# Syllabus

# Operating system

An operating system is a software program that acts as the intermediary between the user and the computer hardware. It provides a platform for other software applications to run on, and manages the various resources of the computer, such as the CPU, memory, input/output devices, and storage

## What are the Key features of Operating System ?

- Process management : This involves managing the various processes or programs that are running on the computer, and allocating system resources (such as CPU time and memory) to them.
- Memory management : This involves managing the computer's memory, including allocating memory to running programs, freeing up memory when it's no longer needed, and managing virtual memory.
- Device management : This involves managing the various input/output devices attached to the computer, such as keyboards, mice, printers, and disk drives.
- File management : This involves managing the computer's file system, including creating, deleting, and moving files and directories, and managing access to files and directories.
- Security : This involves protecting the computer system from unauthorized access, viruses, and other threats.
- Accounting :
    - Which resources are used by which user
- Error detection :
    - Memory Hardware( Power failure, memory error)
    - I/O devices (Parity error on tape, connection failure)
    - User program (Arithmetic error)

## What are different types of Operating Systems ?

- Batch Operating System: This type of operating system is designed to process large volumes of similar jobs in batches, without the need for user interaction. An example of a batch operating system is the IBM OS/360.

- Time-Sharing (Multitasking) Operating System: This type of operating system allows multiple users to access a computer system simultaneously, with the system rapidly switching between users to give each user the illusion of having their own dedicated system. An example of a time-sharing operating system is Unix.
- Distributed Operating System: This type of operating system allows multiple machines to work together as a single system, sharing resources and processing power. An example of a distributed operating system is the Amoeba operating system.
- Network Operating System: This type of operating system is designed to manage network resources, such as servers, printers, and other devices, and to provide file and print sharing services to multiple users. An example of a network operating system is Windows Server.
- Real-Time Operating System: This type of operating system is designed to respond to events in real-time, such as in control systems, robotics, or military applications. An example of a real-time operating system is QNX.

## Example of NOS

- Routers : Routers are networking devices that connect multiple networks together. They typically have their own operating systems known as network operating systems (NOS). Examples include Cisco IOS (Internetwork Operating System) and Juniper Junos.
- Network Cameras : Network cameras, also known as IP cameras, capture and transmit video over a network. They often have embedded operating systems that handle video encoding, streaming, and management. Examples include Axis Communications and Hikvision.

## Real-Time Operating System

- Soft real-time systems provide no guarantee as to when a critical real-time process will be scheduled. They guarantee only that the process will be given preference over noncritical processes.
- Hard real-time systems have stricter requirements. A task must be serviced by its deadline

- Supports priority based algorithms
- No user interface

**Time shared program example Compiler**

**How can we implement the compiler to work as the time shared program?**

- The time shared system the CPU time is shared among the number of processes by means of the time slice.
- Now the compiler can be implemented as the time shared program by breaking the process into the smaller discrete tasks
- This can executed in time slice
- When CPU returns the compiler can work again

**Command Interface in Operating System**

➔ Command interpreter in kernel

➔ Two types of the interface:

- ◆ Commands in the interpreter
  - Commands stored in the interface itself
  - Whenever the command is entered the results are affected from the command interface
  - Less scalable
  - Command interface size is based on the number of commands stored.
- ◆ File System
  - Commands are stored in the file system and whenever the command are entered they are fetched from the file system
  - Used in linux
  - Easily customizable
  - Easy to maintain
  - Secure

## Network Operating System and Distributed Operating System

1. Network operating system
    - Focuses on managing and coordinating network resources, such as file sharing, printer sharing, and user authentication, within a local area network (LAN) or wide area network (WAN).
    - Example - Windows server
2. Distributed operating system
    - Operating system that runs on multiple machines and enables them to work together as a single integrated system.
    - It extends the capabilities of a single computer operating system to a network of interconnected computers, allowing them to share resources, collaborate on tasks, and communicate with each other transparently
    - Example - Amoeba

## Can you explain the difference between a kernel and an operating system ?

- An operating system (OS) acts as an interface between computer hardware and software applications, and provides a platform for running applications.
- On the other hand, a kernel is the heart of an operating system that provides low-level services to other parts of the operating system and to applications. It is the core of the operating system.

## BIOS

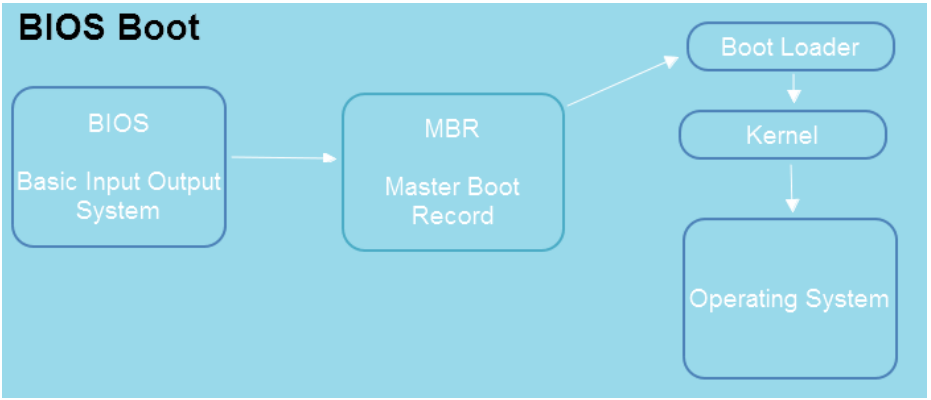
- Stands for (Basic Input Output System)
- It is a firmware (give permanent instructions to hardware device,stored in flash ROM,can be erased and rewritten)
- Helps in hardware initialization during the booting process

## How does BIOS boot the system?

In order to boot the system following procedure is followed

➔ First the POST (Power on Self Test) is called and it Checks for the BIOS
➔ Then the BIOS check for the CMOS-RAM(it saves the BIOS settings in it)
➔ After this the hardware components are checked

➔ BIOS then check for MBR(it stores the information in first sector of any hard disk and helps in finding the location of OS storage)(First sector is always of 512 Bytes)
➔ Finally BOOTLOADER is retrieved and the RAM is assigned to the OS
➔ OS is booted




**WHAT IS MBR?**

- It is a piece of code which resides in Boot Sector of storage device
- The primary purpose of the MBR is to bootstrap the operating system. When the computer is powered on, the system BIOS or UEFI firmware loads and executes the code stored in the MBR.

**UEFI**

- UEFI stands for Unified Extensible Firmware Interface
- UEFI is a low level software that help in the booting the cpu
- (acc to a report INTEL has announced to remove all BIOS chips with UEFI by 2020)

**How is UEFI different from bios ?**

a) Support larger drives(2.2TB -9.4 ZB)
b) Faster booting times
c) Enhanced security
d) Better UI

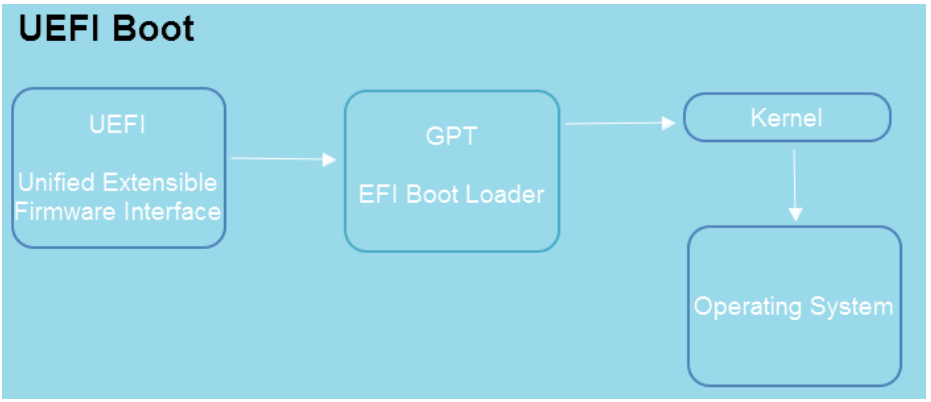
- It maintains a list of valid boot volumes called EFI service partition instead of using MBR it uses GPT (GUILD PARTITION TABLE)

### What is GPT ?

- It is a standard for the layout of partition tables used in storage devices GPT uses logical block addressing [LBA] instead of the historical cylinder-head-sector. [CHS]
- GPT does not have a bootloader
- GPT is not constrained by temporary schemes such as container partitions

### How does UEFI boot the system ?

- It scans the GPT to find a EFI(Extensible Firmware Interface)(partition on storage device)
- After scanning it directly loads the OS from the right positions
- in case it is not able to find the correct position it goes for LEGACY BOOT (booting by BIOS)




### What is a bootstrap program / bootloader ?

- The Bootstrap program, also known as the bootstrap loader or bootloader, is a small piece of code that initiates the booting process of an operating system.
- It is typically stored in the computer's firmware or a dedicated boot device, such as the Master Boot Record (MBR) on a hard disk.

**What happens when you press the power button ?**

1. Power-On Self-Test (POST) : When a computer is turned on, it goes through a series of tests called the Power-On Self-Test (POST). The POST checks the hardware components such as the CPU, memory, storage devices, and input/output devices to ensure that they are working properly.
2. Boot Loader : After the POST is completed, the BIOS (Basic Input/Output System) loads the boot loader program from the boot device, typically the hard disk. The boot loader program is responsible for finding and loading the operating system into memory. It does this by searching for the operating system kernel on the boot device.
3. Kernel Loading : Once the boot loader program has found the kernel, it loads it into memory. The kernel is the core of the operating system that manages system resources and provides essential services to other parts of the operating system and to applications.
4. System Initialization : After the kernel is loaded into memory, it performs system initialization to prepare the computer for use. This includes initializing hardware devices, configuring system settings, and loading device drivers.
5. Launching System Services : Once the system initialization is completed, the kernel launches system services such as network services, file system services, and device drivers. These services are responsible for managing computer resources and providing a platform for applications to run and interact with users and other systems.
6. User Interface : Finally, the operating system provides a user interface that allows users to interact with the computer. This may be a command-line interface, graphical user interface (GUI), or a web-based interface.

**Can you explain the difference between monolithic and microkernel architectures ?**

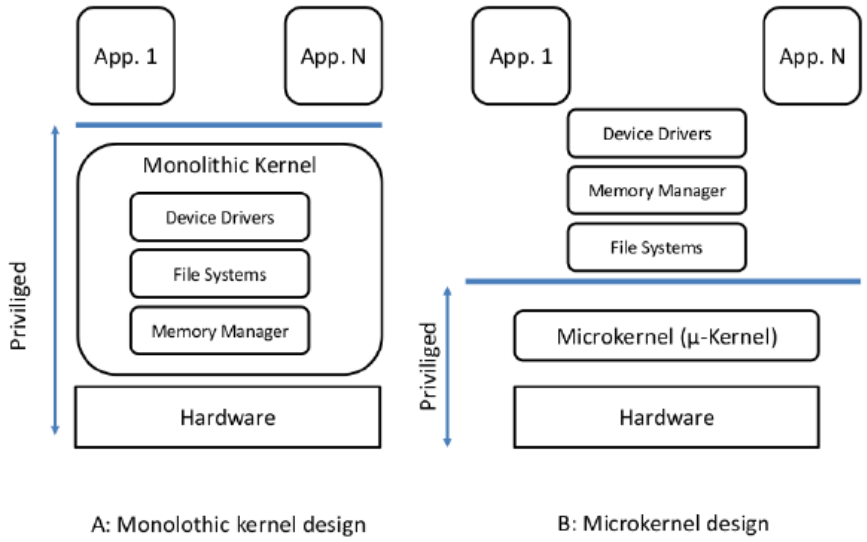
*Monolithic Architecture*:

- Monolithic architecture is an older and more traditional approach to designing an operating system.
- In a monolithic architecture, the entire operating system is implemented as a single, large binary executable file.
- All operating system services, including device drivers, file systems, and process management, are implemented as part of the monolithic kernel.
- This approach offers high performance since system calls can be executed directly without the overhead of inter-process communication.
- However, this approach can also lead to issues with reliability and maintainability, since a single bug in the kernel can bring down the entire system.
- Eg - Linux

*Microkernel Architecture*:

- Microkernel architecture is a newer approach to designing an operating system.
- In a microkernel architecture, the operating system is divided into a small, minimalist kernel that provides only basic services such as thread scheduling and inter-process communication, and a set of separate user-space services that implement higher-level functionality such as device drivers and file systems.
- The microkernel provides a message-passing interface that allows user-space services to communicate with each other and with the kernel.
- This approach offers better reliability and maintainability since a bug in a user-space service will not bring down the entire system.

- However, this approach can lead to lower performance since system calls must be executed through the message-passing interface, which introduces additional overhead.
- Eg - MacOS




A: Monolothic kernel design

B: Microkernel design

### Is windows microkernel ?

No, Windows is not a microkernel-based operating system. Windows uses a hybrid kernel architecture that combines aspects of both monolithic and microkernel designs

Monolithic part includes : a large portion of the operating system, including device drivers, file systems, and process management

Microkernel includes : graphics subsystem and some device drivers that are implemented in user-mode processes that communicate with the kernel through system calls and other interfaces, which is similar to a microkernel design.

### Modules

- This is method is used in many system
- First the important kernel modules are loaded such related to the communication and core components

- Then dynamically based on the usage the other are loaded
- Efficient no use of message passing
- Used in Linux, Unix, MacOs
- MacOS is hybrid has the BSD kernel as well as the Mash Kernel
- The dynamically loaded kernel is known as the kernel extension in MacOs X

*How does I/O operations work*

★ The device driver loads the appropriate register into the device controller
★ The device controller performs further action based on the register
★ Once the operation completes the device driver notifies the operating system and the os starts working. But it may cost if the device is a keyboard which triggers the many actions
★ Hence most of the system uses the DMA (Direct Memory Access)
★ In the CPU grant the I/O to read and write without involvement.
★ Hence it notifies at the start and at the end to the CPU and side by side CPU does its work
★ It needs the special device call the DMA controller

**What is Trap in Operating System ?**

- The operating system is interrupt driven.
- Trap : It is software generated interrupt caused by either by an error or by the special request from the user to the operating system services
- Each interrupt in the operating system has the subroutine to handle
- To protect against the logical error in the program the trap is used

**What are the modes of Operating System ?**

- Operating System has the two modes:
    - User mode (1) : Which has process and threads of the user level or application level

- Kernel mode (0) : Which has the process and threads of the operating system level

*[TIP] The mode bit define the mode of the operating system*

**How does the OS operate and switch privileged mode ?**

When the user requests any system call the OS jumps to the kernel mode and executes the services needed and again jumps back to the user mode.

It is used to protect the privileged instructions from executing in the user mode. IF the user try to access the privileged mode operations in the user mode then the trap or exception is thrown

**Timer:**

- To control the CPU, OS uses the timer
- To prevent the user programs running for too long it assigns the timer
- Ex : 7 minutes (420 seconds)
- After every time the interrupt occurs it decrement the timer
- When timer become negative the control return to the OS
- This method is used to control the amount of resources used by the user program
- It is widely used in linux, windows and MacOs

**What is cache and cache coherency ?**

Cache :

- ➔ The one of the fastest storage device is known as the cache
- ➔ The information from the disk storage is copied to the cache
- ➔ IT is volatile memory
- ➔ Size is less than the main memory
- ➔ Whenever the particular piece of the information need for the operating system it checks the
  - ◆ Cache
  - ◆ Main memory
  - ◆ Hard disk

➔ Also the internal register such as the index register provide the high speed cache to the main memory
➔ Many register allocation and deallocation algorithms are used for the register management
➔ Since cache has the limited size cache management is important
➔ Single data has the multiple copies in the multiple level
➔ In case of the multiprocess system different devices cache may contain the same data
➔ IF data updated in one cache then the data need to be updated in all others
➔ This is known as the cache coherency

**What are Escalated privileges ?**

- Sometimes user need to access the resources beyond its privileges that can be given using the escalated privileges
- Example : In linux the sudo used to give the escalade privileges

# Process Management

## Process

1. A process is an instance of a program in execution in an operating system.
2. It is a unit of work that can be scheduled and executed by the operating system.
3. The operating system maintains a process control block (PCB) for each process, which contains information about the process such as its current state, priority, and resource usage.

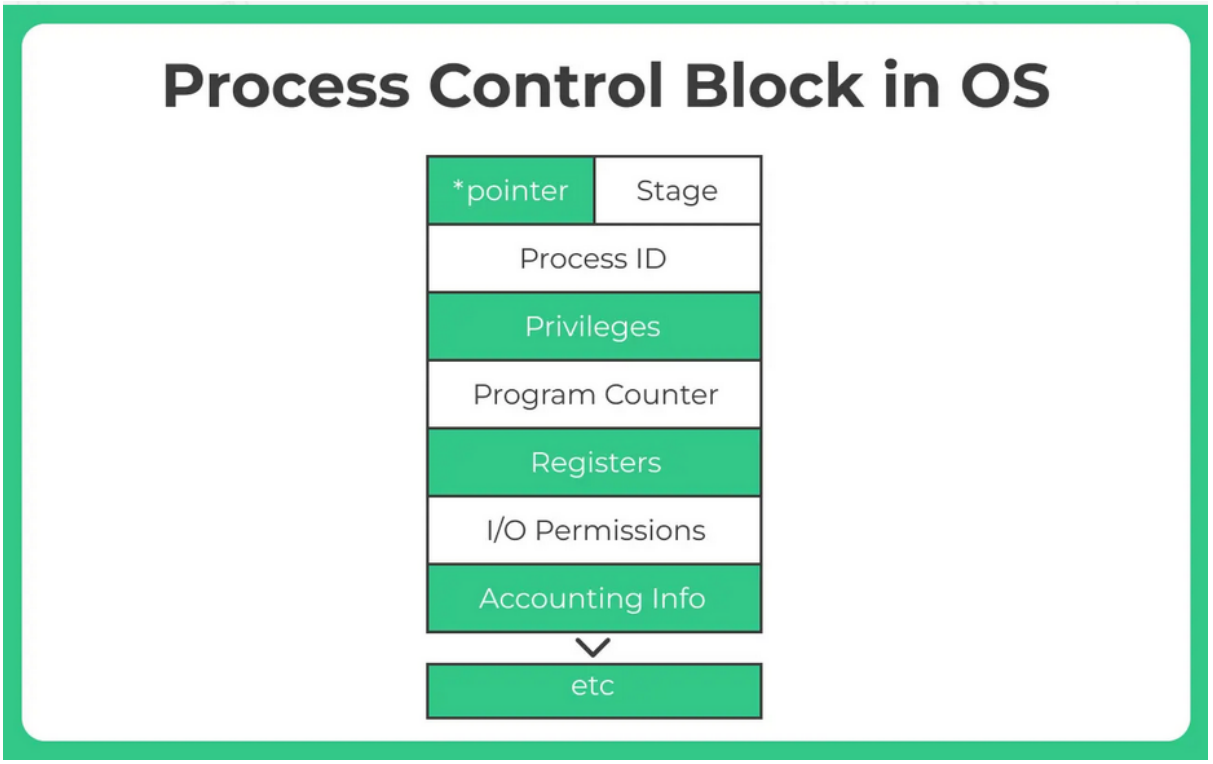
*[TIP] If the process exists without its parent it is known as the orphan process. For such a process init becomes the parent and once execution is completed it will be terminated.*

When child process is created there are two possibilities :

1. Parent execute concurrently with the child
2. Parent waits until the child executes

## Process Termination

A process terminates after calling the exit()
Zombie process : The process who has been terminated but still has an entry known as zombie process.




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 suggests 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.

*Real life Scenario*

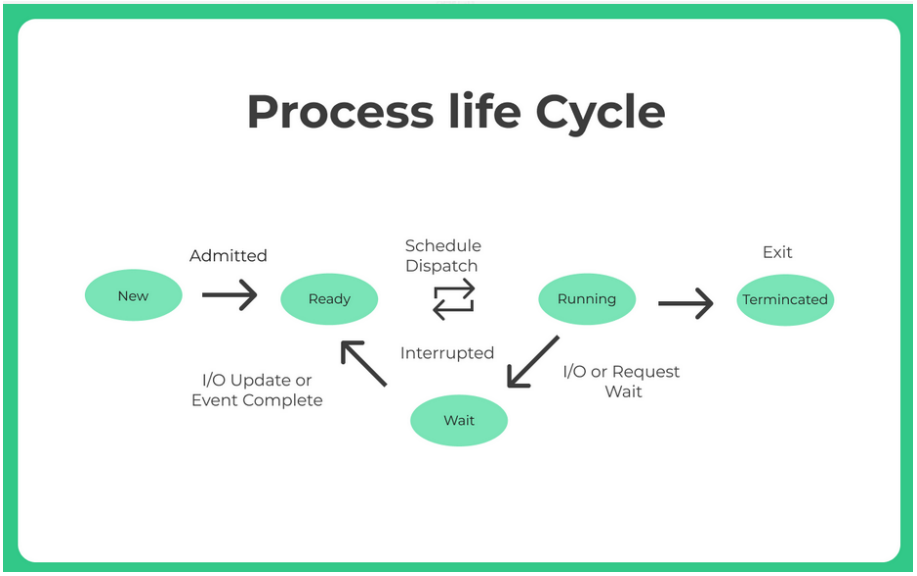
Let us consider we write a program to play a sound, when we execute this program this program creates a process for the operating system. The operating system now queues up the command to implement and music is played. Simple !!!

Well, it's 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 detail.

### What are the different phases of a process

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 meantime
2. Ready – It gets assigned to the 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 the processor, such a queue is called the Ready Queue.
3. Running – The process being executed by the processor gets into the running stage.
4. Waiting – Sometimes, cases happen when a process has to accept an additional input from the user or maybe a higher priority process needs processor, in such cases the process goes to the waiting stage or waiting queue where all the processes are waiting for the processor and complete their execution.
5. Terminated – When a process has completed executing its all instructions or when it's ended by the user directly it goes to the terminated stage.




## System calls

➔ Kernel-Level System Calls:
- Executed in kernel mode.
- Provide access to privileged operations and system resources.
- Involve a transition from user mode to kernel mode.
- Examples: open(), read(), write(), fork(), exec().

➔ User-Level System Calls:
- Executed in user mode.
- Provide access to a subset of system services and operations.
- Involve communication with the operating system through libraries.
- Examples: printf(), scanf(), fopen(), fclose(), malloc(), free().

Kernel level system calls

➔ Provides a way to user level processes to interact with underlying hardware.

➔ Common one
- open() - Opens a file specified by the path and returns a file descriptor.
- read() - Reads data from a file descriptor into a buffer.
- write() - Writes data from a buffer to a file descriptor.
- close() - Closes a file descriptor.

- fork() - Creates a new child process by duplicating the calling process.
- exec() - Replaces the current process with a new process.
- wait() - Suspends execution of the calling process until a child process exits.
- exit() - Terminates the calling process and returns a status code.

### Can we Directly execute system calls ?

Yes, We have to write corresponding assembly code for the same

Here are some steps to do so :

- Load the system call number into a register. The system call number identifies which system call you want to execute.
- Load the arguments for the system call into the appropriate registers or memory locations, according to the calling convention of the operating system and architecture.
- Trigger a software interrupt, which will transfer control to the operating system kernel.
- The kernel will use the system call number to dispatch the appropriate system call handler, which will execute the requested operation on behalf of the calling process.
- The kernel will return the result of the system call to the calling process.
- The calling process can then use the result of the system call as needed

### System programs

- System programs provide a convenient environment for programs to execute.
- They are the user interface to the system calls

### Why do we need system programs when we already have the system calls?

- Basically the system calls are at lowest level
- System programs provide efficiency they provide more information and combine many system calls to perform specific tasks

- Abstract the way the system is implemented
- Used for the hardware device management

Implementation

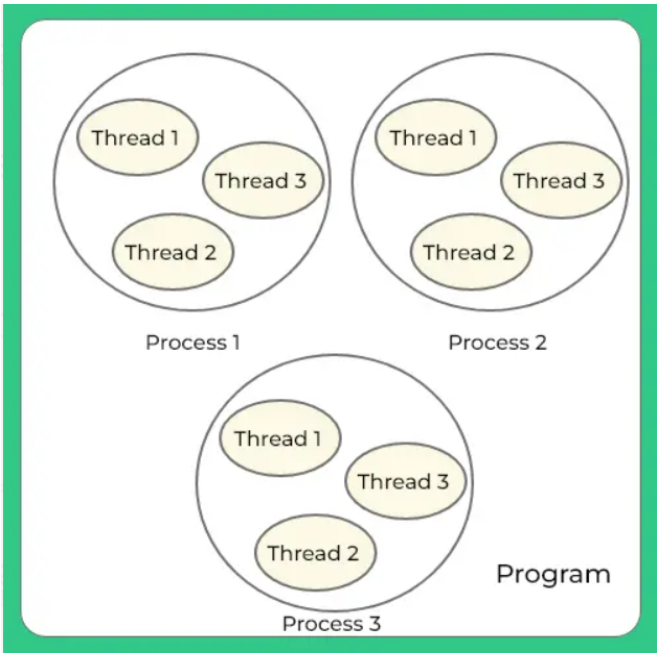
1. Use high level language such as the C or C++
2. Can be easy to integrate with the application programs and if the performance bottleneck the change that part to the assembly code
3. Linux is written in the C language
4. Advantage of the high level language is easily portable, efficient

## What is a thread?

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



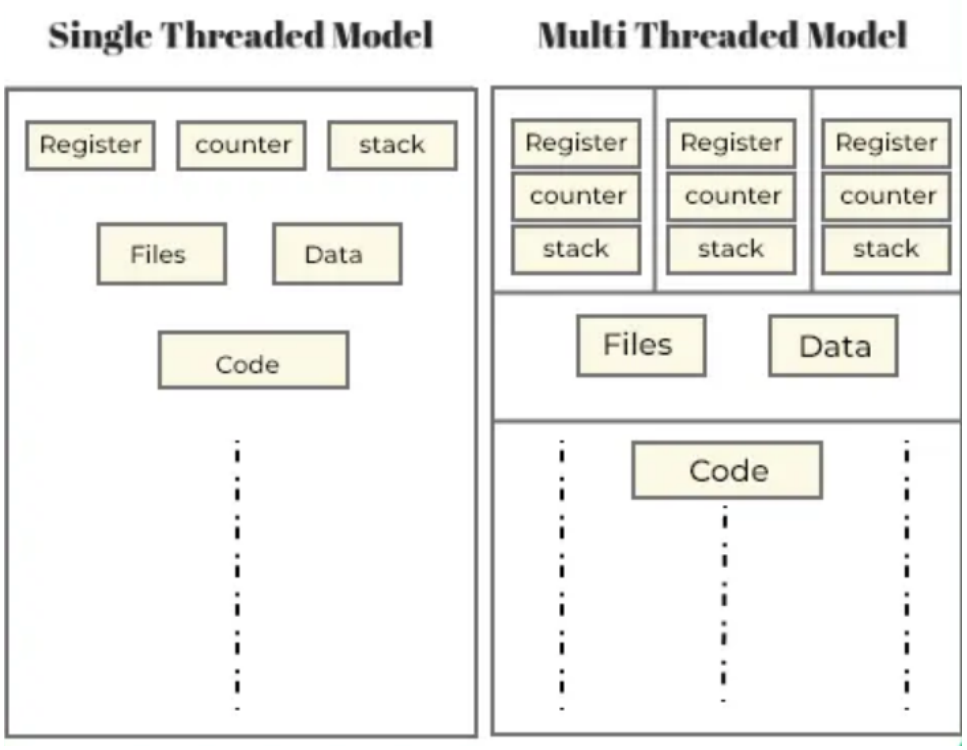

**What is the difference between User Level and Kernel Level Threads ?**

➔ Features of User Level Thread :

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

➔ Features of Kernel Level Thread :

- Managed by the operating system kernel directly.
- Thread creation, scheduling, and management are handled by the kernel.
- Threads are independent entities with their own kernel-level data structures.
- Kernel-level threads can run concurrently on multiple processors/cores.
- Synchronization and communication between kernel-level threads often involve system calls.




**Explain Thread Vs. Process**

- Processes are independent whereas threads exist as subsets of process.
- Processes have separate address space whereas Threads within a same process have the same address space.
- Processes are resource intensive whereas Threads are lightweight processes.
- No other process changes data of another process. They are independent, whereas one thread can read, write or change another thread.

| Process | Thread |
|---|---|
| Process means any program is in execution. | Thread means a segment of a process. |
| It takes more time for creation. | It takes less time for creation. |
| It also takes more time for context switching. | It takes less time for context switching. |
| The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. |
| Multiprogramming holds the concepts of multi-process. | We don't need multi programs in action for multiple threads because a single process consists of multiple threads. |
| The process is isolated. | Thread shares memory. |
| The process is called the heavyweight process. | A Thread is lightweight as each thread in a process shares code, data, and resources. |
| Process switching uses an interface in an operating system. | Thread switching does not require calling an operating system and causes an interrupt to the kernel. |
| If one process is blocked then it will not affect the execution of other processes. | If a user-level thread is blocked, then all other user-level threads are blocked. |
| The process has its own Process Control Block, Stack, and Address Space. | Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space. |
| Changes to the parent process do not affect child processes. | Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process. |
| A system call is involved in it. | No system call is involved, it is created using APIs. |
| The process does not share data with each other. | Threads share data with each other. |

★ If one process is blocked then it will not affect the execution of other processes. If a user-level thread is blocked, then all other user-level threads are blocked :

Consider a scenario where a person is using a computer to perform multiple tasks simultaneously. Let's say the person has opened a word processing software to type a document, a web browser to surf the internet, and a media player to listen to music. These three tasks are being executed by three different processes in the operating system.

Now, suppose the web browser process gets blocked because it's waiting for a response from a slow website. In this case, the other two processes (word processing and media player) will continue to execute normally, as they are not dependent on the web browser process.

if the media player process has two user-level threads, one for playing the music and another for displaying the playlist, then both threads will get blocked if the file read operation blocks the media player thread.

Multithreaded environment in chrome browser

***MULTIPROCESS ARCHITECTURE—CHROME BROWSER***

Many websites contain active content such as JavaScript, Flash, and HTML5 to provide a rich and dynamic web-browsing experience. Unfortunately, these web applications may also contain software bugs, which can result in sluggish response times and can even cause the web browser to crash. This isn't a big problem in a web browser that displays content from only one website. But most contemporary web browsers provide tabbed browsing, which allows a single instance of a web browser application to open several websites at the same time, with each site in a separate tab. To switch between the different sites , a user need only click on the appropriate tab. This arrangement is illustrated below:


A problem with this approach is that if a web application in any tab crashes, the entire process—including all other tabs displaying additional websites—crashes as well.

Google's Chrome web browser was designed to address this issue by using a multiprocess architecture. Chrome identifies three different types of processes: browser, renderers, and plug-ins.

- The **browser** process is responsible for managing the user interface as well as disk and network I/O. A new browser process is created when Chrome is started. Only one browser process is created.
- **Renderer** processes contain logic for rendering web pages. Thus, they contain the logic for handling HTML, Javascript, images, and so forth. As a general rule, a new renderer process is created for each website opened in a new tab, and so several renderer processes may be active at the same time.
- A **plug-in** process is created for each type of plug-in (such as Flash or QuickTime) in use. Plug-in processes contain the code for the plug-in as well as additional code that enables the plug-in to communicate with associated renderer processes and the browser process.

The advantage of the multiprocess approach is that websites run in isolation from one another. If one website crashes, only its renderer process is affected; all other processes remain unharmed. Furthermore, renderer processes run in a **sandbox**, which means that access to disk and network I/O is restricted, minimizing the effects of any security exploits.

Parallelism

Running multiple application simultaneously on multiple processors

Concurrency

Concurrency supports more than one task to make progress. Hence we can achieve the concurrency without parallelism

Multithreading model

To run the user level thread on the CPU-core the kernel level threads are important. This distributes the user level threads to the appropriate CPU core.

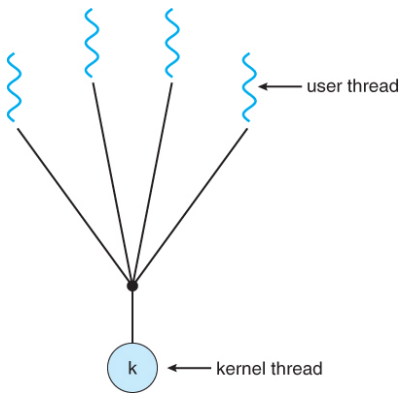
Execution of the user-level thread

When a user-level thread is scheduled for execution by the user-level threading library the corresponding kernel thread is dispatched by the OS thread scheduler onto a processor or CPU core.

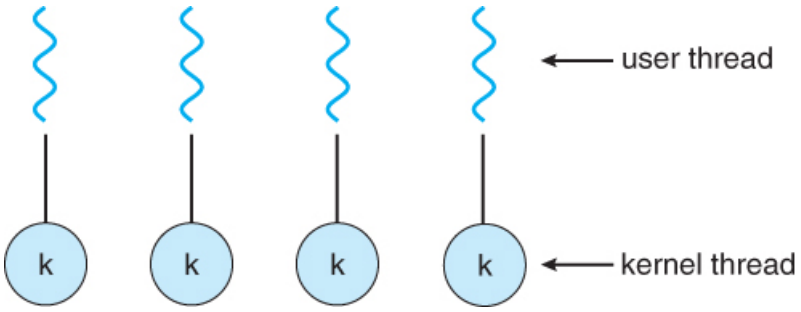
The kernel thread executes the instructions of the user-level thread until a blocking operation or context switch occurs.

Models of relationship between the user thread and kernel thread

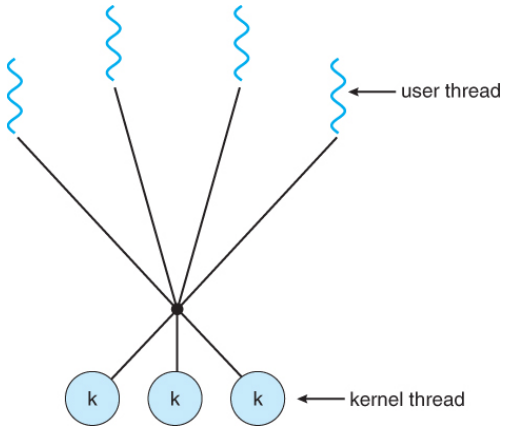
1) Many to one model : Many user threads to one kernel thread. Entire process will block if a thread makes the blocking system call. Only one thread can access the kernel thread at time makes multithreading unavailable



2) One-to-One model : Maps each user thread to the kernel thread. It provides more concurrency than a many-to-one model. The only drawback of the system is for each user thread the corresponding kernel thread is needed which creates the overhead to the system




3) Many-to-Many model : IT multiplexes many user threads to a smaller or equal number of kernel threads. It overcomes the shortcomings of the other model in which the user can create as many threads need and threads can run parallel on different cores of the CPU.




Thread Libraries

To manage the threads multiple operating system uses multiple different thread

1. Posix's Pthreads used by unix and linux
2. Windows threads used by windows

Java threads are also used but since the java runs on operating system, ultimately uses the OS's thread library

Pthreads and Windows use the global variables to share the between the threads but java is pure object oriented hence it uses the shared objects

There are two ways of creating the threads :

1. Asynchronous : Parent and child runs concurrently
2. Synchronous : Parent waits till child executes and then executes itself

Integer objects in java are immutable hence they cannot be changed.

Implicit threading

Automatic threading in which system identifies the part that can be parallelized and runs on different cores.

**Thread pool**

- ➔ The number of threads or concurrent execution in the system are limited and amount of time required for thread creation is less still its costly
- ➔ Hence the fixed number of threads required for the process are already created in the thread pool at the start of the process. Whenever the threads are necessary they are taken from the pool and after working the thread is disabled in the thread pool.
- ➔ Benefits of thread pool
  - ◆ Serving the existing thread is faster than creating the new thread
  - ◆ IT limits the number of threads created

OpenMP

It is a set of compiler directive written in C, C++ that provides the support of parallel programming in a shared memory environment. Divides the block in a program which can be run parallel.

Threading issues

The fork() and exec() system calls
fork() duplicates the process and exec() assigns the task to the process. If after duplicating immediately exec() is used then there is no use of creating the threads of the process because they may be changed.

Signal Handling
In multithreading the signal can be handled in two ways-

1) Asynchronous : Signal or interrupt is reported to any other thread such as ctrl + c
2) Synchronous : Signal reported to only process which caused the signal.

Thread Cancellation

Thread cancellation refers to the terminating the thread before it has completed its execution
(Pressing a button on the system which stops the searching in web page)

Types of cancellation

- Asynchronous : One thread immediately terminates the target thread
- Synchronous : The target thread periodically checks whether it should terminate.

- The difficulty with the cancellation is that a thread may not release all the system resources and it's difficult for the OS to do that.

- A cleanup handler relinquishes all the resources held by the thread if it found the cancellation request pending for the thread.

Thread Local Storage

Same as static storage but it is unique for each thread. Persistent across the function invocation.

Scheduler Activation

- A thread requires the LWP (Light weight process) data structure to interact with the kernel threads.
- In scheduler activation the kernel first allocates the LWP to the thread, if the thread is blocked it revokes the LWP and assigns it to any other process. IF the thread is again enabled the kernel assigns the new LWP to the thread or revokes existing one.

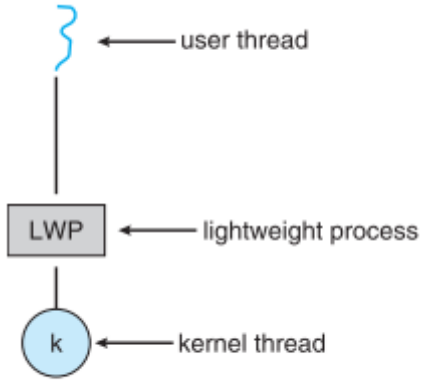



**Figure 4.13** Lightweight process (LWP).

### Interprocess communication

Independent process : Which does not gets affected by other processes' execution

Cooperating process : Does get affected

★ Cooperating processes require an interprocess communication (IPC ) mechanism that will allow them to exchange data and information. There are two fundamental models of interprocess communication: shared memory and message passing.

**What are the different ways processes can communicate with each other ?**


Shared-Memory Systems

- Shared memory is used
- Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment.
- Other processes that wish to communicate using this shared-memory segment must attach it to their address space.
- Shared memory is also called as buffer

Types of buffer

1. unbounded buffer - Unlimited size of buffer, consumer can wait, producer produces unlimited items
2. bounded buffer - Limited size of buffer, consumer can wait, producer assumes a fixed size buffer

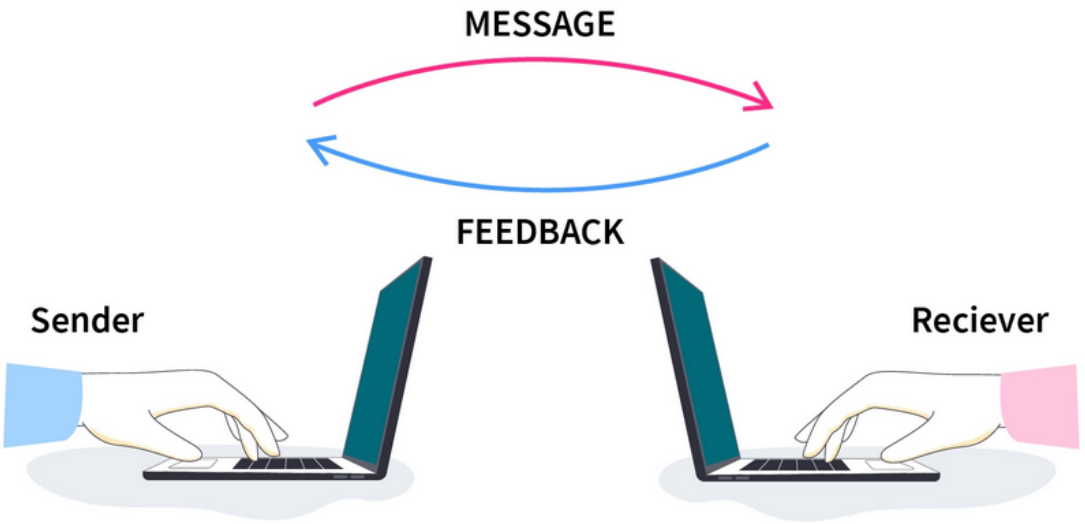
Message passing system

- Communication and synchronization is done by passing messages between processes without sharing a common address space
- It is mostly useful in distributed environments
- It has two actions, Sending (fixed size) message and receiving message.

Methods of Message Passing systems :

1. Naming [direct communication]
    - Processes that want to communicate must have a way to refer to each other.
    - They can use either direct or indirect communication. Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication.
    - In this scheme, the send() and receive() primitives are defined as:
        - send(P, message) —Send a message to process P .
        - receive(Q, message) —Receive a message from process Q

2. Mailbox [ Indirect communication ]
    - With indirect communication, the messages are sent to and received from mailboxes, or ports
    - A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed
    - POSIX [ Portable operating system interface for UNIX ] uses Message queue uses integer as id for mailbox




**How does IPC work in windows ?**

- The message-passing facility in Windows is called the advanced local procedure call (ALPC) facility.
- Similar to Standard RPC

Some of the commonly used IPC mechanisms in Windows include:

1. Named pipes [FIFO]
    - Anonymous pipes and names pipes both uses same mechanism but Anonymous pipes are volatile
    - Pipes allow two processes to communicate in standard producer–consumer fashion: the producer writes to one end of the pipe (the write-end) and the consumer reads from the other end (the read-end)
    - Named pipes allows bidirectional flow [ full duplex ], Anonymous used to require two pipes
    - Unlike shared memory, which involves processes accessing a common block of memory, piping relies on the operating system's process scheduling and memory management to handle the data transfer.

2. Windows Sockets [ Winsock ]
    - A socket is defined as an endpoint for communication.

- A pair of processes communicating over a network employs a pair of sockets ID one for each process.
- A socket is identified by an IP address concatenated with a port number. In general, sockets use a client–server architecture.
- Winsock is a programming interface that allows processes to communicate over network protocols, such as TCP/IP or UDP. It provides functions for creating sockets, establishing connections, and sending/receiving data between processes.

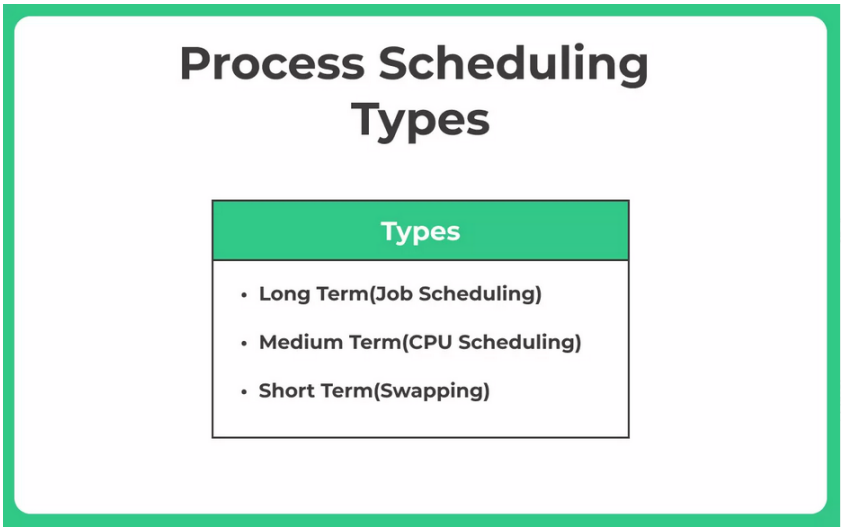
3. RPC
    - You know

**Where to use pipes [named or anonymous] and where to use Shared memory [FIFO]**

pipes for one-to-one communication, less coding and letting the OS handle things, shared memory for many-to-many, more manual control over things but at the cost of more work and harder debugging.

## Process Scheduling

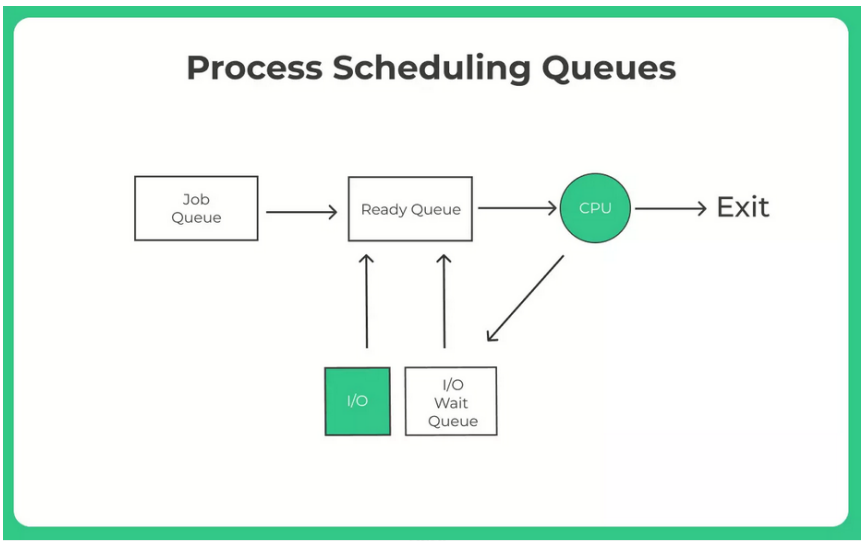
- Process scheduling is the mechanism that determines the order in which processes are executed by the operating system.
- The goal of process scheduling is to maximize the overall system performance, while ensuring that all processes get a fair share of the available resources




### 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.
- Non Preemptive Scheduling : The scheduling in which a running process cannot be interrupted by any other process is called non-preemptive scheduling.




Imagine a company that has a large number of tasks to complete, such as processing customer orders, generating invoices, and managing inventory. To manage these tasks efficiently, the company uses a job scheduler queue, which operates as follows:

1. Submission: Employees or users submit tasks to the job scheduler queue, either through a web interface, command-line interface, or batch files. These tasks may be submitted individually or in batches, depending on the workload.
2. Job Queue: The job scheduler queue then adds the submitted tasks to a job queue, which is a data structure that contains all pending tasks that are waiting to be executed.
3. Long-term Scheduler: The long-term scheduler (or job scheduler) then selects tasks from the job queue based on various factors, such as the available system resources, task priority, and other scheduling policies. The selected tasks are then moved from the job queue to the ready queue, where they wait for execution by the short-term scheduler.
4. Short-term Scheduler: The short-term scheduler (or CPU scheduler) then selects tasks from the ready queue and allocates CPU time to each task based on various scheduling policies, such as round-robin or priority-based scheduling. The selected task is then moved to the running state and starts executing on the CPU.
5. Execution and Completion: The selected task executes on the CPU and may use various system resources, such as memory, disk, and network.

Once the task is completed, it is moved to the terminated state, and its resources are released back to the 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.

- CPU Burst Time : In simple terms, the duration for which a process gets control of the CPU is the CPU burst time.
- CPU Utilization : CPU utilization can be defined as the the percentage of time CPU was handling process execution to total time

| 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 : Total amount of time spent in the ready queue to gain the access of the CPU for execution.

| Waiting Time = TurnAround Time - Burst Time |
|---|

- TurnAround Time : From the time the process is submitted to the time the process is completed, is defined as Turn Around Time.

| Turnaround Time = Burst Time + Waiting Time |
|---|

OR

| |
|---|
| Turnaround Time = Exit Time + Arrival Time |

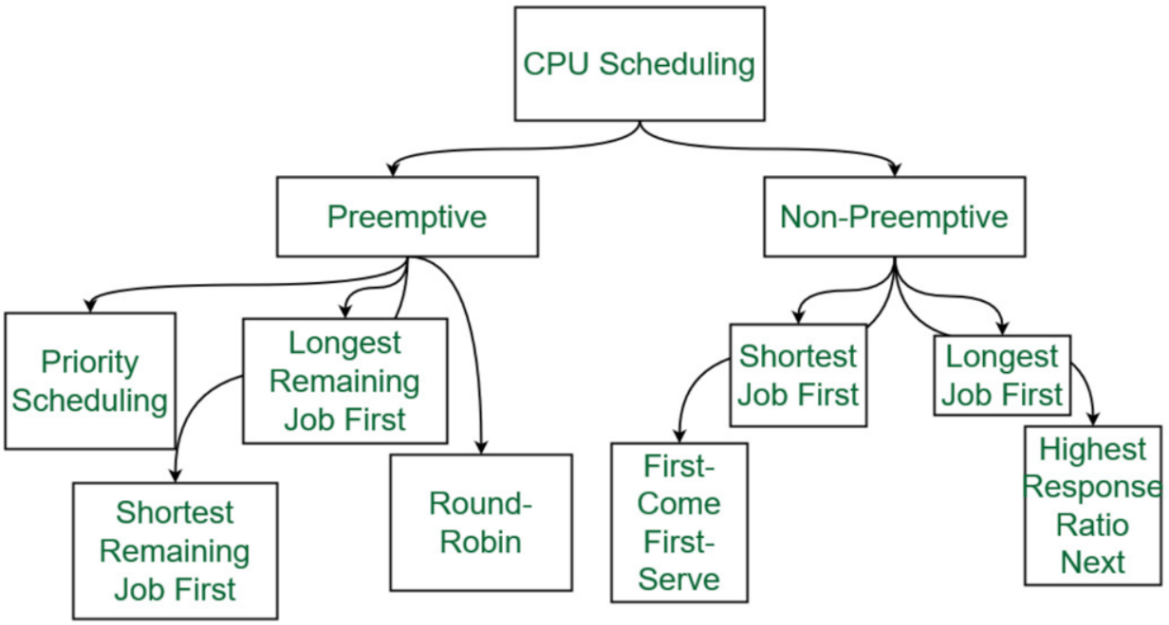
- Throughput : In a unit time the number of processes that can be completed is called the throughput.

| |
|---|
| Throughput = Number of processes completed / Total time |

- Load Time : It is defined as the average number of processes 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.

| |
|---|
| Response time = Time of first response - Arrival Time |

**Types of CPU Scheduling ?**



FCFS :

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.

### What is the convoy effect?

FCFS may suffer from the convoy effect if the burst time of the first job is the highest among all. As in 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.

SJF :

SJF (Shortest Job First) is a scheduling algorithm in which the scheduler selects the process with the smallest execution time to execute next.

One liner: SJF is a scheduling algorithm that selects the process with the shortest execution time.

Characteristics :

- Provides optimal average waiting time for a given set of processes.
- May cause starvation for long processes.
- Can be preemptive or non-preemptive.

SRTF :

SRTF (Shortest Remaining Time First) is a CPU scheduling algorithm where the process with the smallest amount of time remaining until completion is scheduled next.

One liner: SRTF schedules the process with the shortest remaining burst time first.

Characteristics :

- SRTF provides optimal average waiting time and turnaround time for processes.
- It can result in starvation for processes with long burst times.
- It requires preemption of currently executing processes.

RR :

RR is a scheduling algorithm that allocates CPU time to each process for a fixed time quantum, and then moves on to the next process in a circular fashion.

Characteristics :

- Provides fair and balanced CPU time allocation to processes.
- Uses a fixed time quantum for each process, typically in the range of 10-100 milliseconds.
- Prevents any single process from monopolizing the CPU.
- Easy to implement and suitable for time-sharing systems.

Advantages :

- Provides fair CPU time allocation to processes.
- Prevents any single process from monopolizing the CPU.
- Simple and easy to implement.

Disadvantages :

- Can result in lower CPU utilization than other scheduling algorithms, as processes are frequently switched in and out of the CPU.
- May not be suitable for real-time applications with strict timing requirements.

Effects of keeping the time quantum big and small :

- Keeping the time quantum big can lead to longer response times for interactive processes, as they may have to wait longer for their turn to use the CPU.

- Keeping the time quantum small can result in higher overhead due to frequent context switches, and can lead to lower CPU utilization.

Real-life use case :

- RR scheduling is commonly used in time-sharing systems, where multiple users share the same computing resources.

Priority scheduling :

Priority scheduling is a scheduling algorithm where the process with the highest priority is executed first.

Characteristics :

- Each process is assigned a priority, typically based on factors such as resource requirements, importance, and deadline.
- Processes with higher priority are executed before processes with lower priority.
- In case of ties, other scheduling algorithms such as first-come, first-served or round-robin can be used.

Advantages :

- Allows high-priority processes to be executed quickly, which can be important for time-critical applications.
- Can improve system responsiveness by prioritizing user-facing processes.
- Can be used to ensure that important system tasks are executed before less important ones.

Disadvantages :

- Can lead to starvation of low-priority processes if high-priority processes continue to arrive.

- May not be fair to low-priority processes, which may have to wait a long time to be executed.
- Requires careful management of priorities to prevent priority inversion, where a low-priority process holds a resource that a high-priority process needs to continue executing.

Multilevel scheduling :

Multilevel scheduling is a CPU scheduling technique that partitions the ready queue into separate queues with different priority levels, and applies different scheduling algorithms to each queue.

Characteristics :

- Multiple queues with different priorities
- Different scheduling algorithms for each queue
- Processes move between queues based on their priority and state

Advantages :

- Can provide better system responsiveness by giving higher priority to interactive processes
- Can prevent low-priority processes from monopolizing system resources
- Can allow for better resource utilization by optimizing CPU usage for different types of processes

Disadvantages :

- May cause starvation for low-priority processes if the higher-priority queues are always full
- Requires more complex implementation than a simple, single-queue scheduling algorithm
- May lead to unpredictable behavior if the priorities and scheduling algorithms are not carefully designed and configured.

Suppose we have three queues, namely Q1, Q2, and Q3, with Q1 being the highest priority queue and Q3 being the lowest priority queue. Each queue has a different scheduling algorithm, as follows:

- Q1: First-Come, First-Served (FCFS) scheduling algorithm
- Q2: Round-Robin (RR) scheduling algorithm with a time quantum of 10 milliseconds
- Q3: Priority scheduling algorithm, where processes with higher priority are executed first

Now let's assume that a system receives three processes, P1, P2, and P3, with the following attributes:

- P1: CPU-bound process with high priority
- P2: I/O-bound process with medium priority
- P3: CPU-bound process with low priority

When these processes arrive in the system, they are placed in their respective queues based on their priority and other attributes. So, P1 is placed in Q1, P2 is placed in Q2, and P3 is placed in Q3.

The scheduling algorithm now executes the processes in the queues according to their respective algorithms. In Q1, since we are using the FCFS algorithm, P1 is executed first. In Q2, since we are using the RR algorithm, P2 is executed for 10 milliseconds, and then the CPU is switched to the next process in the queue, which could be P1 again if it is still in the queue. In Q3, since we are using the priority algorithm, P3 is executed first.

As the processes are executed, they may change their attributes, such as their priority or CPU burst time, which can cause them to be moved to a different queue. For example, if P2 completes its I/O operation, it may be moved to Q1 if it now has a high priority due to its CPU burst time.

Multilevel feedback queue scheduling algorithm :

Multilevel Queue with option to switch between queues called as aging to avoid starvation problem

Characteristics :

- Multiple queues: Processes are placed in multiple queues based on their priority and time-slice requirements.
- Priority-based: Processes with higher priority are placed in higher-priority queues and are given shorter time-slices.

- Dynamic priority adjustment: If a process uses up its allotted time-slice, it is moved to a lower-priority queue, while processes that complete quickly are moved to higher-priority queues.
- Time-slice adjustment: The length of time-slices can be adjusted based on the current system load.

Advantages :

- Efficient use of resources: It ensures that CPU resources are efficiently allocated to processes based on their priority and time-slice requirements.
- Fairness: It provides fairness by ensuring that all processes are given an opportunity to execute, regardless of their priority.
- Response time: It reduces response time for interactive processes by giving them higher priority and shorter time-slices.

Disadvantages :

- Complexity: It is a complex algorithm that requires multiple queues and dynamic priority adjustment.
- Overhead: It can create additional overhead due to the need for multiple queues and frequent priority adjustments.

**How CPU Scheduling works in Linux systems ?**

a. Linux uses Completely Fair Scheduler (CFS)
    - This algorithm is used in modern Linux kernels and is designed to provide fairness and maximize overall system throughput.
    - The CFS scheduler uses a red-black tree data structure to maintain a dynamic scheduling queue.
    - The process with the smallest "virtual runtime" value is selected next for execution, where the virtual runtime represents the amount of CPU time that a process has used divided by its weight.
b. RR
c. PS
d. MLFQ

# Process Synchronization

Definition : Process synchronization refers to the coordination and orderly execution of multiple processes in a way that prevents conflicts and ensures they work together harmoniously to achieve their intended tasks.

### What is race condition ?

Race conditions occur when multiple processes or threads access shared resources in an unpredictable or undefined order, leading to inconsistent or incorrect results.

### What are Synchronization primitives ?

Synchronization primitives are tools used to ensure that multiple processes or threads do not access shared resources at the same time. Common synchronization primitives include mutexes, semaphores, and monitors.

### What is a critical section ?

Critical sections are sections of code in a process where shared resources are accessed. To ensure that only one process can access a shared resource at a time, critical sections must be protected with synchronization primitives

Two general approaches to handle the critical section:

- ➔ Preemptive Kernel
  - ◆ It allows a process to be preempted while it is running in kernel mode
  - ◆ More responsive
  - ◆ More suitable for real-time programming
- ➔ Non preemptive Kernel
  - ◆ It does not allow a process running in kernel mode to be preempted

- ◆ A kernel mode process will run until it exists kernel mode blocks.
- ◆ It is free from the race condition
- ◆ Only one process is active at a kernel at time

**Why is Process Synchronization 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 situations 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 lets you enter ATM password and withdraw the 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 the balance was just 5000$.

*How to solve this Situation ?*

To avoid such situations process synchronization 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 conditions. 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.

### What are the types of Processes?

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

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

### Do independent processes need to have synchronization ?

- Independent processes generally do not need to be synchronized with each other because they do not share any resources or data that could be affected by concurrent access.
- But there is a rare case where independent processes needs synchronization, like in the situations where we have to audit the changes made by individual processes

### What is a producer consumer problem ? Give a real life example of it ?

- Also known as the bounded-buffer problem
- Two types of processes, producer and consumer which operates on same data
- Both work concurrently
- Producer generates data items and stores in common buffer in memory

- Consumer takes that data and process it
- Eg - When we give command to printer, if the prints are less frequent the consumer will sit idle, or it can exceed the memory if produced in more number
- This is called as printer spooler problem

**What is a Reader Writer problem ? Give a real life example of it ?**

➔ It involves multiple threads accessing a shared resource that can be read or written to.

➔ It can be solved using various synchronization techniques, including semaphores, mutexes

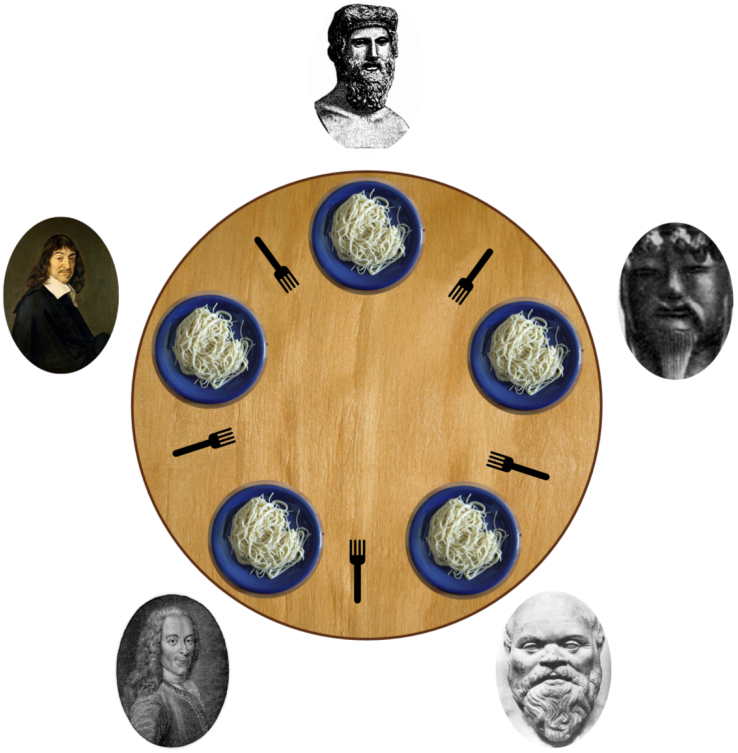
➔ Real-life example in Computer Science:

- Consider a database system that has multiple users accessing the same database simultaneously.
- Readers can access the database to read information, while writers can modify the database by adding or updating records.
- If multiple readers try to access the database simultaneously, they can do so without interfering with each other.
- However, if a writer is modifying the database, no other reader or writer should be allowed to access the database until the write operation is complete.

**What is a Dining Philosopher problem ? Give a real life example of it ?**

Dining philosophers problem Animation

- The dining philosophers problem is a classic synchronization problem in computer science that illustrates the challenge of avoiding deadlock and resource starvation in a multi-threaded environment
- Classic Deadlock



**What is a Bounded Buffer problem ?**

- A variation of the producer-consumer problem
- There is a shared fixed-size buffer that can hold a limited number of items
- There are multiple threads, including producers and consumers, that need to access the shared buffer
- Producers may block when the buffer is full, and consumers may block when the buffer is empty
- There is a need for synchronization to ensure that the threads do not access the buffer at the same time

**To achieve synchronization three conditions should be satisfied**

1. Mutual Exclusion

   Only one process at a time is allowed to operate inside critical section. All cooperating processes should follow this.

2. Progress

   If no process is executing in the critical section and there are no other processes waiting to enter the critical section, then any process that wants to enter the critical section should be able to do so immediately.

3. Bounded Waiting

   A bound must exist on the number of times that other processes are allowed to enter the critical section after a process has made a request to enter the critical section and before that request is granted.

### What is the lock variable ?

1. The lock variable is initially set to "unlocked."
2. Thread A wants to access the shared resource, so it sets the lock variable to "locked" to indicate that it is using the resource.
3. If Thread B tries to access the shared resource while Thread A has the lock, it will see that the lock variable is "locked" and will have to wait until Thread A releases the lock.
4. Once Thread A has finished using the shared resource, it sets the lock variable back to "unlocked" so that other threads can access it.

   Process1:

```
while(true){
    while(lock !=0);
    lock=1;
    // Critical section
    lock=0;
    //Remaining section
}
```

Process 2:

```
while(true){
    while(lock !=0);
    lock=1;
    // Critical section
    lock=0;
    //Remaining section
}
```

**What is Test and Set method ?**

1. The process that wants to access the shared resource first reads the value at the memory location.
2. If the value is false, the process sets the value to true and enters the critical section (i.e., accesses the shared resource). If the value is true, the process spins in a loop, repeatedly trying to read the value until it becomes false.
3. When the process is finished using the shared resource, it sets the value back to false to indicate that it has finished.
4. Any other process that wants to access the shared resource must first perform the Test and Set operation. If the value is true, it must spin in a loop until it becomes false.
5. Code:

```
boolean TestAndSet(boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
do
{
    while (TestAndSet(&lock))
        ; // do nothing
    // critical section
    lock = FALSE;
    // remainder
} while (TRUE);
```

**Then what is the difference between both ?**

- Implementation : Lock variables are implemented using conditional statements and mutexes, while Test and Set is implemented using an atomic instruction
- Use cases : Lock variables are often used in complex synchronization scenarios, while Test and Set is used for simple synchronization problems such as spin locks.

## Semaphores

- Semaphore is an entity in OS that is used to solve critical section problem
- A variable that controls access to a resource, such as a file, a piece of memory, or a hardware device.

Two main operations :

1. wait() : The wait() operation, also known as the P() operation, decrements the value of the semaphore by 1. If the result is negative, it indicates that the resource is currently unavailable and the process must wait for it to become available.
2. signal() : The signal() operation, also known as the V() operation, increments the value of the semaphore by 1. If any processes are waiting on the semaphore, it wakes up one of them to access the shared resource.

## Mutexes

➔ Mutex is a programming object that allows only one thread to access a shared resource at a time, thereby ensuring mutual exclusion.
➔ It provides synchronization between threads and is commonly used in multithreaded programming to protect shared resources from concurrent access.
➔ A mutex can be either locked or unlocked, and only one thread can hold the lock at a time.

➔ Mutexes are typically used to protect data structures such as critical sections, shared variables, and other shared resources.
➔ Mutexes can be implemented in different ways, such as spinlocks, sleep locks, or hybrid locks.

Implementing Mutexes :

1. Mutexes can be implemented using several mechanisms such as hardware instructions, atomic operations, or software locks.
2. The most commonly used software-based implementation is Peterson's algorithm, Dekker's algorithm, and the Bakery algorithm.
3. Operating systems provide built-in support for mutexes that can be used by applications to synchronize threads.

**Difference between Mutex and Semaphore ?**

| Criteria | Mutex | Semaphore |
|---|---|---|
| Definition | A mutex is a binary semaphore that provides mutual exclusion, allowing only one thread to enter a critical section at a time. | A semaphore is a signaling mechanism that allows multiple threads to access a shared resource concurrently, subject to a limit set by the semaphore count. |
| Type | Mutex is a binary semaphore. | Semaphore can be binary or counting. |
| Count | Mutex count is always 1. | Semaphore count can be more than 1. |
| Ownership | Mutex can be owned by a single thread. | Semaphore does not have ownership. |
| Purpose | Mutex is used for protecting a critical section of code, allowing only one thread to access | Semaphore is used for synchronization between multiple threads, limiting the number of threads accessing a |

| | | |
|---|---|---|
| | the section at a time. | shared resource. |
| Implementation | Mutexes are usually implemented using atomic hardware instructions. | Semaphores are usually implemented using software operations. |

### What are 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.

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.

[for codependent processes] If 1 Process is blocked , no other processes can execute unless the blocked process is unblocked true or false

It depends on the type of blocking. If the process is blocked on a shared resource using synchronization mechanisms such as semaphores or mutexes, then other processes can continue to execute. However, if the process is blocked due to an I/O operation or waiting for a signal, then other processes may also be blocked until the operation completes or the signal is received. So, the statement is false in general.

### Spin Loop

In the spin loop, the thread/process repeatedly checks the lock status to see if it has been released. This constant checking is called "spinning." Instead of yielding the CPU or putting the thread/process to sleep, it actively waits for the lock to be released. This spinning loop can consume CPU resources, so spinlocks are most effective when the waiting time is expected to be very short.

Example : Let's consider a scenario where two threads, Thread A and Thread B, need to access a shared resource protected by a spinlock.

- ➔ Initially, the spinlock is in an unlocked state.
- ➔ Thread A arrives first and attempts to acquire the spinlock. Since it's unlocked, Thread A takes ownership of the lock and enters the critical section.
- ➔ While Thread A is executing in the critical section, Thread B also wants to access the resource and attempts to acquire the spinlock. However, the lock is currently locked by Thread A, so Thread B enters a spin loop, repeatedly checking the lock status.
- ➔ Thread A finishes its work in the critical section and releases the spinlock, marking it as unlocked.
- ➔ Thread B, which has been spinning, detects that the lock is now unlocked and immediately acquires the spinlock, taking ownership and entering the critical section.
- ➔ Both Thread A and Thread B access the shared resource sequentially, ensuring that they don't interfere with each other.

**Advantages:**

- ➔ Less overhead when waiting time is short
- ➔ No context switching involved.
- ➔ Suitable when waiting time is less

**Disadvantages:**

- ➔ less suitable when waiting times are longer as they can waste CPU resources.

**What if more than one process is there in spinLoop , which process will get the access to the lock when the process holding the lock releases?**

- ➔ The order in which processes acquire the lock may vary each time, depending on the scheduling algorithm and the dynamic conditions of the system.
- ➔ The objective is to provide fairness and ensure that all waiting processes eventually get an opportunity to access the critical section, preventing any particular process from monopolizing the lock indefinitely.

## Monitors

- ➔ Monitors in operating systems are a synchronization construct used to manage access to shared resources or critical sections in a concurrent programming environment.
- ➔ They provide a higher-level abstraction for synchronization and ensure that only one thread or process can execute the critical section at a time.

## Purpose

The primary purpose of monitors is to provide mutual exclusion and synchronization. They ensure that only one thread or process can enter a critical section within a monitor at a time, allowing for safe and coordinated access to shared resources.

## Encapsulation

Monitors encapsulate the shared data and the procedures (also known as methods or functions) that operate on that data. The shared data and the procedures are combined into a single construct called a monitor. The monitor defines the boundaries of the critical section, ensuring that only one thread or process can be inside the monitor executing its procedures at any given time.

### Entry and Exit

- To access the critical section within a monitor, a thread or process must acquire the monitor's lock. This lock ensures exclusive access to the monitor's procedures.
- When a thread or process enters the monitor, it gains ownership of the lock and can execute the procedures within the critical section.
- Once it completes its work, it releases the lock, allowing other threads or processes to acquire it and enter the monitor.

### Condition Variables

- Monitors often provide condition variables, which are mechanisms for threads or processes to wait for specific conditions to be met before proceeding.
- Condition variables are associated with the monitor and allow threads or processes to suspend their execution, freeing the monitor's lock, until a certain condition becomes true.
- For example, a producer-consumer problem can be solved using a monitor with condition variables, where the consumer waits until the buffer is not empty, and the producer signals the consumer when it adds an item to the buffer.

### Example of monitor

Let's consider an example where multiple threads need to access a shared counter variable in a program. We can use a monitor to ensure that only one thread can access the counter at a time to prevent inconsistencies.

- The monitor encapsulates the counter variable and the procedures that operate on it.
- Threads wishing to access the counter first acquire the monitor's lock.
- Once a thread acquires the lock, it can increment or read the counter safely within the critical section of the monitor.
- Other threads that try to access the counter will be blocked until the lock is released.
- When a thread finishes its work on the counter, it releases the lock, allowing other waiting threads to acquire it and access the counter.

Advantages

- Monitors provide a higher-level abstraction for synchronization,
- They encapsulate the shared data and synchronization mechanisms within a single construct,
- Promot's modular and structured programming.
- Help to prevent race conditions and ensure the integrity and consistency of shared data.

[ Syntax ]

```
monitor monitor name
{
        /* shared variable declarations */
        function P1 ( . . . ) {
        . . .
        }
        function P2 ( . . . ) {
        . . .
        }
        .
        .
        .
        function Pn ( . . . ) {
        . . .
        }
        }
        initialization code ( . . . ) {
        . . .
}
```

**Why do we use the monitors when we already have the semaphores?**

Semaphores provide a more general-purpose and flexible approach to synchronization, monitors offer a higher-level abstraction that simplifies the management of shared resources.

## Priority Inversion

Priority inversion is a phenomenon that can occur in a multitasking or multi-threaded system when a higher-priority task or thread is blocked or delayed by a lower-priority task.

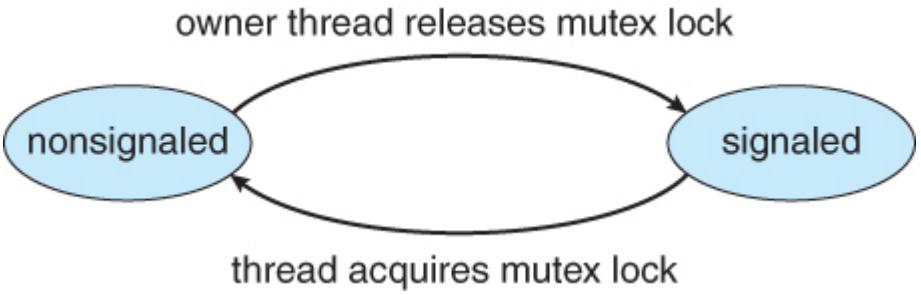
Priority inversion occurs when a higher-priority task is indirectly delayed or blocked by a lower-priority task due to resource dependency.

- Suppose there are three tasks: Task A (high priority), Task B (medium priority), and Task C (low priority).
- Task A is currently executing and needs to access a shared resource that is currently held by Task B.
- However, Task B gets preempted by Task C, which has a higher priority than Task B.
- Task A cannot proceed until Task B releases the shared resource, but it is unable to do so because it is preempted by Task C.
- As a result, Task A, which has a higher priority than both Task B and Task C, remains blocked or delayed, causing priority inversion.

## Synchronization in linux

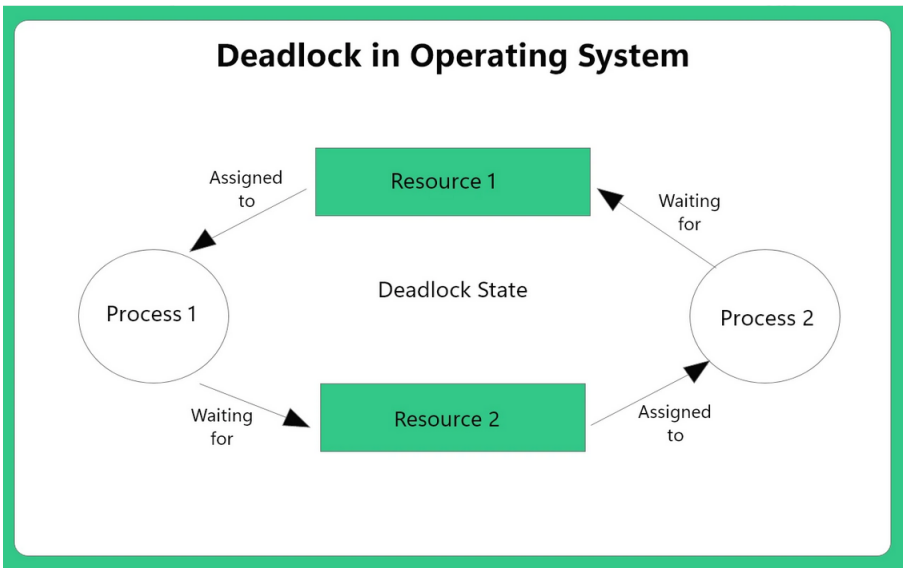
| single processor | multiple processors |
|---|---|
| Disable kernel preemption. | Acquire spin lock. |
| Enable kernel preemption. | Release spin lock. |

## Synchronization in windows



## 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




### What are the four necessary conditions for Deadlock to occur ?

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.

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

### What are the methods to prevent Deadlock from occurring ?

1. Deadlock prevention : In this method, one or more of the necessary conditions for Deadlock are prevented. For example, by ensuring that

resources are requested and released in a specific order, it is possible to prevent the "hold and wait" condition.

2. Resource allocation graph : In this method, a directed graph is used to represent the resources and processes involved in a system. If a cycle is detected in the graph, it indicates the presence of a Deadlock.
3. Banker's algorithm : This is a resource allocation algorithm that is used to avoid Deadlock. It involves predicting the maximum amount of resources a process may need and then only allocating them if they are available.

### What is deadlock prevention and avoidance ? explain the difference

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.

Methods of deadlock prevention :

1. Resource allocation
2. Timeout
3. Priority based resource allocation
4. 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.
5. 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.

Methods of deadlock avoidance

1. Banker’s Algorithm [IMP]

2. 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.

### What is livelock ? How is it different from a deadlock ?

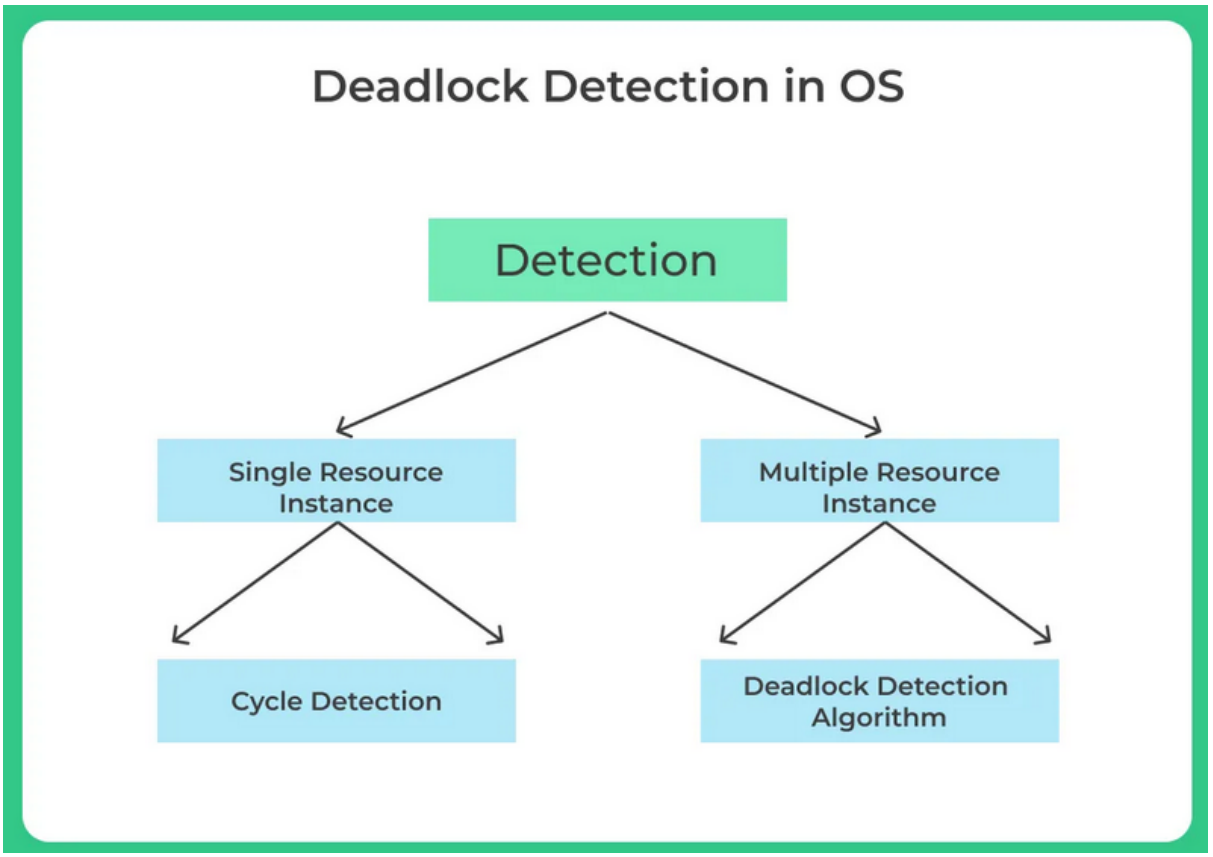
Livelock is a scenario in which processes are continuously changing their states in response to other processes, without making progress towards completing their task.

```
import { useEffect, useState } from "react";

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(`Count: ${count}`);
    setCount(count + 1);
  }, [count]);

  return null;
}
will this code make reactjs toenter in livelock situation
```

Yes, this code has the potential to cause an infinite loop and lead to a livelock situation.

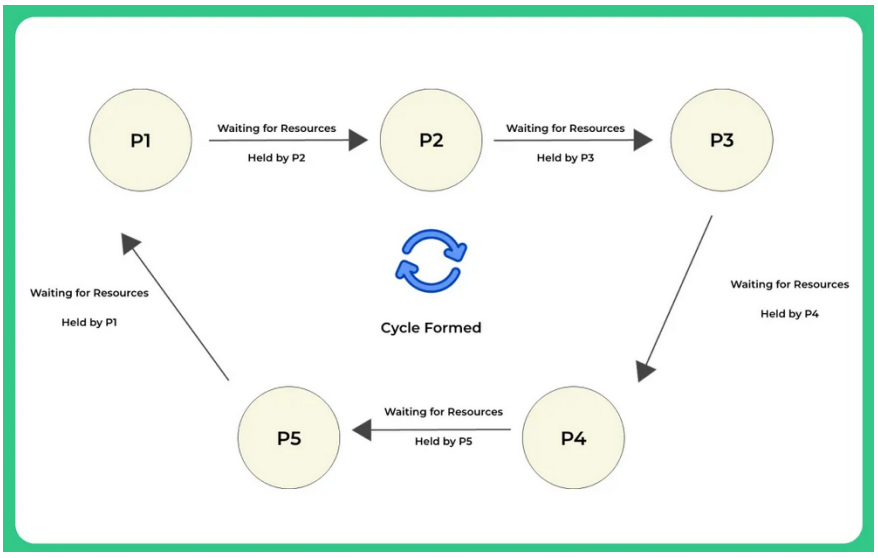
The useEffect hook is triggered on every re-render of the component due to the dependency array [count], which includes the count state variable. Inside the useEffect callback, the setCount function is called, which updates the count state variable, triggering a re-render and executing the useEffect callback again

**What is Deadlock detection ?**




If there is a single instance of the Operating System

- Wait for Graph : Draw RAG and boom



### What is deadlock recovery ? [same as avoidance]

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.

### Can you describe a situation where deadlock may occur even in the presence of a deadlock prevention algorithm?

Suppose there are two resources A and B, and three processes P1, P2, and P3. The system has one instance of each resource. The processes have the following sequence of requests:

- P1 requests A and then B
- P2 requests B and then A
- P3 requests A and then B

If we use the Banker's algorithm for deadlock prevention, then the system will be in a safe state if we start with any of the following sequences:

- P1, P2, P3
- P3, P2, P1
- P3, P1, P2

However, if we start with P1 and P2 executing simultaneously and P3 waiting, the system will enter a deadlock state because P1 holds A and is waiting for B, while P2 holds B and is waiting for A. The deadlock prevention algorithm did not anticipate this particular sequence of events and hence failed to prevent the deadlock.

## Memory management

- CPU is not directly connected to Secondary storage, reason : Slow
- Memory management is the process of 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.

**When Operating System receives an API request, how does it handle it with the job queue, secondary storage ready queue ?**

➔ When you receive an API request, the OS does not directly handle it. The request first goes to the application that is responsible for handling the request. However, when the application requires additional resources such as memory, the OS comes into play.

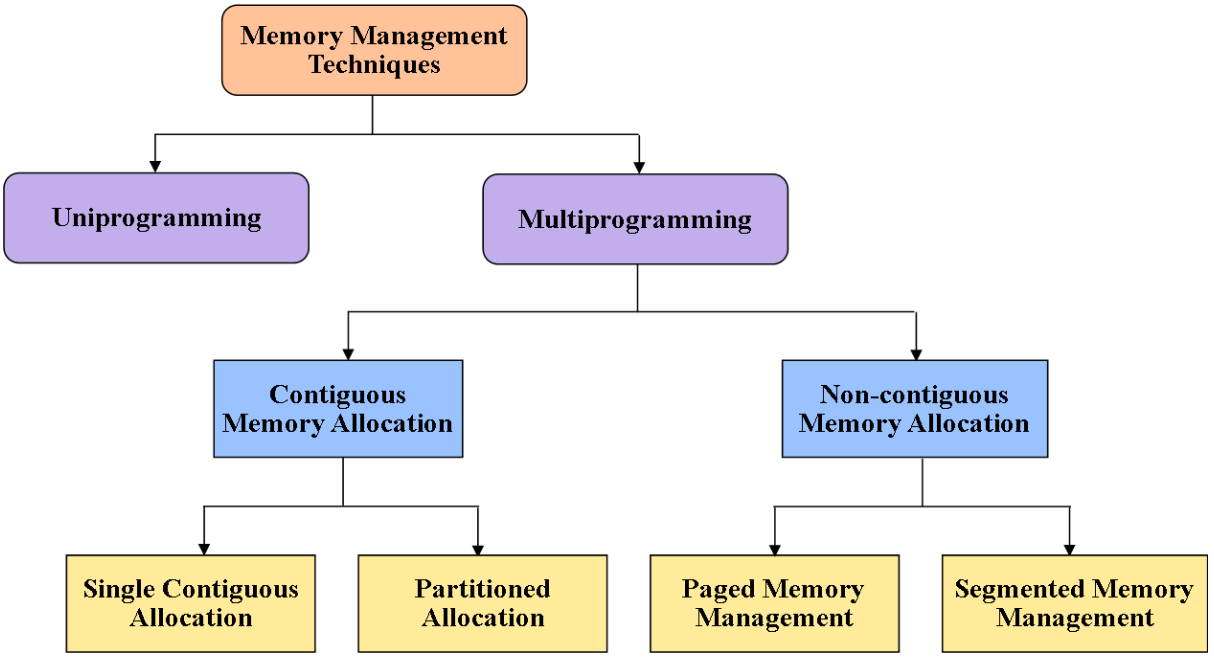
➔ The application may request memory for its data and code from the OS. The OS then checks if the requested memory is available in the main memory. If it is not available, the OS creates a new process for the application and assigns it to the job queue.

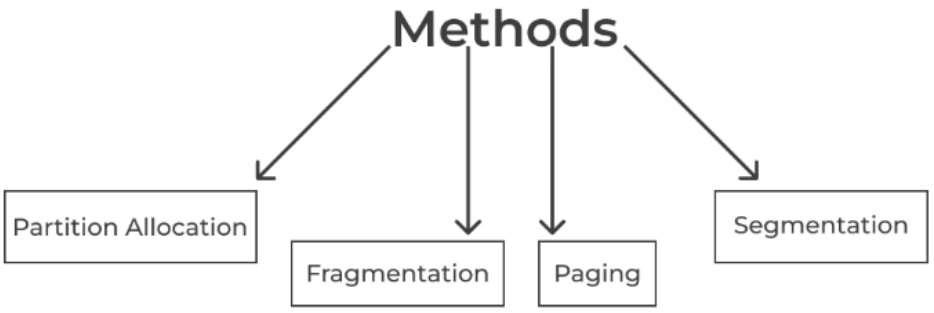
➔ The job queue contains a list of processes that are waiting for memory allocation. When the required memory is available, the process is moved to the ready queue, which is a list of processes that are waiting to be executed by the CPU.

➔ If the CPU is available, the OS assigns the process from the ready queue to the CPU, and the process starts executing. As the process executes, it may request more memory from the OS, which repeats the same process of checking memory availability and allocating it to the process.

➔ When the process is done with its task, it is removed from the memory, and the allocated memory is released back to the OS for further allocation to other processes

**Memory Management Techniques**




Contiguous memory allocation :

Contiguous memory allocation is a memory management technique in which a process is allocated a contiguous block of physical memory

Types of Contiguous Memory Allocation :

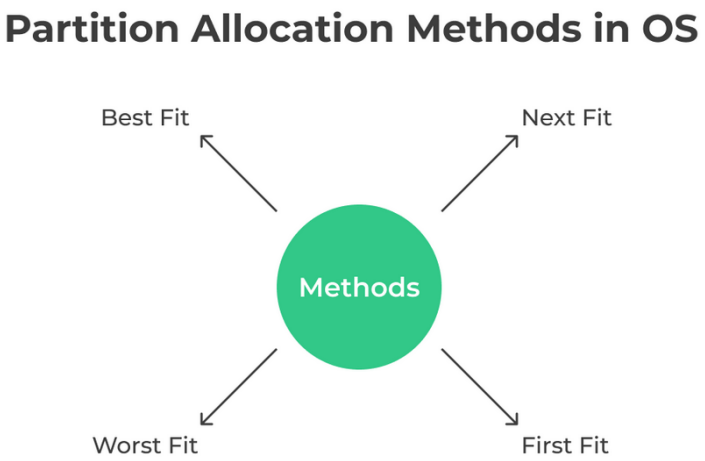
A. Single Partition Allocation
B. Multiple Partition Allocation

Single Partition Allocation:

- In this type of allocation, the operating system reserves the entire memory for a single process.
- suffers from internal fragmentation
- MS-DOS.

Multiple Partition Allocation:

- In this type of allocation, the memory is divided into fixed-size partitions, and each process is allocated one partition.
- Types :
    - Fixed partitioning
    - Variable partitioning



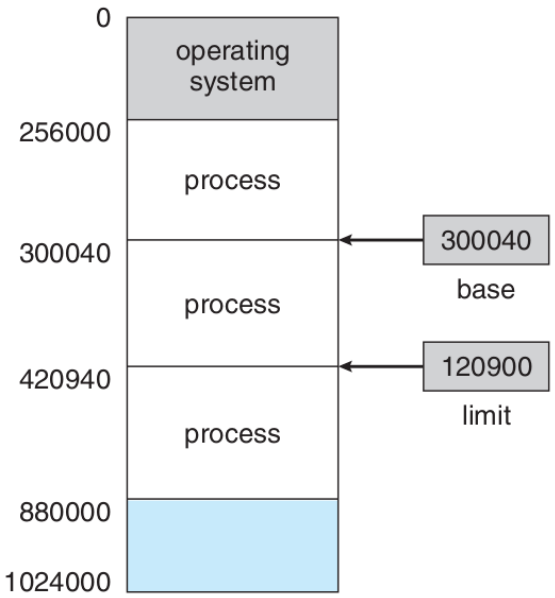



Figure 8.1 A base and a limit register define a logical address space.

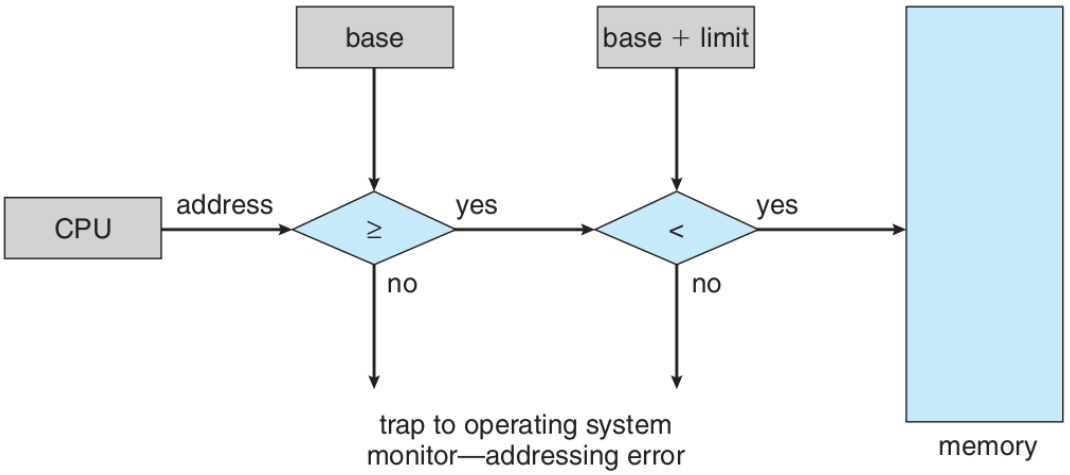



**Figure 8.2** Hardware address protection with base and limit registers.

The base and limit registers can be loaded only by the operating system, which uses a special privileged instruction. Since privileged instructions can be executed only in kernel mode, and since only the operating system executes in kernel mode, only the operating system can load the base and limit registers.

**Address Binding**

Address binding is the process of associating a specific address with a particular entity, such as a variable, function, or memory location, in a computer system. It involves mapping logical or symbolic addresses to physical addresses or locations in memory.

**1. Compile Time Binding:**

1. In compile time binding, the addresses of variables, functions, and memory locations are determined at compile time and are fixed throughout the execution of the program.
2. Example : Consider a C program where a variable `x` is declared as `int x = 5;`. During compilation, the compiler assigns a fixed memory location for the variable `x`. The address of `x` is determined during the compilation process and remains the same during program execution.

**2. Load Time Binding**

1. In load time binding, the addresses are determined and assigned to entities during the program loading phase, which happens before the execution starts. The addresses may depend on the current memory layout and other factors at the time of loading.
2. Example : In a compiled program, when the program is loaded into memory, the addresses of functions and global variables are determined and assigned by the loader. These addresses are then used by the program during execution.

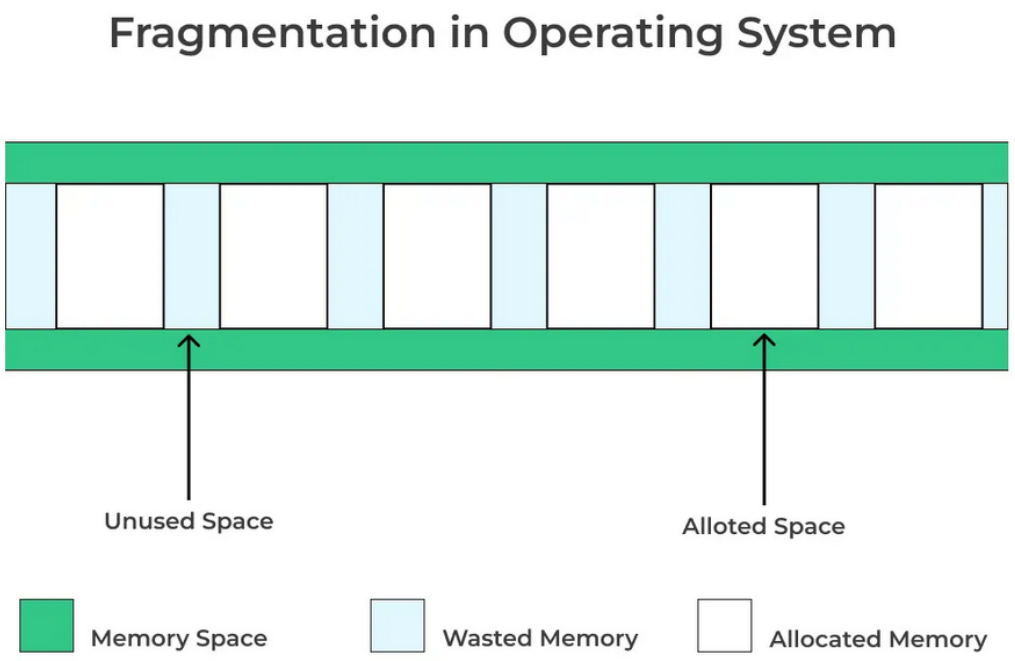
**3. Run Time Binding**

1. Run time binding allows the addresses to be determined dynamically during program execution. The mapping of addresses is not fixed before the program starts but happens during runtime based on various factors.
2. Example : In a program that uses dynamic memory allocation, such as C's `malloc()` function, the addresses assigned to dynamically allocated memory blocks are determined during runtime. The

program requests memory from the operating system at runtime, and the operating system assigns the memory block's address.

**What is Fragmentation in Operating System ?**

- 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 utilized fragmented memory spaces, which are so small that normal processes can not fit into those small fragments, causing those memory spaces not getting utilized at all
- Fragmentation is of the following two types :
    1. Internal Fragmentation
    2. External Fragmentation



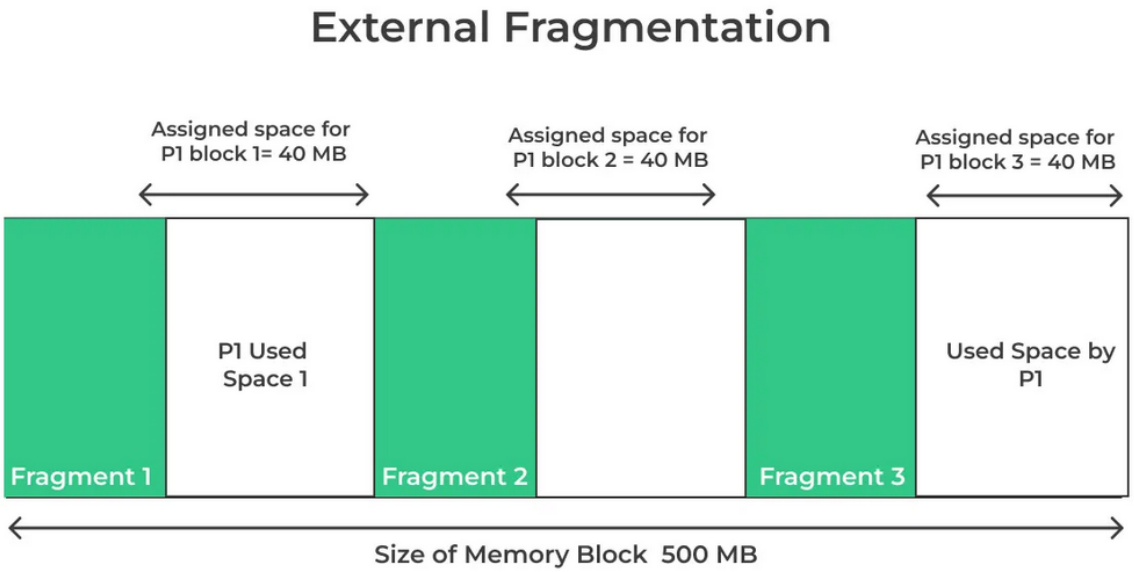

Internal fragmentation

- Occurs when a process is allocated more memory than it actually needs

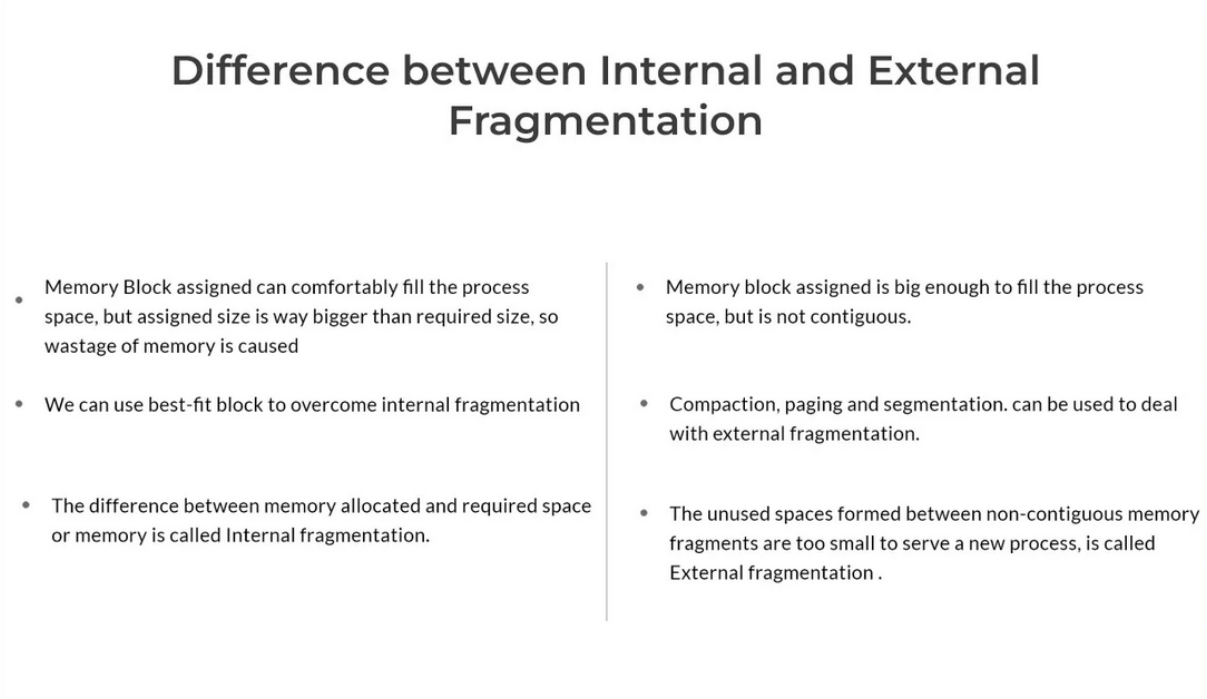
- For example, consider a process that requests 100 KB of memory, but the only available block of memory is 120 KB. The process is allocated the entire block, but 20 KB of memory is wasted as internal fragmentation

External fragmentation

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




Required space for P1 = 100 MB
(P1 memory is scattered in 3 non - contagious blocks 40 + 40 + 40 = 120)




Fixed partitioning :

- In fixed size partitioning, the memory is divided into fixed size partitions, and each partition can be allocated to a process.
- The size of the partition is determined at system initialization and cannot be changed dynamically.
- Each process is allocated a single partition, and no other process can use that partition.
- Examples include IBM 360/370 OS and the early versions of Microsoft Windows.
- Advantages:
    - Simple and easy to implement.
    - No external fragmentation.
- Disadvantages:
    - Internal fragmentation.
    - Inefficient memory usage.
    - Limited number of processes can be executed.

Variable Size Partitioning :

- In variable size partitioning, the memory is divided into partitions of varying sizes, and each partition can be allocated to a process.
- The size of the partition is determined at runtime and can be changed dynamically.
- When a process requests memory, the system searches for an available partition that can accommodate the requested size.
- Examples include UNIX, LINUX, and modern versions of Windows.
- Allocation methods:

1. First Fit:
    - It allocates the first available partition in the memory that is large enough to accommodate the process.
    - Advantages: It is easy to implement and has a relatively low overhead.
    - Disadvantages: It may result in inefficient use of memory due to fragmentation.
    - Example: Suppose the memory is divided into partitions of size 50KB, 100KB, 200KB, and 300KB. If a process requests 150KB of memory, it will be allocated the 200KB partition.

2. Best Fit :
    - It allocates the smallest partition in the memory that is large enough to accommodate the process.
    - Advantages: It reduces fragmentation and makes efficient use of memory.
    - Disadvantages: It requires more time to search for the best-fit partition, leading to higher overhead.
    - Example: Suppose the memory is divided into partitions of size 50KB, 100KB, 200KB, and 300KB. If a process requests 150KB of memory, it will be allocated the 200KB partition.

3. Worst Fit :
    - It allocates the largest partition in the memory that is large enough to accommodate the process.
    - Advantages: It reduces the probability of larger processes being left out of the memory.
    - Disadvantages: It leads to higher fragmentation of memory.
    - Example: Suppose the memory is divided into partitions of size 50KB, 100KB, 200KB, and 300KB. If a process requests 150KB of memory, it will be allocated the 300KB partition.

- Advantages:
    - Efficient memory usage.
    - Can accommodate processes of varying sizes.
- Disadvantages:
    - External fragmentation.
    - Overhead in searching for available partitions.

Non Contiguous Memory Allocation :

- Memory is not allocated in contiguous blocks
- Each process can be allocated a portion of memory from any part of the physical memory, without the need for the memory to be contiguous

Two types of NCMA :

a. Paging
b. Segmentation

Paging

- Eliminates the need for contiguous allocation of physical memory
- We can have logically use memory spaces that physically lie at different locations in the memory
- This is possible via mapping of physical and virtual memory that is done by a hardware component called memory management unit (MMU). The mapping process that is done by MMU is basically a paging process.

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

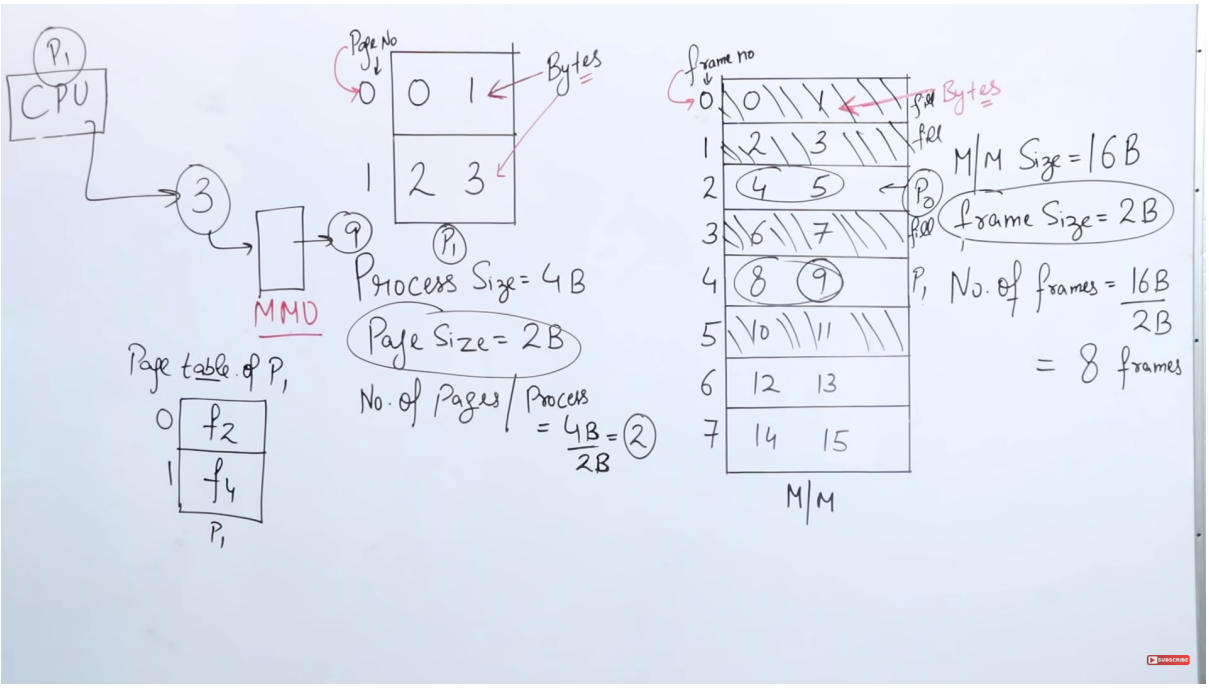
Physical View

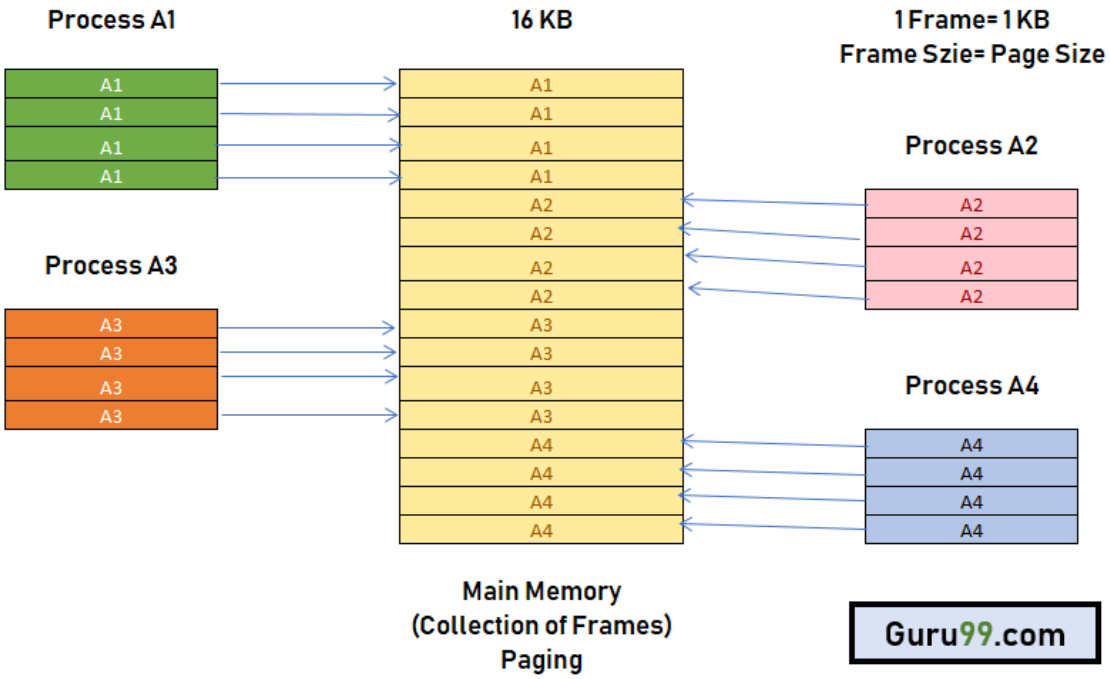
- Physical Space Available – All possible set of physical store locations
- Physical Address – This is the true address as seen where information is stored in the hardware.
- Conversion Examples –
    - If Logical address is 34 bit then -> Logical Address space = $2^{34}$ which is $2^4$ * $2^{30}$. Thus 16 G words

Frames :

➔ 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
➔ FrameSize === PageSize
➔ Frames == Main Memory divided into parts,
➔ Pages == Process divided in parts

*Tip : byte addressability refers to the ability of a CPU or memory system to individually address and access bytes of data. This means that the CPU can access any byte of memory directly by specifying its address. In a byte-addressable system, each byte is assigned a unique memory address, and the CPU can access or manipulate individual bytes of data in memory using their respective addresses*



*In simple terms,*

A bit can be 0 || 1

If the page size is :

2Bit = it can store upto 2^2 = 4-1 = 3

3Bit = it can store upto 2^3 = 8-1 = 7

Lets suppose the CPU demands 0x7FFF1234 page

If the page size is 4kb = 4096 Bytes

How many bits are required to represent 4096 Bytes ?

Log2(4096) = 12;

That means To calculate the page number, we need to remove the 12 least significant bits, which represent the offset. Therefore, the page number is 0x7FFF1

Why first 5 chars after X ?

Because total 8 chars and 32 bit address so

1 char = 4 bits

5 chars = 20 bits == page number

Remaining is offset

Memory Management Unit (MMU)

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)

Full Example

1. Process Request :

   The process generates a logical/virtual address, let's say 0x12345678, to read data from.

2. Address Translation:

   The CPU receives the logical/virtual address and sends it to the MMU for address translation. The MMU splits the logical address into two parts: the page number and the offset.

   - Page Number: The MMU extracts the page number from the logical address, in this case, it's 0x12345.
   - Offset: The MMU retrieves the offset within the page, which is 0x678.

3. Page Table Lookup:

   The MMU consults the page table, which is maintained by the operating system, to find the corresponding physical frame for the given page number.

   - Page Table Lookup: The MMU looks up the page number, 0x12345, in the page table.
   - Physical Frame: The page table provides the MMU with the corresponding physical frame number, let's say it's 0x9876.

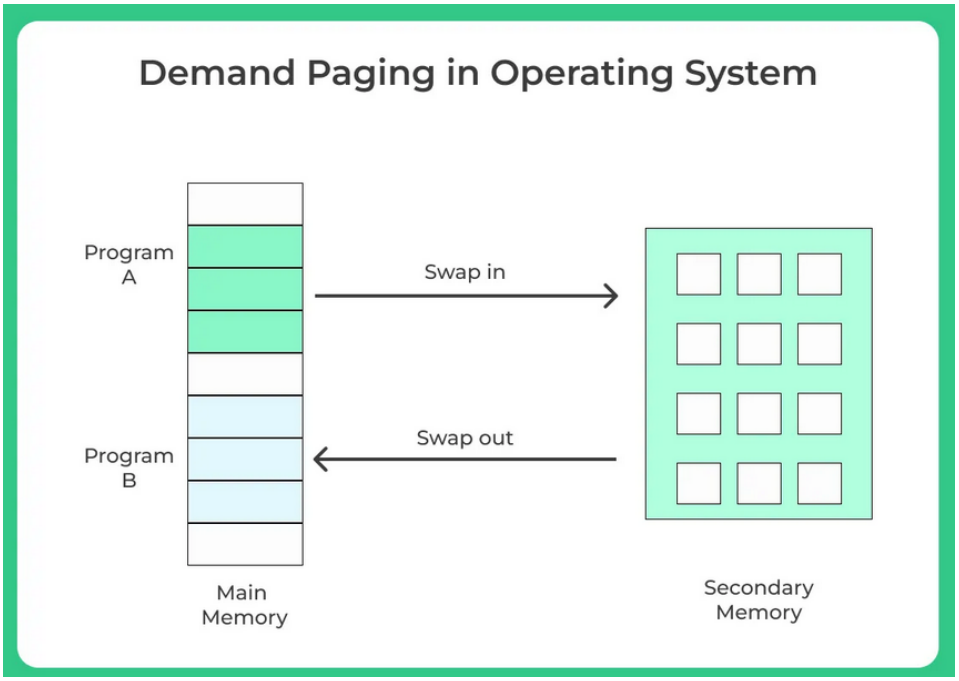
4. Physical Address Generation :

   With the physical frame number and the offset, the MMU generates the physical address.

   - Physical Address: The MMU combines the physical frame number, 0x9876, with the offset, 0x678, to form the physical address, which is 0x9876678.

5. Memory Access:

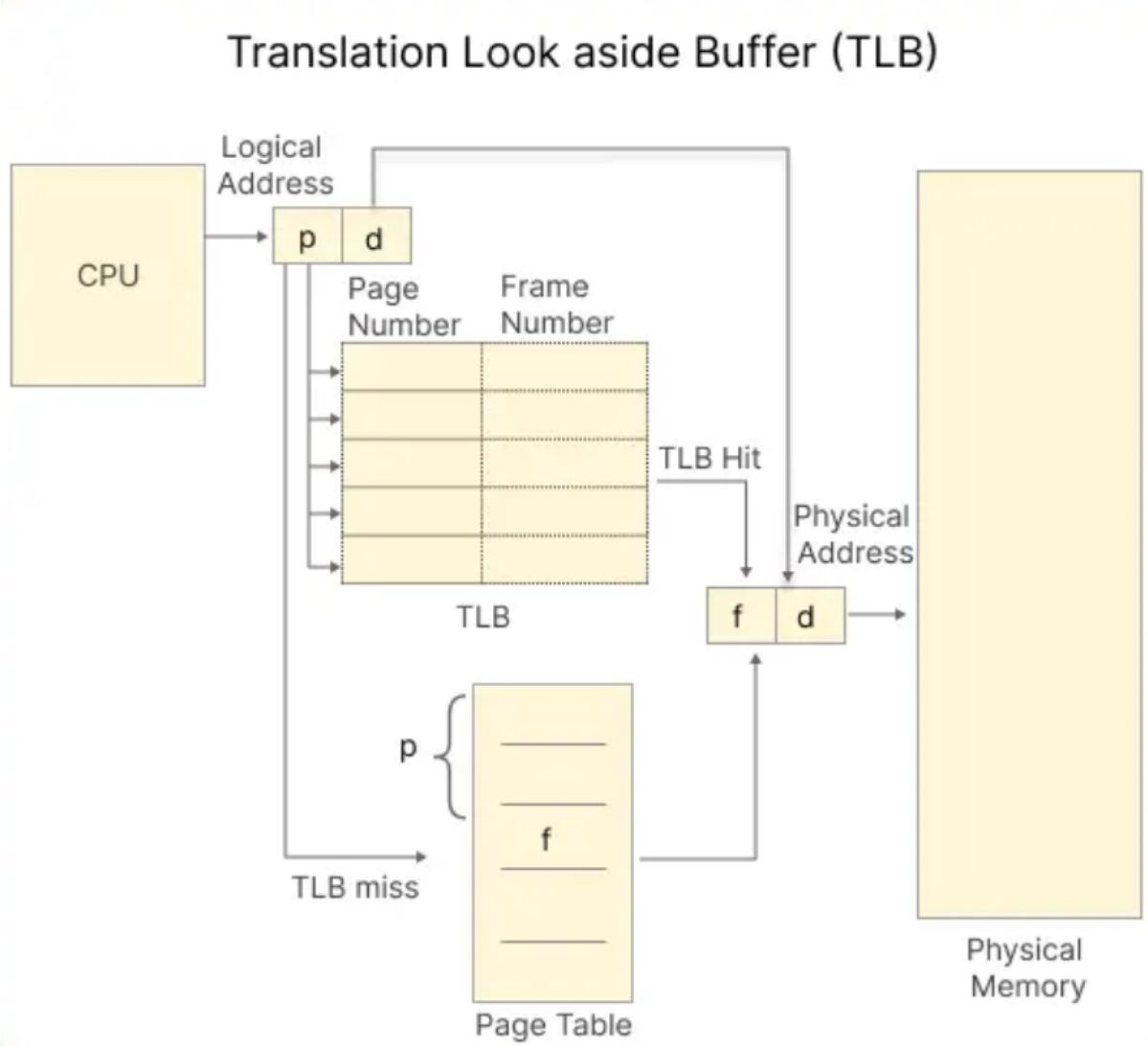
   The CPU uses the generated physical address, 0x9876678, to access the main memory and read the data from that location.




Normal paging, also known as eager paging or pre-paging, refers to a paging system where all the required pages of a process are loaded into physical memory before the process begins its execution. In normal paging, the entire address space of the process is loaded into memory, including both the pages that are immediately needed and those that may not be accessed for a while.

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 1,024.
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.

### Shared pages

- ➔ We can share the pages between the process
- ➔ Each page have the different virtual address of the page which will map to the same physical address
- ➔ Useful in multi-user or multi-process environments
- ➔ Unix uses “mmap” to create the shared page and access it.

### Demand Paging

Demand paging is a virtual memory management technique employed by modern operating systems to optimize memory usage and improve system performance. It allows the operating system to load pages into physical memory only when they are actually needed, rather than loading the entire process into memory at once.

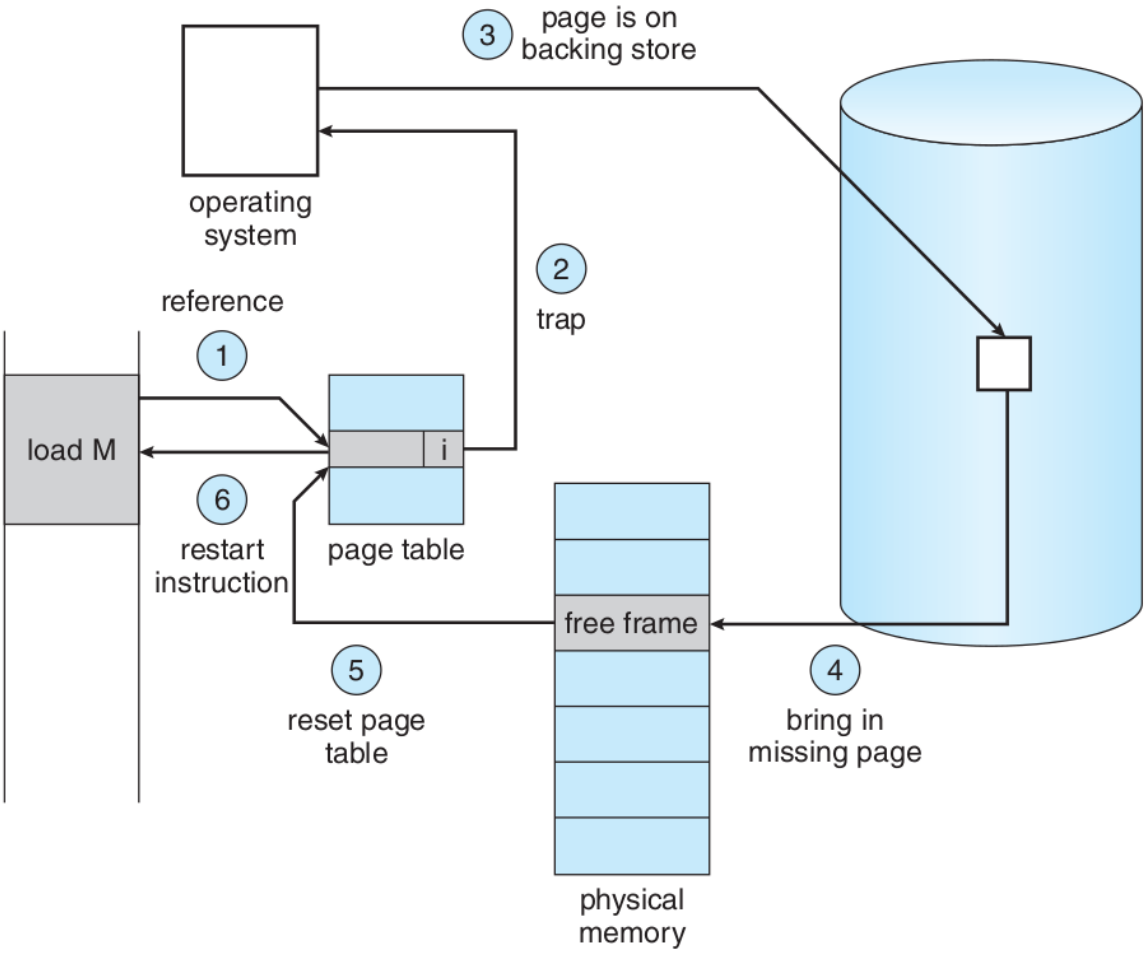



**Figure 9.6** Steps in handling a page fault.

### How is page fault handled?

When a page fault occurs, the operating system's page fault handler takes control to resolve the fault and bring the required page into physical memory.

- ➔ Trap to the operating system.
- ➔ Save the user registers and process state.
- ➔ Determine that the interrupt was a page fault.
- ➔ Check that the page reference was legal and determine the location of the page on the disk.
- ➔ Issue a read from the disk to a free frame:
  - ◆ Wait in a queue for this device until the read request is serviced.
  - ◆ Wait for the device seek and/or latency time.

    - ◆ Begin the transfer of the page to a free frame.
- ➔ While waiting, allocate the CPU to some other user (CPU scheduling, optional).
- ➔ Receive an interrupt from the disk I/O subsystem (I/O completed).
- ➔ Save the registers and process state for the other user (if step 6 is executed).
- ➔ Determine that the interrupt was from the disk.
- ➔ Correct the page table and other tables to show that the desired page is now in memory.
- ➔ Wait for the CPU to be allocated to this process again.
- ➔ Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

**What difficulties arise when restarting the instruction after page fault?**

- ➔ The difficulty described in the scenario arises when an instruction, such as the IBM System 360/370 MVC (move character) instruction, modifies multiple locations, and a page fault can occur during the execution of the instruction.
- ➔ This is particularly problematic when the source or destination blocks of the move operation straddle a page boundary. Additionally, if the blocks overlap, the source block may have been modified, making a simple restart of the instruction impossible.

**Two different solutions can address this issue:**

1. Computing and Accessing Both Ends of Blocks:

- In this solution, the microcode responsible for executing the instruction calculates the addresses of both ends of both blocks involved in the move operation.
- Before any modification is made to the blocks, the microcode attempts to access both ends of both blocks.
- If a page fault is going to occur, it will happen at this step, before any modifications have been made.

- By performing these access checks, the system can determine whether all the relevant pages are in memory and no page fault will occur during the move operation.
- If no page faults are detected, the move operation can proceed with the assurance that all necessary pages are already in memory.

2. Using Temporary Registers

- Another solution involves using temporary registers to hold the values of the overwritten locations during the move operation.
- Before the move operation begins, the original values of the overwritten locations are saved in these temporary registers.
- If a page fault occurs during the move operation, the system can detect it and handle it appropriately.
- In the event of a page fault, the system can restore the overwritten values from the temporary registers back into memory, effectively reverting the memory to its state before the instruction started.
- By restoring the original values, the system ensures that the memory is in the correct state to repeat the instruction without any inconsistencies.

## COPY-ON-WRITE

Copy-on-write (COW) is a memory management technique used in operating systems to optimize resource usage when creating new processes or threads. It allows multiple processes or threads to share the same memory pages until a write operation is performed, at which point a copy of the shared page is created to ensure data integrity.

## Segmentation

- Segmentation is a memory management technique used by operating systems to divide the physical memory into logical segments
- The physical memory is divided into variable-sized segments, each serving a specific purpose, such as code, data, stack, heap, etc
- Segments are identified by a unique segment number or segment descriptor

Segment Descriptor Table :

- The operating system maintains a data structure called the segment descriptor table (SDT) to store information about each segment.
- Each entry in the SDT is a segment descriptor that contains crucial details about a particular segment, such as its base address, length, access permissions, and other attributes.

Seg Descriptor :

- A segment descriptor is a data structure that describes the characteristics of a segment.
- It typically includes:
- Base Address: The starting address of the segment in the physical memory.
- Length: The size or length of the segment.
- Access Permissions: The privileges granted to the segment, such as read, write, or execute permissions.
- Other Attributes: Additional information like segment type, protection level, etc.

Generating Logical Addresses :

- During program execution, the CPU generates logical addresses to access segments.
- A logical address consists of two parts: the segment number and the offset within the segment.
- The segment number specifies the segment to be accessed, and the offset represents the location within that segment.

Memory protection :

- Segmentation enables memory protection by assigning access permissions to segments.
- The access permissions specified in the segment descriptor control the type of operations (read, write, execute) allowed on the segment.
- The CPU checks these permissions during address translation and raises an exception if an unauthorized access is attempted.

EXAMPLE

1. Dividing the Program into Segments:

- ➔ Suppose we have a C++ program that consists of three functions: addition(), subtraction(), and multiplication().
- ➔ We can divide the program into three segments: code segment, data segment, and stack segment.
- ➔ The code segment contains the executable instructions of the functions.
- ➔ The data segment holds the global variables used by the functions.
- ➔ The stack segment is used for storing local variables and function call information.

2. Generating the Segment Descriptor Table (SDT) :

- ➔ We create a Segment Descriptor Table (SDT) to store the information about each segment.
- ➔ For our example, the SDT may have the following structure:

| Segment Number | Base Address | Length | Permissions |
|---|---|---|---|
| 0 (Code) | 0x1000 | 0x2000 | Execute |
| 1 (Data) | 0x3000 | 0x1000 | Read/Write |

| | | | |
|---|---|---|---|
| 2 (Stack) | 0x4000 | 0x1000 | Read/Write |

➔ Each entry in the SDT represents a segment and contains information like the segment number, base address, length, and access permissions.

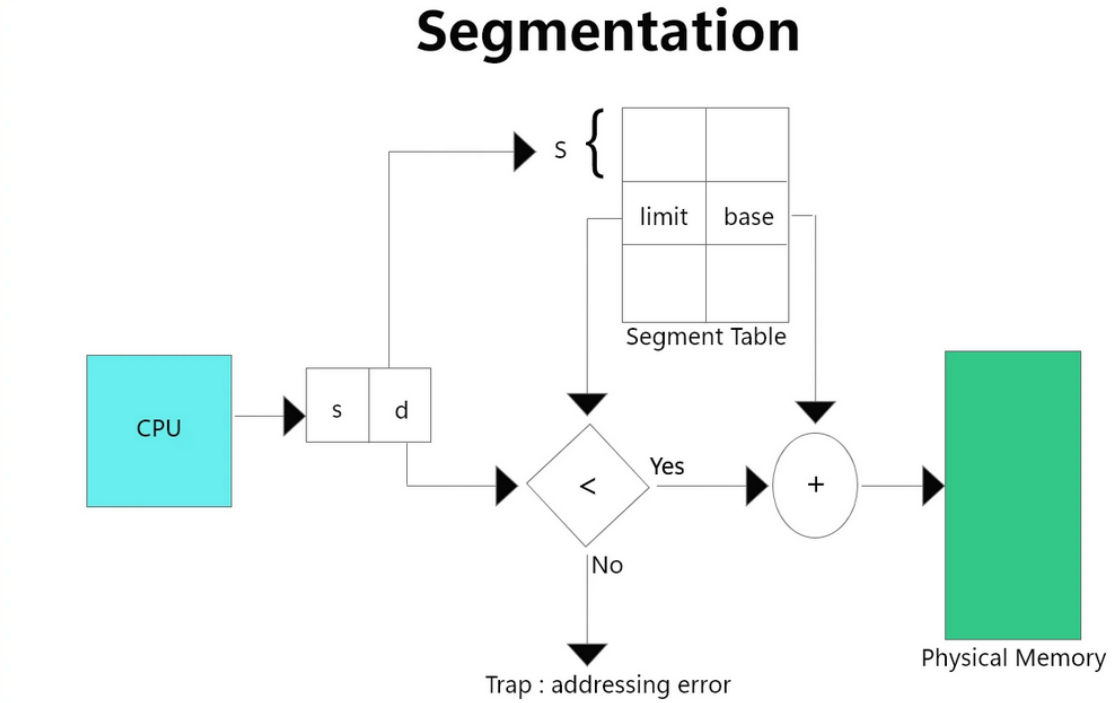
3. Address Translation :

➔ Suppose we want to execute the addition() function, which is located in the code segment.
➔ The CPU generates a logical address, which consists of a segment number (0 for the code segment) and an offset within the segment.
➔ For example, let's assume the logical address for the addition() function is 0x000A.
➔ The CPU uses the segment number (0) to access the corresponding segment descriptor from the SDT.
➔ In this case, the segment descriptor for the code segment has a base address of 0x1000 and a length of 0x2000.
➔ The CPU adds the offset (0x000A) to the base address (0x1000) to obtain the physical address, which becomes 0x100A.

4. Memory Access :

➔ With the physical address (0x100A) obtained through address translation, the CPU can access the memory location where the addition() function is stored.
➔ The CPU retrieves the machine instructions from the physical address and executes them.
➔ Similarly, if the program accesses global variables or uses the stack, the CPU performs address translation for data accesses or stack operations using the respective segment descriptors from the SDT.
➔ The CPU enforces memory protection by checking the access permissions specified in the segment descriptor during address

translation. Unauthorized memory accesses would result in exceptions or errors.




1. Segmentation Fault (SIGSEGV) :

   Error Code: 11

   Description: Occurs when a program tries to access memory that it is not allowed to access or when there is an invalid memory reference.

2. Bus Error (SIGBUS) :

   Error Code: 10

   Description: Indicates an error related to an invalid memory access, such as misaligned memory or accessing memory that does not exist.

3. Illegal Instruction (SIGILL) :

   Error Code: 4

   Description: Indicates that the CPU has encountered an illegal or unrecognized instruction during program execution.

4. Floating-Point Exception (SIGFPE) :

   Error Code: 8

   Description: Indicates that a floating-point arithmetic operation has encountered an exceptional condition, such as division by zero or an invalid operation.

5. Abort (SIGABRT) :

   Error Code: 6

   Description: Typically used by programs to intentionally terminate due to a critical error or when an assertion fails.

6. Segmentation Violation (SIGSEGV) :

   Error Code: N/A

   Description: Similar to a segmentation fault, this error indicates an attempt to access an invalid or protected memory segment.

7. Stack Overflow :

   Error Code: N/A

   Description: Occurs when a program's call stack exceeds its maximum size, often due to excessive recursion or large stack allocations.

| 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 |

## Page replacement algorithms

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

### What is page replacement Algorithms and  Why do we use it ?

- Page replacement algorithms are used in operating systems to determine which pages should be removed from memory when a page fault occurs and a new page needs to be brought in. They are responsible for selecting the victim page that will be removed from memory.
- We use page replacement algorithms to efficiently manage memory resources by maximizing the utilization of available physical memory. These algorithms aim to minimize page faults and optimize the overall system performance by making intelligent decisions on which pages to replace.

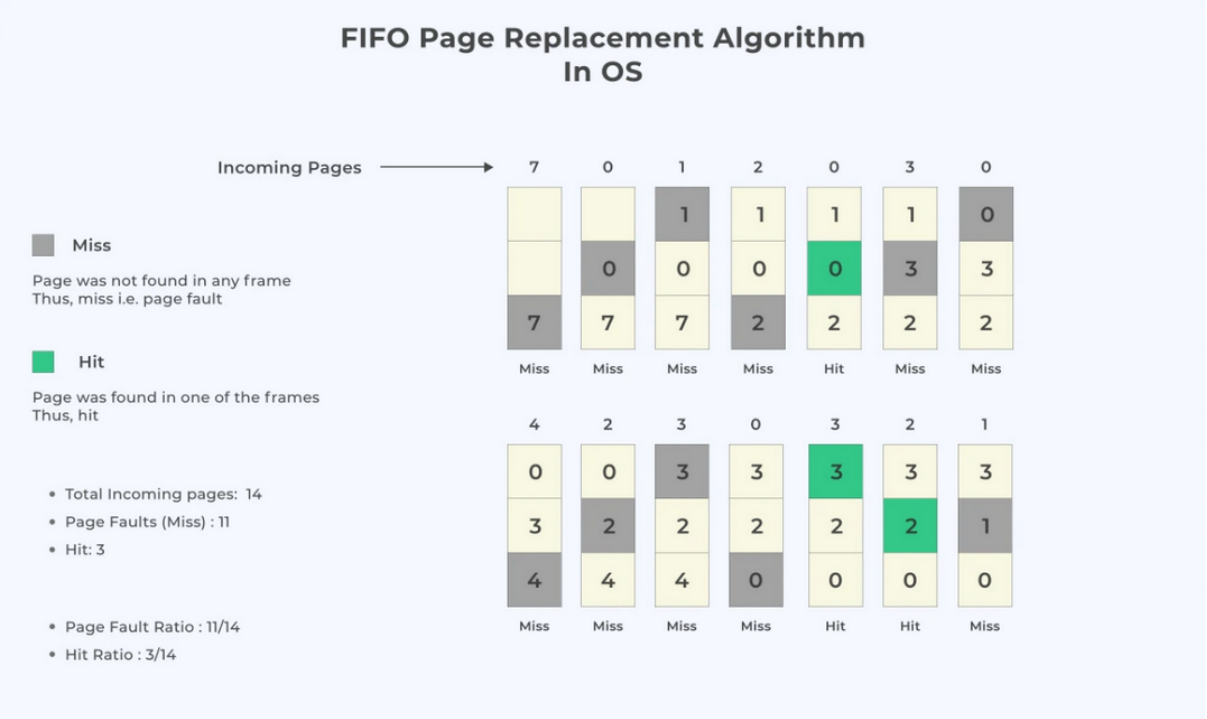
**What is a reference string in PRA ?**

We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a reference string.

EX - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

1. FIFO algorithm

- FIFO is one of the simplest page replacement algorithms used in operating systems.
- It follows a queue-like structure and operates on the principle of "first in, first out."
- Each page that is brought into memory is added to the end of the queue.
- It simply maintains  a queue data structure
- When a new page is brought into memory, it is added to the end of the queue.
- When a page needs to be replaced, the page at the front of the queue (the oldest page) is removed.




Belady's Anomaly

Think – Now, one should think that increasing the page size(frame size as page size = frame size) will lead to less page fault, right!! since there are more chances of elements to be present in the queue.

But, that's not always the case. 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)

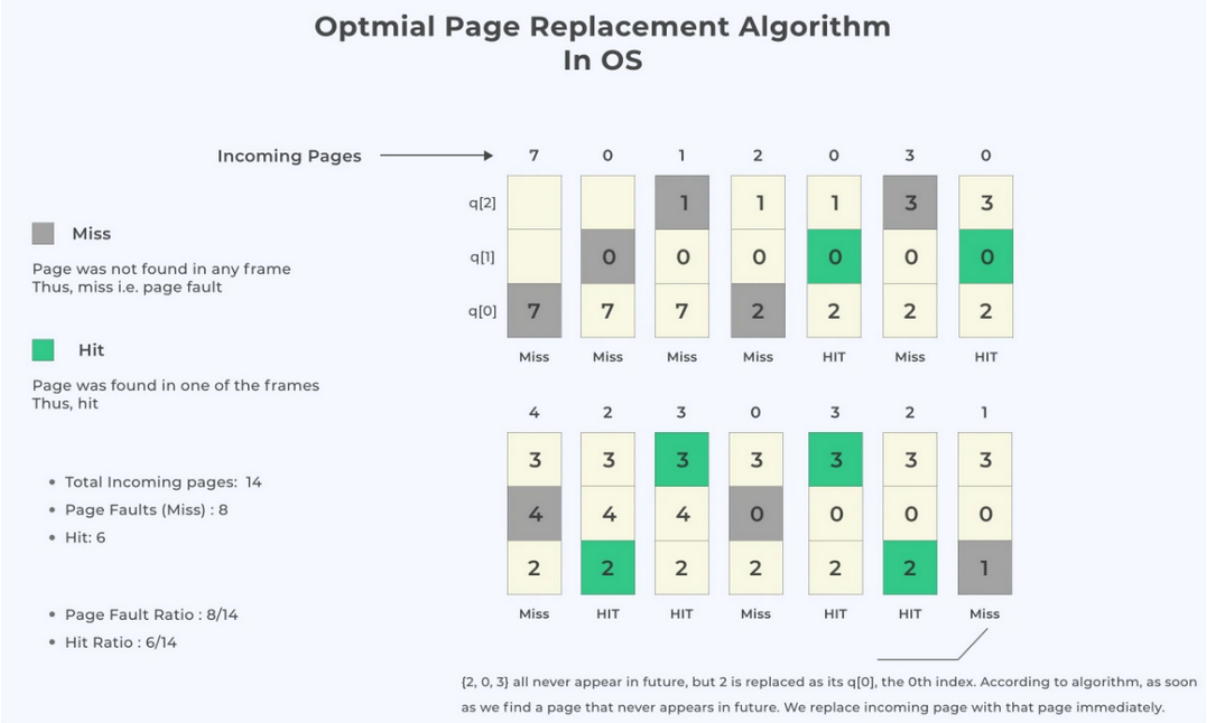
- 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4
    - Using 3 slots gives 9-page faults
    - Using 4 slots gives 10-page faults

Optimal page replacement algorithm or [MIN Algorithm]

One result of the discovery of Belady's anomaly was the search for an optimal page-replacement algorithm—the algorithm that has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly. Such an algorithm does exist and has been called OPT or MIN. It is simply this:

*Replace the page that will not be used for the longest period of time.*

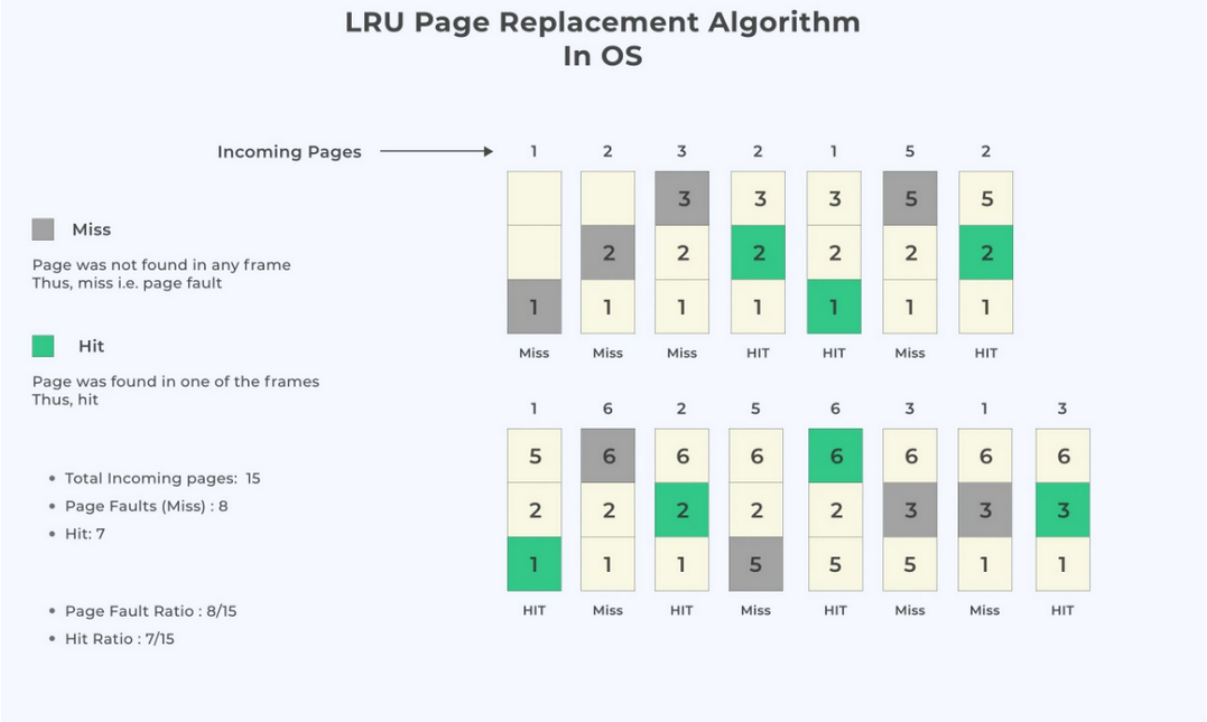
- Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests.
- The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.




LRU

- If the optimal algorithm is not feasible, perhaps an approximation of the optimal algorithm is possible.

- The key distinction between the FIFO and OPT algorithms (other than looking backward versus forward in time) is that the FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be used.
- If we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time.
- This approach is the least recently used (LRU) algorithm.




Least Frequently Used

Basically in the current stack at any iteration we choose that element for replacement which has the smallest count in the incoming page stream.

Most Frequently Used

Basically in the current stack at any iteration we choose that element for replacement which has the highest count in the incoming page stream.

### Which one out of FIFO,LRU,LFU,MFU is best ?

The effectiveness of a page replacement algorithm in reducing page faults depends on the specific characteristics of the workload and access patterns of the running applications. However, among the commonly used algorithms (FIFO, LRU, LFU, MFU), the LRU (Least Recently Used) algorithm is generally considered one of the best for reducing page faults in a wide range of scenarios

### Why ?

- The LRU algorithm selects the page for replacement that has not been accessed for the longest time.
- It assumes that the pages that have not been used recently are less likely to be used in the near future.
- LRU takes advantage of temporal locality, which suggests that recently accessed pages are more likely to be accessed again soon.

Also it Reflects Real-World Usage Patterns :

- LRU closely aligns with the natural behavior of many applications, as they often exhibit temporal locality by frequently accessing recently used pages.
- It matches well with the access patterns of most programs, making it a practical choice for reducing page faults.

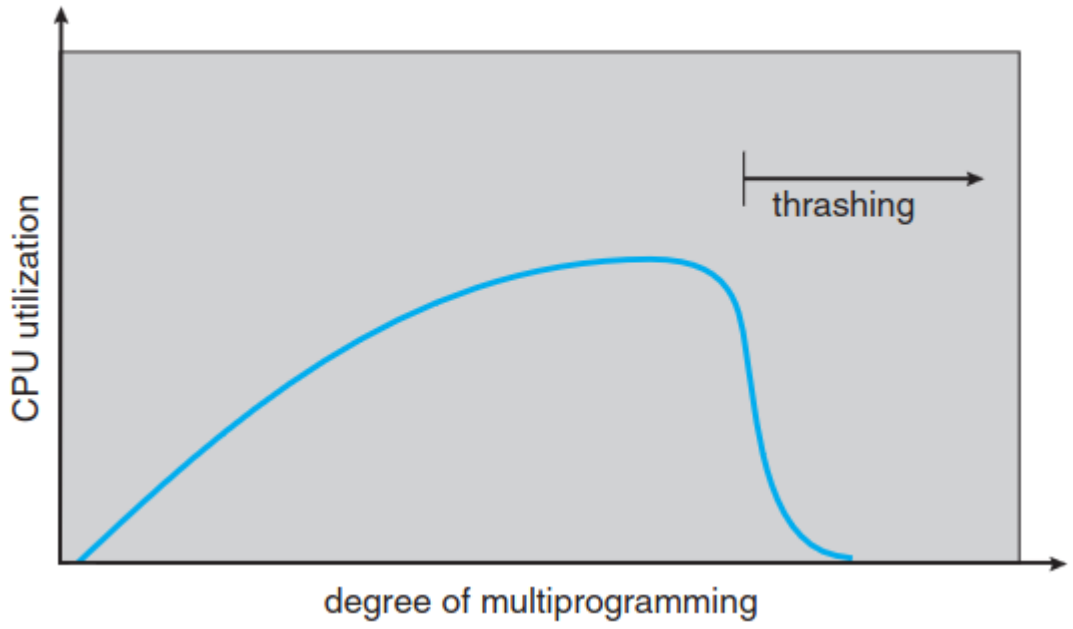
### Thrashing

Definition and Impact:

- Thrashing refers to the high paging activity in a virtual memory system.
- It occurs when a process spends more time paging than executing, leading to severe performance problems.

Cause of Thrashing:

- Insufficient frames allocated to a low-priority process result in suspending its execution.
- When a process lacks the necessary frames to support actively used pages, it experiences frequent page faults.
- The process continuously replaces pages that it immediately needs again, creating a cycle of repeated page faults.
- Scenario and Behavior:
- In a multiprogramming environment, the operating system monitors CPU utilization.
- If CPU utilization is low, the system introduces a new process to increase multiprogramming.
- A global page-replacement algorithm replaces pages without considering the process they belong to.
- When a process enters a new phase and requires more frames, it starts faulting and taking frames from other processes.
- The affected processes also fault, leading to a cascade effect of page faults and increased paging activity.
- As processes queue up for the paging device, the ready queue empties, and CPU utilization decreases.



Static and Dynamic Loading

Static :

- Static loading refers to the process of loading all the necessary executable code and data into memory before the program starts its execution.
- In static loading, the entire program, including libraries and dependencies, is loaded into memory as a single unit.
- The addresses of functions and variables are typically fixed and determined at compile-time.
- The operating system reserves memory for the program and assigns the necessary resources in advance.
- The loaded program remains in memory throughout its execution.
- Examples of statically loaded programs include traditional executable files (e.g., .exe, .elf) that are compiled and linked before execution.

Dynamic :

- Dynamic loading is a technique where executable code and data are loaded into memory on-demand during program execution.
- In dynamic loading, the program is divided into smaller units or modules, and only the necessary modules are loaded when required.
- Dynamic loading can save memory resources by loading only the required modules, reducing the initial memory footprint.
- Examples of dynamically loaded modules include libraries, plugins, and shared objects (.dll, .so) that can be loaded and unloaded as needed.

*Static loading is commonly used for standalone executable files, while dynamic loading is often used for libraries and modular systems.*

*[EXTra Ahead]*

.so (Shared Object) Files:

- .so files are used for shared libraries in Unix-like systems, including Linux and other Unix variants.
- Shared libraries provide a mechanism for code and data reuse among multiple programs. They contain compiled code, functions, and data that can be dynamically linked and loaded into a program's memory at runtime.
- .so files are dynamically linked during runtime, allowing multiple programs to share a single copy of the library in memory.
- The dynamic linker/loader (ld.so or ld-linux.so) is responsible for resolving symbols and dependencies of .so files when they are loaded into a program's memory.
- Examples of .so files include libm.so (the math library), libpthread.so (the POSIX threads library), and user-defined shared libraries like libimage.so.

.dll (Dynamic Link Library) Files:

- .dll files are used for shared libraries in Windows operating systems.
- Similar to .so files, .dll files contain compiled code, functions, and data that can be dynamically linked and loaded into a program's memory at runtime.
- .dll files are dynamically linked during runtime, allowing multiple programs to share a single copy of the library in memory, just like .so files in Unix-like systems.
- Windows uses the dynamic link library manager (kernel32.dll) to handle the loading, linking, and resolving of symbols and dependencies of .dll files.
- Examples of .dll files include kernel32.dll (the Windows API library), user32.dll (the user interface library), and third-party libraries distributed as .dll files.

## What is linking in Operating System ?

Linking in operating systems refers to the process of combining multiple object files or libraries together to create a single executable file. It is an essential step in the software development process, particularly during compilation, to transform source code into an executable program. The linking process resolves references between different modules, ensures that all necessary code and data are available, and prepares the program for execution.

There are typically two types of linking involved in the software development process:

Compile-time Linking:

- Compile-time linking, also known as static linking, occurs during the compilation phase.
- It involves combining object files, generated by compiling individual source code files, into a single executable file.
- During compile-time linking, the linker resolves symbols (functions, variables) between object files and resolves dependencies.
- The resulting executable file contains all the necessary code and data from the object files, making it self-contained.
- Static libraries (.lib, .a) are commonly used for compile-time linking.

Run-time Linking:

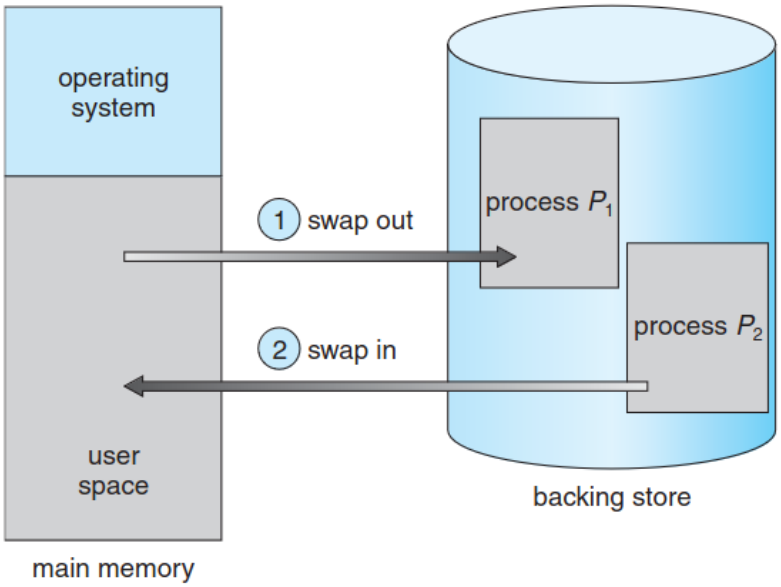
- Run-time linking, also known as dynamic linking, occurs during the program's execution or loading into memory.
- It involves resolving references to external libraries or modules that the program relies on.
- During run-time linking, the dynamic linker/loader locates and loads the required shared libraries or dynamically linked libraries (DLLs, .so) into memory.
- The dynamic linker resolves addresses and dependencies between the program and the shared libraries.
- Dynamic linking allows multiple programs to share a single copy of a library, reducing redundancy and memory usage.

### What is the difference between linking and loading ?

Linking is the process of combining code and data from multiple object files to create an executable or shared library, while loading is the process of bringing the executable or shared library into memory for execution.

### What is Swapping in Operating System ?

➔ A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution




➔ **Figure 8.5** Swapping of two processes using a disk as a backing store.

➔ Linux has there own swap area which extends the memory by adding virtual memory

➔ Although most operating systems for PC s and servers support some modified version of swapping, mobile systems typically do not support swapping in any form.

- ◆ They generally uses flash memory which has limited number of writes

## Memory Mapped Files

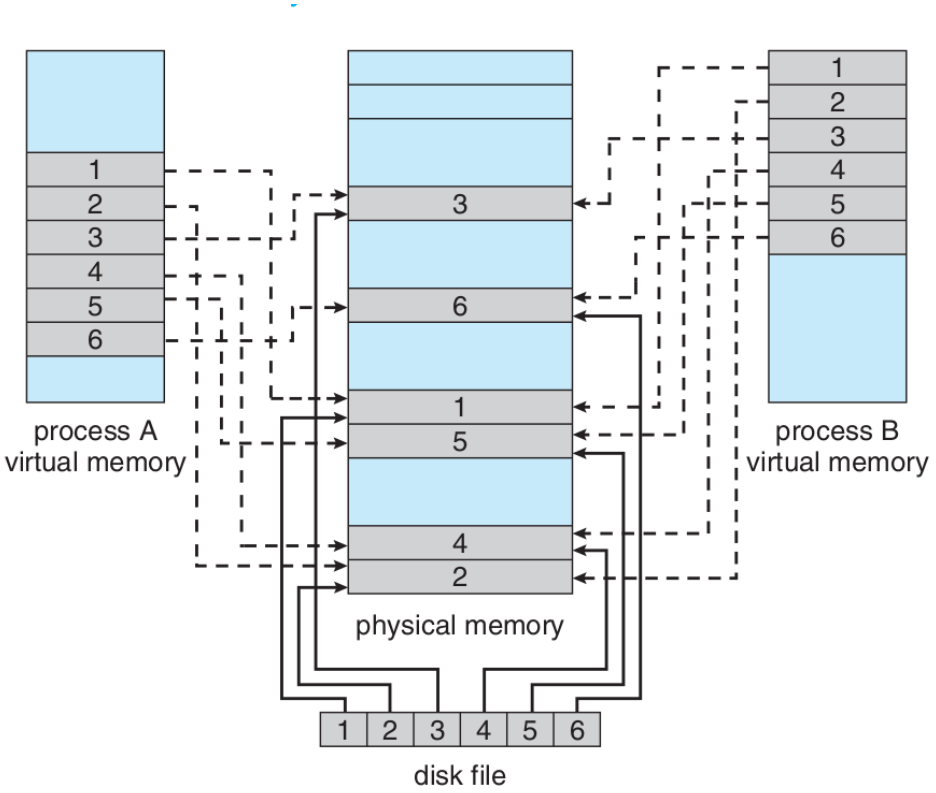



**Figure 9.22** Memory-mapped files.

- ➔ Memory mapping a file is accomplished by mapping a disk block to a page (or pages) in memory.
- ➔ A page-sized portion of the file is read from the file system into a physical page
- ➔ Subsequent reads and writes to the file are handled as routine memory accesses.
- ➔ It is faster then read() and write() system calls
- ➔ Multiple process can access the shared file (as shared memory)
- ➔ Sometimes it is used in shared memory

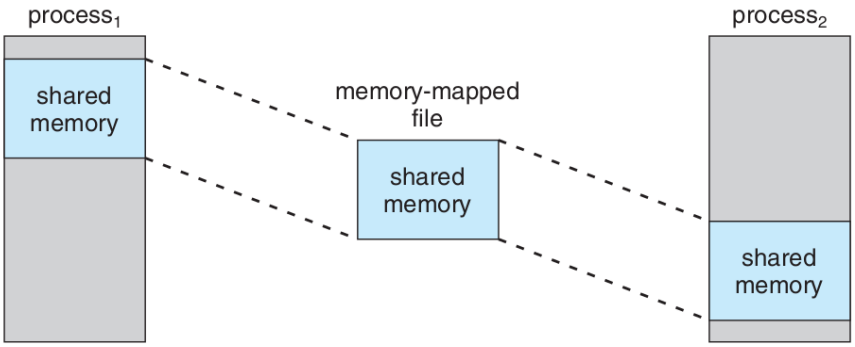



**Figure 9.23** Shared memory using memory-mapped I/O.

## File systems

### What is file systems, Why is it used ?

➔ A file system is a software component in an operating system that manages how data is stored, organized, and accessed on storage devices such as hard drives or SSDs.

### Why is it used ?

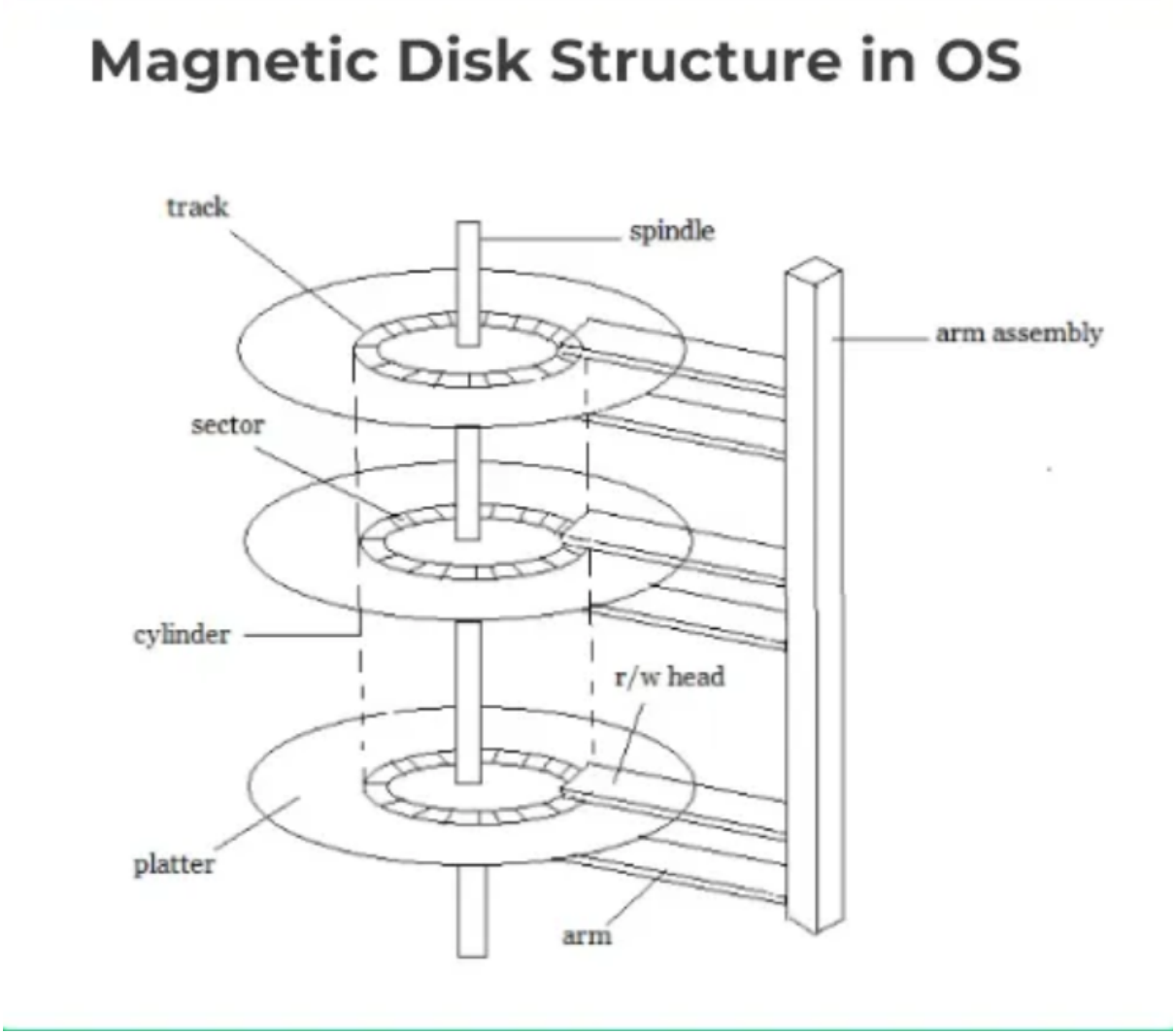
➔ File systems are used to enable data persistence, allowing files to be stored and accessed even after the system is powered off or restarted.
➔ They provide a hierarchical structure with directories and subdirectories, facilitating organization and efficient retrieval of files.
➔ File systems handle file access permissions and security
➔ They implement various techniques for efficient data storage and retrieval.

| 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 |

Magnetic Disk Structure in Operating System

- ➔ In computers the secondary storage typically is formed of stacked up magnetic disks on the top of one another.
- ➔ One single unit of such a 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.

*[TIP] - The length of the tracks near the center is less than the length of the tracks farther from the center.*



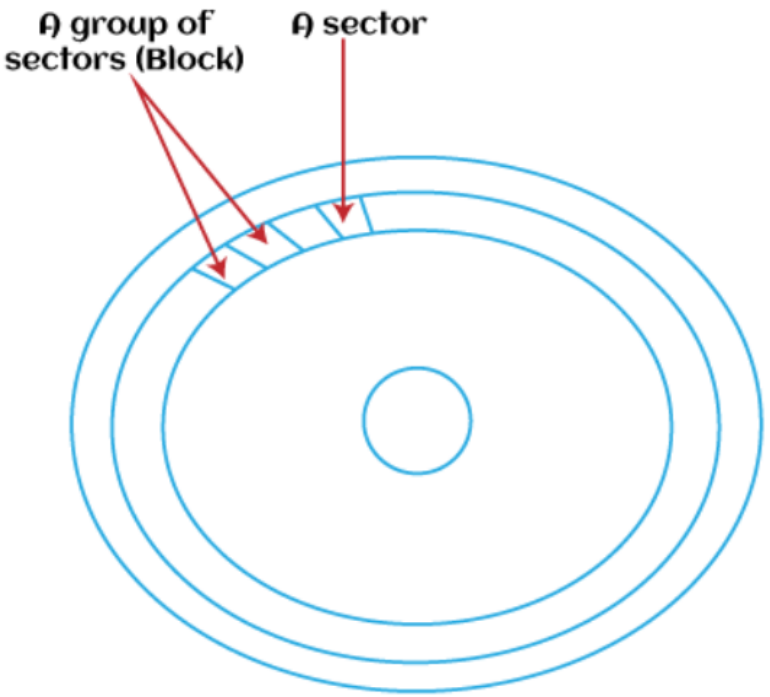

HDD Vs SDD

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

Block :

- A block is a fixed-size unit of data storage used by file systems to manage and organize data on storage devices
- Each block has one or more sector




Sector :

A sector is the smallest addressable unit on a storage device, and multiple sectors are typically combined to form a block in a file system

Device driver :

A device driver is a software component that serves as an interface between an operating system and a specific hardware device.

★ A file-control block (FCB ) (an inode in UNIX file systems) contains information about the file, including ownership, permissions, and location of the file contents.

★ UNIX uses the UNIX file system (UFS)
★ Windows supports disk file-system formats of FAT , FAT 32, and NTFS (or Windows NT File System)
★ Linux supports over forty different file systems, the standard Linux file system is known as the extended file system, with the most common versions being ext3 and ext4.

Swap Space Management

Swapping in that setting occurs when the amount of physical memory reaches a critically low point and processes are moved from memory to swap space to free available memory.

RAID (Redundant Array of Independent Disks)

- ➔ RAID 0
  - ◆ Simple
  - ◆ No redundancy
- ➔ RAID 1
  - ◆ Mirroring
  - ◆ Easy
  - ◆ Costly
- ➔ RAID 2
  - ◆ Memory style error correcting codes
  - ◆ Contains parity of the bits record(set bit count and unset)
  - ◆ Hence can check bit wise error by memory system
  - ◆ First bit of each byte stored on disk1 and second disk2 and so on
  - ◆ Error correcting are stored in other extra disks
- ➔ RAID 3
  - ◆ bit-interleaved parity organization
  - ◆ Improve 2 by using sector based parity checking
  - ◆ Easy
  - ◆ Less Storage
- ➔ RAID 4
  - ◆ block-interleaved parity organization
  - ◆ USes block level parity than the bit
  - ◆ Separate blocks parity in separate N disks
- ➔ RAID 5

- Block-interleaved distributed parity organization
- Avoid single block parity on same block
- Ith block parity sorted on (i mod N)+1

➔ RAID 6

- P + Q redundancy
- Same as RAID 5 but with extra information
- Instead of parity, error correction codes Reed Solomon code is used

RAID problems

➔ RAID does not always assure that data is available for the operating system and its users.
➔ A pointer to a file could be wrong, for example, or pointers within the file structure could be wrong. Incomplete writes, if not properly recovered, could result in corrupted data.
➔ Some other process could accidentally write over a file system's structures, too.
➔ RAID protects against physical media errors, but not other hardware and software errors.
➔ As large as is the landscape of software and hardware bugs, that is how numerous are the potential perils for data on a system.

## Protection

➔ The most common approach to the protection problem is to make access dependent on the identity of the user.
➔ Different users may need different types of access to a file or directory.
➔ The most general scheme to implement identity-dependent access is to associate with each file and directory an access-control list (ACL) specifying user names and the types of access allowed for each user.
➔ There are three types:

- Owner : The user who created the file is the owner.
- Group : A set of users who are sharing the file and need similar access is a group, or work group.
- Universe : All other users in the system constitute the universe

File system architectures

- ➔ Hierarchical File System: This architecture uses a tree-like structure where directories can contain subdirectories and files. It allows for easy organization and navigation of files through a hierarchical directory structure.
- ➔ Flat File System: In this architecture, files are stored in a single-level directory without any subdirectories. All files reside in a single directory, making it simple but potentially limiting in terms of organization and scalability.
- ➔ Network File System (NFS): NFS is a distributed file system architecture that allows files to be accessed and shared over a network. It enables remote clients to mount and access files from a central server, facilitating file sharing and collaboration.
- ➔ Distributed File System (DFS): DFS is a decentralized file system architecture where files are distributed across multiple servers or nodes in a network. It provides improved fault tolerance, scalability, and load balancing by distributing file storage and access across multiple servers.

Components of File system [ Logical ]

- File : A file is a named collection of data or information that is stored on a storage device. It represents the basic unit of data in a file system.
- Directory :A directory, also known as a folder, is a container that holds files and other directories. It provides a hierarchical structure for organizing and locating files within a file system.
- File Control Block (FCB) : The File Control Block, also known as an Inode (Index Node) in some file systems, stores metadata about a file, including its name, location, size, access permissions, timestamps, and other attributes. It serves as a reference to locate and manage the associated file data.
- Superblock : The superblock is a data structure that contains essential information about the file system, such as the file system type, size, location of the free space, and other parameters. It is typically located at the beginning of the file system and provides critical data for mounting and accessing the file system.

- Free Space Management : This component tracks the available free space on the storage device. It maintains information about unused blocks or clusters and handles the allocation and deallocation of storage space for files.
- File Allocation Table (FAT) or Inode Table : The file allocation table or inode table is a data structure that maintains a mapping between file names, file control blocks (FCBs), and their corresponding physical storage locations. It helps in locating and accessing files efficiently.

★ FAT32 - TO serve 32 bit addresses
★ NTFS ( New Technology File System) Took over FAT32
★ Inode Table ( Index Node Table ) - FCB —> Actual location, ext3,ext4,ext5(extended fs)

**Inode table and fat both use the same data structure which maps logical address to physical location of file ?**

➔ No, the inode table and the File Allocation Table (FAT) do not use the same data structure to map logical addresses to physical locations of files

★ Inode Table (Unix-like file systems):

Consider a Unix-like file system with an inode-based structure. Let's say we have a directory that contains two files: "file1.txt" and "file2.txt". In the inode table, each file will have a corresponding inode entry that stores metadata about the file, such as its size, permissions, timestamps, and pointers to the data blocks that hold the file's content. The inode table allows for direct access to the metadata and data blocks of each file.

Example inode table:

| Inode Number | File Name | Size (Bytes) | Data Block Pointers |
|---|---|---|---|
| 1 | file1.txt | 1024 | 5, 10, 15 |
| 2 | file2.txt | 512 | 3, 8 |

In this example, the inode table contains information about file1.txt and file2.txt. Each file has an inode number, file name, size, and pointers to the corresponding data blocks where the file's content is stored.

★ File Allocation Table (FAT) (FAT file systems):

Now let's consider a FAT file system where the file allocation table is used to track the allocation status and location of data blocks. The FAT stores a sequence of entries, with each entry corresponding to a cluster (a group of contiguous data blocks) on the storage device. The FAT entries maintain the information about which clusters are allocated and the next cluster in the file's chain.

Example FAT:

| Cluster Number | Next Cluster |
|---|---|
| 0 | 2 |
| 1 | 4 |
| 2 | 5 |
| 3 | END |
| 4 | END |
| 5 | 7 |
| ... | ... |

In this example, the FAT contains entries that map the cluster numbers to the next cluster in the file's chain. Each file in the file system will have a starting cluster number, and by following the chain of entries in the FAT, the file system can determine the sequence of clusters that make up the file's content.

Key Differences:

- Structure: Inode table is associated with inode-based file systems, whereas the FAT is used in FAT file systems.
- Purpose: Inode table stores metadata about files and pointers to data blocks, enabling direct access to file information. The FAT tracks the allocation and location of data clusters to provide a chain-like structure for storing file data.
- Mapping: Inode table maps files to their corresponding inodes, whereas the FAT maps clusters to their next clusters in the file's chain.

File System Interface

Allows to perform File operations: create, open, read, write, close, All features

**What is the importance of file attributes and metadata ?**

- ➔ File Identification : File attributes help in uniquely identifying files within a file system.
- ➔ Efficient File Management : Metadata enables effective organization and categorization of files, making file management tasks easier.
- ➔ Access Control : File attributes and permissions ensure that only authorized users can access, modify, or execute files, enhancing file system security.

File organization techniques :

1. Contiguous  Files are stored as continuous blocks of data on the storage medium.
2. Linked : Files are stored in non-contiguous blocks, and each block contains a pointer to the next block.
3. Indexed : Files are stored in separate blocks, and an index structure is used to keep track of the file blocks.

File allocation techniques:

1. Allocation tables : A table maintains a list of allocated and free blocks on the storage device.
2. Linked lists : Each block contains a pointer to the next block in the file.
3. Index nodes : An index structure is used to map logical file blocks to physical blocks on the storage device.

File system mounting and unmounting:

- Normal file system mount,unmount
- Example of External Hard disk drive

Low-level disk organization:

Low-level disk organization refers to the physical organization of data on a disk. It involves dividing the disk into sectors and tracks, which are the smallest units of storage. Sectors are typically 512 bytes in size and tracks are concentric circles on the disk. This organization allows the disk to read and write data efficiently.

➔ Data structures :

- Inodes : Inodes are data structures used by file systems to store metadata about files, such as file permissions, ownership, timestamps, and pointers to data blocks. Each file has an associated inode that contains this information.
- File Control Blocks (FCBs) : FCBs are data structures used by the operating system to manage files. They contain information about a file's location, size, and attributes. FCBs are used to perform file operations like opening, reading, and writing files.
- Directory structures : Directory structures are used to organize files into a hierarchical structure. Directories contain entries that associate file names with their corresponding inodes or FCBs. This allows for efficient file lookup and management.

➔ File system layout :

The file system layout consists of three main components: boot block, superblock, and data blocks.

- Boot block : The boot block is the first sector on the disk and contains the bootloader program. It is responsible for loading the operating system into memory during the boot process.
- Superblock : The superblock is a data structure that stores important information about the file system, such as the total number of blocks, the size of each block, and the location of the root directory.

- Data blocks : Data blocks are the actual units of storage where file data is stored. They are allocated to files and directories as needed and are typically of fixed size.

➔ File system consistency and recovery mechanisms:

- File system consistency refers to the state where the file system is free from errors or inconsistencies.
- To maintain consistency, file systems employ various mechanisms such as File System Check (fsck) in Unix,Linux, (Chkdsk) in Windows.

Disk Scheduling algorithm :

➔ FCFS (First-Come-First-Serve) :

- FCFS is a simple disk scheduling algorithm that handles requests in the order they arrive.
- It works by servicing the requests in the order they are submitted to the disk.
- This algorithm suffers from the "elevator effect" where the head moves back and forth on the disk, resulting in poor performance.

➔ SSTF (Shortest Seek Time First) :

- SSTF is a disk scheduling algorithm that selects the request with the shortest seek time from the current head position.
- It aims to minimize the seek time by selecting the closest request to the current head position.
- SSTF is more efficient than FCFS in terms of reducing the total distance traveled by the disk head.

➔ SCAN :

- SCAN, also known as Elevator algorithm, moves the disk arm from one end of the disk to the other, serving requests along the way.
- It starts servicing requests in one direction until it reaches the end, and then reverses direction.
- SCAN provides a better response time compared to FCFS and reduces the average seek time.

➔ C-LOOK (Circular LOOK) :

- C-LOOK is an improvement over the SCAN algorithm.
- Instead of going all the way to the end of the disk, C-LOOK only goes to the last request in the current direction and then reverses the direction.
- This reduces unnecessary head movement and improves the disk's throughput and response time.

Access Control Lists (ACLs) :

- ACLs provide more granular control over file permissions by allowing specific permissions for individual users or groups.
- ACLs can be used in addition to or instead of traditional Unix-style permissions.
- They enable setting permissions for multiple users or groups on a single file or directory.
- With ACLs, permissions can be granted or denied for specific actions, such as reading, writing, executing, or deleting files.
- ACLs enhance security and flexibility by allowing fine-grained access control to resources.

Let's consider a directory named "documents" with the following ACL entries:

➔ User: John
  ◆ Permissions: Read, Write
➔ User: Mary
  ◆ Permissions: Read, Write, Execute
➔ Group: Employees
  ◆ Permissions: Read
➔ Other (Everyone else)
  ◆ Permissions: No access

Performance optimization techniques :

➔ Buffering :
- Buffering involves using a region of memory to temporarily store data during input/output (I/O) operations.
- It helps reduce the number of direct disk accesses by buffering data in memory before reading or writing it to the disk.
- Buffering can improve performance by reducing disk I/O latency and increasing the efficiency of data transfers.

➔ Caching :
- Caching is a technique that stores frequently accessed data in a cache, which is a faster and smaller memory closer to the processor.
- When a program requests data, the operating system checks the cache first. If the data is found in the cache, it is retrieved quickly, avoiding the need to access slower storage devices.
- Caching can significantly speed up data retrieval and improve overall system performance by reducing the reliance on slower storage devices.

➔ Prefetching :
- Prefetching involves predicting the future data or instructions that a program is likely to access and retrieving them in advance.
- By fetching data or instructions proactively, prefetching reduces the latency associated with accessing data from slower storage devices.
- Prefetching can be performed at various levels, such as hardware prefetching in processors or software-based prefetching in the operating system.

## Virtual File system

- It is integrating multiple file system into single directory structure
- This type of file system uses object-oriented techniques to simplify, organize, and modularize the implementation.
- The four main object types defined by the Linux VFS are:
    - The inode object, which represents an individual file
    - The file object, which represents an open file
    - The superblock object, which represents an entire file system
    - The entry object, which represents an individual directory entry
- Each has set of operations they provide abstract view of single file system

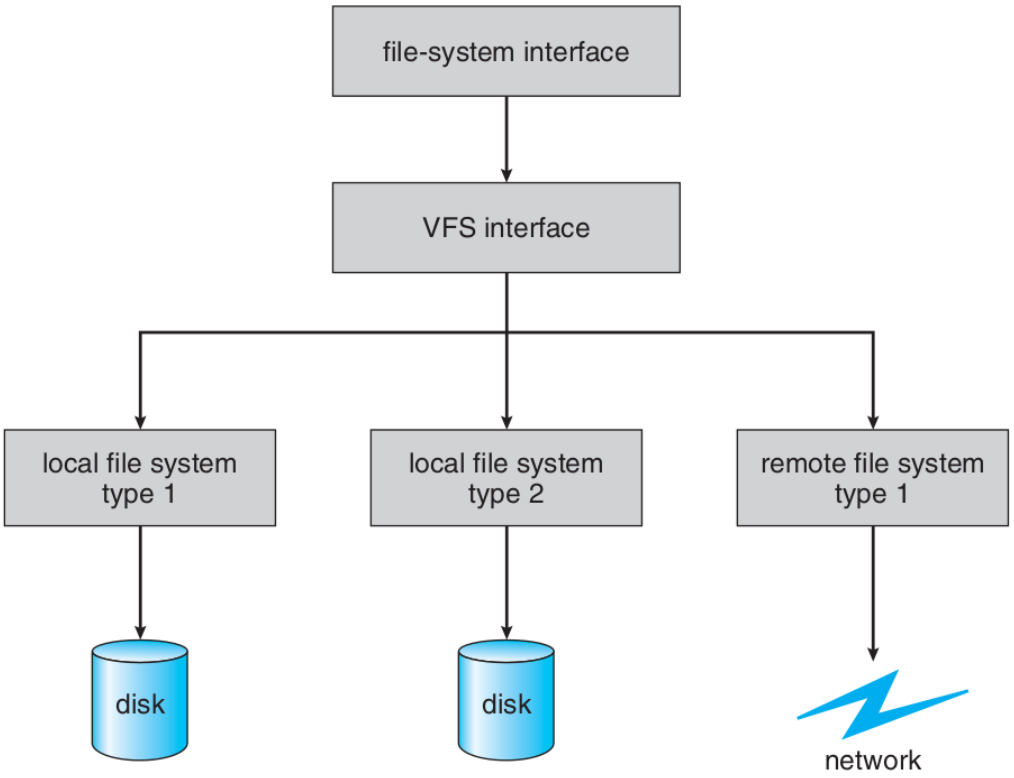



**Figure 12.4** Schematic view of a virtual file system.