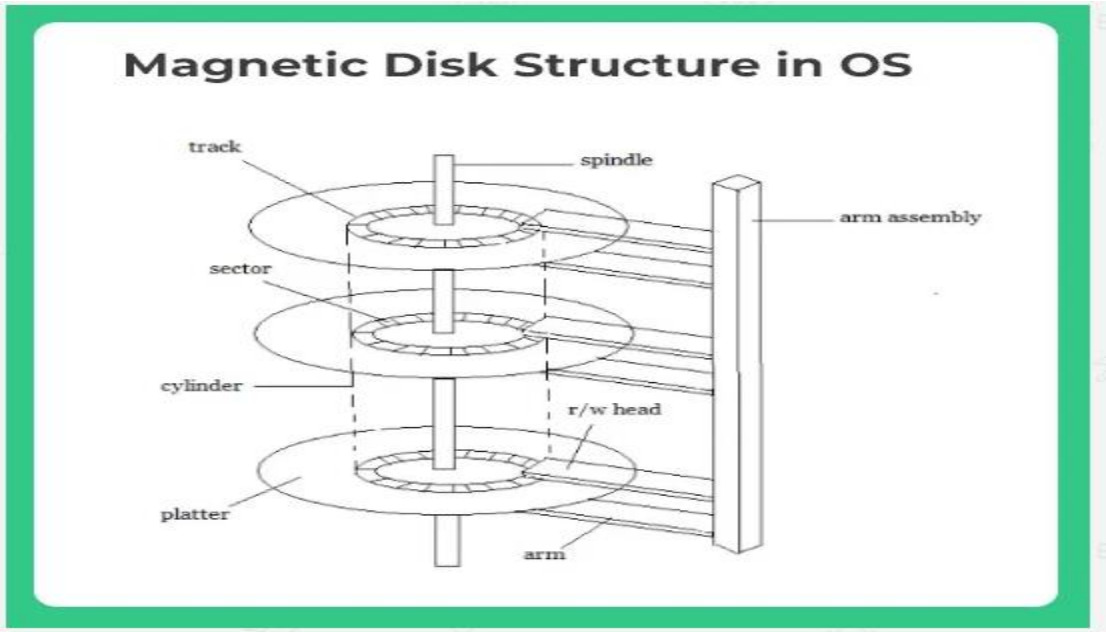
In computers the secondary storage typically is formed of stacked up magnetic disks on the top of one another.One single unit of such disk is called platter.A single unit of secondary storage device like HDD may have 100-200 such platters stacked up on one another.Each platter is divided into circular shaped tracks.

More about Magnetic Disk Structure

Note – The length of the tracks near the centre is less than the length of the tracks farther from the centre.**(Asked in AMCAT, HirePro)**

- Each track is further divided into **sectors.**
- Spindle revolves the platters and is controlled by r/w unit of OS. Some advanced spindles have capability to only revolve a particular disk and keep others intact.

- Arm Assembly is there which keeps a pointy r/w head on each disk to read of write on a particular disk.
- A world cylinder may also be used at times to refer disk stack.



The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access Time is:

Seek Time + Rotational Latency + Transfer Time

Disk Response time is the average of all the tasks starting from disk i/o request till the time when first data output from disks is sent to the system (This is given wrong on GeeksforGeeks)

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

HDD Vs SDD

SSD's are A solid-state drive (SSD) is a solid-**state storage device** that uses integrated circuit assemblies as memory to store data persistently.

They don't have disks, thus at times also known as shock resistant storage system as they don't break is hard disk falls.(Generally, as everything breaks if you know how to throw :D)

Disk Scheduling Algorithms in Operating System

Disk Scheduling Algorithms in Operating System

As we all know a hard disk(typically found in a computer) is a collection of disks placed on the top of one another. A typical hard disk may have 100-200 such disks stacked.

Now to have the best seek time or reduce seek time of an hard disk, we have different types of algorithms.

Types of Disk Scheduling Algorithms

1. First-Come, First-Served (FCFS)

- **Description**: Processes requests in the order they arrive in the queue.
- **Advantages**:
  - Simple and fair.
  - No starvation (all requests are served).
- **Disadvantages**:
  - Can lead to high seek time if requests are not near each other.
  - Poor overall performance compared to other algorithms.

---

2. Shortest Seek Time First (SSTF)

- **Description**: Selects the request closest to the current disk head position.
- **Advantages**:
  - Reduces average seek time compared to FCFS.
- **Disadvantages**:
  - Can cause starvation for requests that are far away.

---

3. SCAN (Elevator Algorithm)

- **Description**: Moves the disk arm in one direction, fulfilling requests along the way, and then reverses direction at the end of the disk.

- **Advantages**:
  - Ensures fairness by serving all requests in both directions.
  - Reduces starvation compared to SSTF.
- **Disadvantages**:
  - Still results in longer wait times for requests at the ends of the disk.

---

4. C-SCAN (Circular SCAN)

- **Description**: Similar to SCAN, but instead of reversing direction, the disk arm jumps to the beginning when it reaches the end and processes requests in one direction only.
- **Advantages**:
  - Provides more uniform wait time compared to SCAN.
- **Disadvantages**:
  - Increased seek time due to the jump to the beginning of the disk.

---

5. LOOK

- **Description**: A variation of SCAN, but the disk arm only goes as far as the last request in each direction before reversing.
- **Advantages**:
  - Saves time by avoiding unnecessary traversal.
- **Disadvantages**:
  - Slightly more complex implementation than SCAN.

---

6. C-LOOK (Circular LOOK)

- **Description**: A variation of C-SCAN, where the disk arm moves only as far as the last request in one direction and then jumps to the beginning of the next set of requests.
- **Advantages**:
  - Further reduces unnecessary movement compared to C-SCAN.
  - Provides uniform performance for all requests.
- **Disadvantages**:
  - Increased complexity compared to SCAN or LOOK.

---

7. Priority Scheduling

- **Description**: Processes requests based on priority rather than position.
- **Advantages**:

    - Suitable for time-critical operations.
- **Disadvantages**:
    - Low-priority requests may starve if high-priority requests dominate.

---

8. N-Step SCAN

- **Description**: Divides requests into batches of a fixed size and processes each batch sequentially using SCAN.
- **Advantages**:
    - Provides better control over request handling in high-load scenarios.
- **Disadvantages**:
    - May delay requests if they arrive after the current batch is being processed.

---

9. FSCAN

- **Description**: Uses two queues—one for new requests and one for current requests. While the current queue is being processed, new requests are added to the other queue.
- **Advantages**:
    - Prevents starvation by isolating new requests from current processing.
- **Disadvantages**:
    - Slightly higher memory overhead due to maintaining two queues.