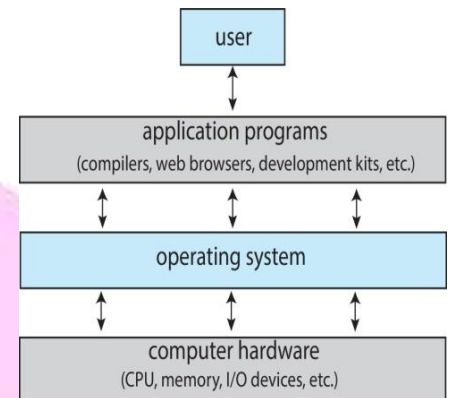


Operating System

- Computer System divided into 4 components- the hardware, the operating system, the application program & the user.
- The hardware (the CPU, memory & I/O devices) provides the basic Computing resources for the system.
- The application program defines the way in which these resources are used to solve user's computing problems.
- The OS coordinates hardware use among the various application programs for the various users & also controls the hardware.
- Operating System (OS): - An Operating system acts as an interface between user & hardware.
- An OS is a program that manages the computer hardware. It also provides a basis of application program & acts as an intermediary between the computer user and the computer hardware.

**Computer Booting Process**

There are two types: Cold Booting (starting the computer from an off state) and Warm Booting (restarting the computer). Here's a detailed step-by-step breakdown of the booting process:

- Power On and check hardware components like the CPU, RAM, keyboard, and storage drives.
- Once hardware setup is complete, the BIOS/UEFI looks for a bootable device
- The BIOS/UEFI hands over control to the bootloader, which is stored on the primary boot device.
- The bootloader loads the operating system's kernel into memory.
- The kernel starts the first system process, usually called init (or systemd in newer Linux distributions). For Windows, the process is called wininit.exe.
- The operating system loads essential background services and daemons (programs running in the background) necessary for the functioning of the system.
- The login screen appears, allowing the user to authenticate and start a user session.
- Once a user logs in, the system loads the user's environment, and the computer is ready for use.

Operating System operation

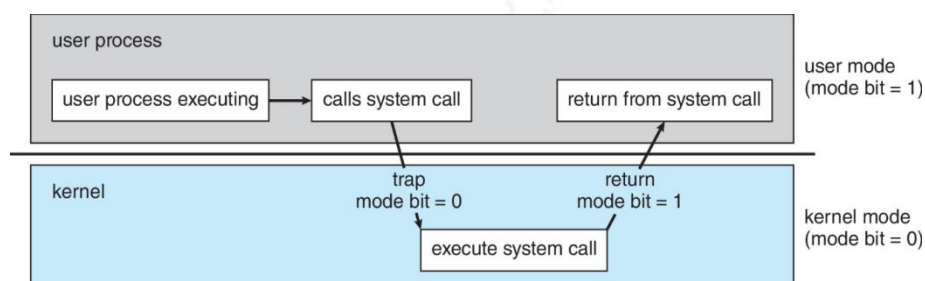
- Modern operating systems are interrupt driven.
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
- An Interrupt Service Routine is provided that is responsible for dealing with the interrupt.
- A properly designed operating system must ensure that an incorrect (or malicious) program can not cause other programs to execute incorrectly.

Dual-Mode operation

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating system Code & user defined code.

For this we need 2 separate modes of operation.

1. User mode and
2. Kernel mode (also called supervisor mode, system mode, privileged mode)



Transition from user to kernel mode diagram

A bit called mode bit is added to the hardware of Computer to indicate the current mode: Kernel (0 bit) or User (1 bit).

- When the computer system is executing, on behalf of a user application, the system is in user mode (0).
- However when a user application requests a service from Operating System (via a System Call) it must transition from user to kernel mode to fulfil the request.

Operating System Services

1. User interface
 - a) Command line interface (CLI)
 - b) Graphics user interface
2. Communication
3. Error detection
4. Resource management
 - a) Process Management
 - b) Memory Management
 - c) Input/Output management
 - d) File - System management
5. Accounting
6. Protection & Security'

Operating System Structure

Till now we have seen what Operating System looks like on the outside (Programmer's interface). Now we will see the internal structure of the Operating System.

1. Monolithic Structure

The entire operating system runs as a single program in the kernel mode. All components are intertwined, with no clear separation between services.

Examples: UNIX, MS-DOS.

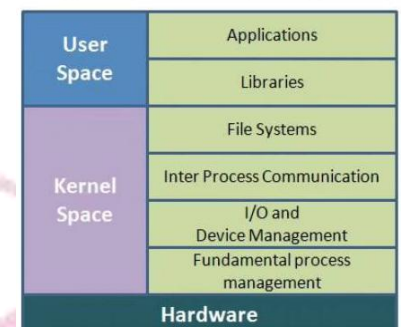
Advantages:

High performance due to minimal overhead.
Simplicity in design.

Disadvantages:

Difficult to maintain or update.
Debugging is complex due to tightly coupled components.

Monolithic Kernel



2. Layered Structure

The operating system is divided into layers, with each layer built on top of the lower ones. Each layer only interacts with its immediate lower layer.

Example: THE OS (an early implementation).

Advantages:

- Modular and easy to manage.
- Easier debugging and maintenance.

Disadvantages:

- Layered communication may introduce performance overhead.
- Designing the system with proper layering can be complex.

Layers	Function
Layer5	Users
Layer4	User programs
Layer3	I/O management
Layer2	Operator-process communication
Layer1	Memory management
Layer0	Hardware

3. Microkernel Structure

The microkernel design moves most of the services to user space, keeping only essential functions like communication and basic scheduling in the kernel space.

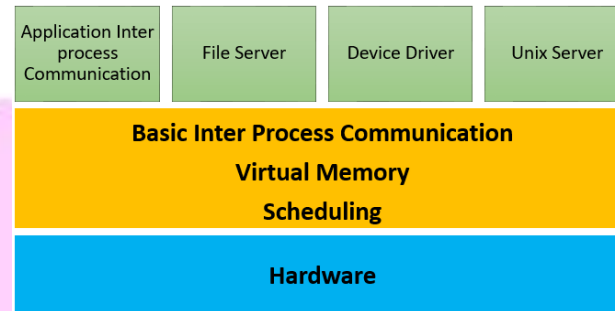
Examples: QNX, Minix.

Advantages:

- Improved reliability and security.
- Easier to add or modify features without affecting the kernel.

Disadvantages:

- Potential performance overhead due to message passing between user space and kernel space.
- Complex to design and implement.



4. Modular Structure

The system uses modules that can be loaded or unloaded as needed, offering flexibility. It combines the advantages of layered and monolithic structures.

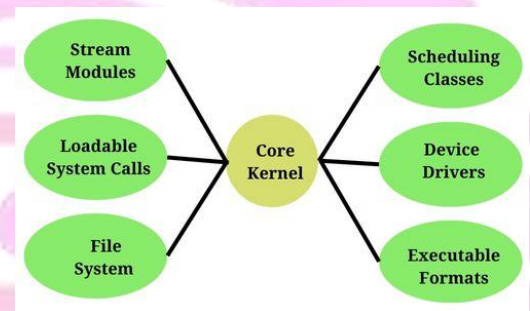
Examples: Linux kernel with loadable modules.

Advantages:

- Highly flexible and easier to update.
- Offers better performance compared to microkernels.

Disadvantages:

- Can still suffer from complexity in managing dependencies.



5. Hybrid Structure

A combination of multiple structures, using the best features of each. It often integrates microkernel features into a monolithic kernel.

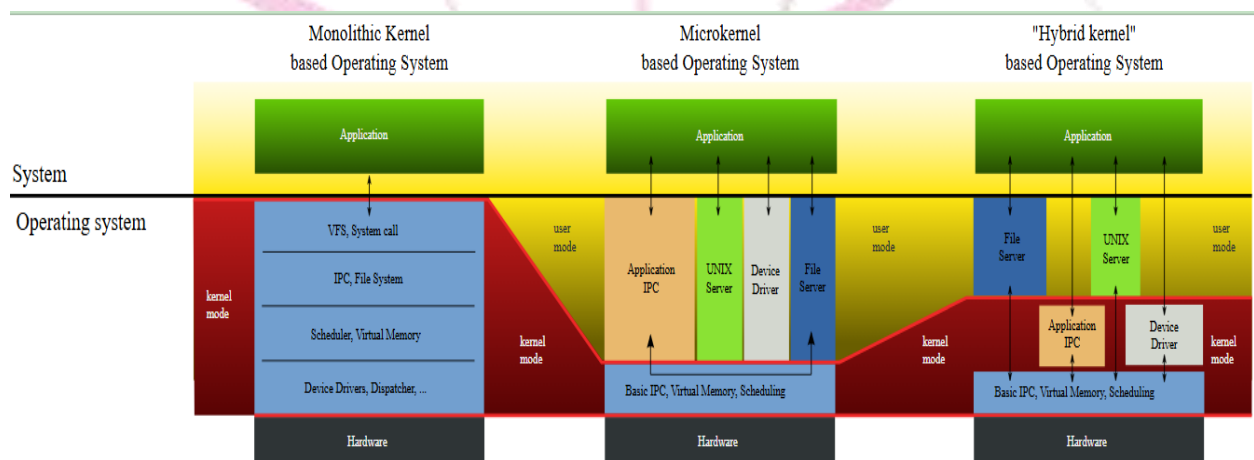
Examples: Windows NT, macOS.4

Advantages:

- Balances performance and modularity.
- Easier to manage and more secure.

Disadvantages:

- Complex architecture and design.
- Harder to maintain than a pure microkernel.



Types of operating System**1. Batch Operating System**

Jobs are collected, grouped, and executed in batches without user interaction. The OS loads and runs jobs one by one, automatically moving to the next job after completing the current one.

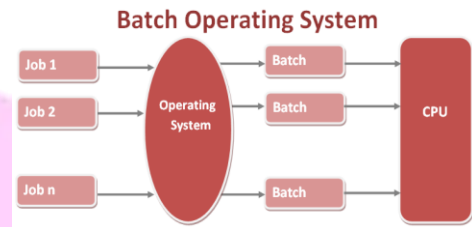
Examples: Early IBM mainframes.

Advantages:

- Efficient for large volumes of repetitive tasks.
- Reduces idle time for the CPU.

Disadvantages:

- No real-time user interaction.
- Debugging is difficult due to delayed output.

**2. Multiprogramming Operating System**

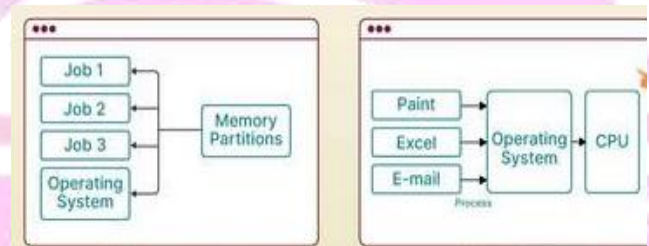
A multiprogramming operating system allows multiple programs to reside in the computer's memory simultaneously and ensures efficient utilization of the CPU by keeping it busy with another task when one program is waiting for input/output (I/O).

Advantages

- Increased CPU Utilization
- Improved Throughput
- Reduced Waiting Time

Disadvantages

- Complex Design
- Context switching creates Overhead

**3. Time-Sharing Operating System**

Multiple users share system resources simultaneously, each user getting a small time slice. The OS switches rapidly between user tasks, giving the impression of parallel execution.

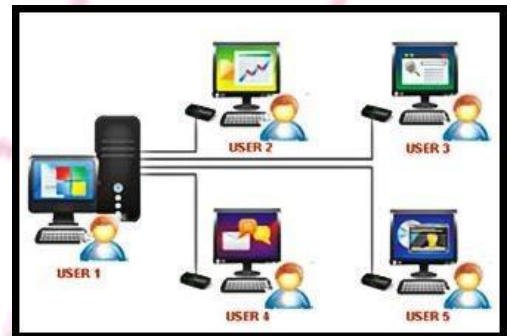
Examples: UNIX.

Advantages:

- Better CPU utilization.
- Interactive computing for multiple users.

Disadvantages:

- Overhead due to context switching.

**4. Distributed Operating System**

Manages a group of independent computers and makes them appear as a single coherent system to users. Resources are shared across multiple interconnected systems, often over a network.

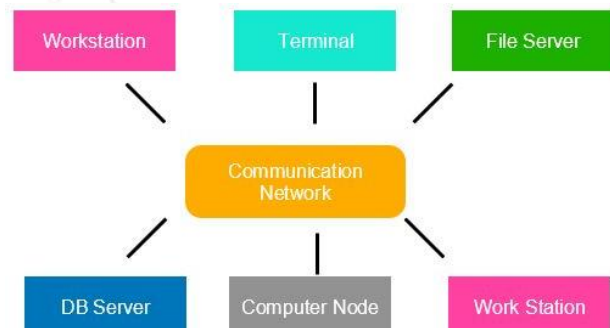
Examples: Locus, VAX/VMS.

Advantages:

- Resource sharing and increased reliability.
- Load balancing to optimize performance.

Disadvantages:

- Complex design and maintenance.



5. Network Operating System (NOS)

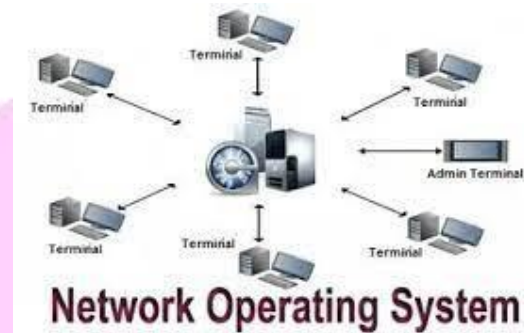
Provides networking capabilities and manages data, users, groups, security, and applications over a network. Each computer runs its own OS but has network features like remote login, file sharing, and printer access. Examples: Microsoft Windows Server, Novell NetWare.

Advantages:

- Centralized resource management.
- Easy access and collaboration over a network.

Disadvantages:

- High cost of setting up and maintaining the network.
- Dependency on a central server can lead to issues if it fails.



6. Real-Time Operating System (RTOS)

Provides immediate processing and response to input, critical for systems requiring timely and predictable behavior.

Types:

Hard Real-Time: Guarantees task completion within a strict deadline (e.g., medical systems).

Soft Real-Time: Prioritizes tasks but allows some delays (e.g., multimedia systems).

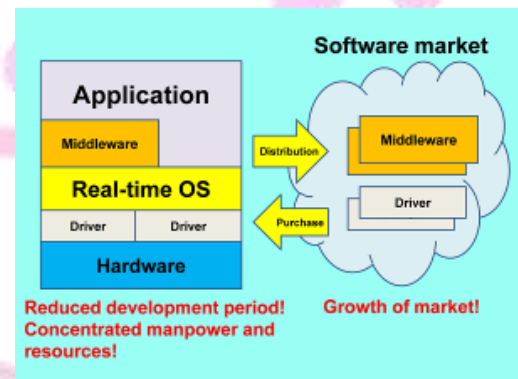
Examples: VxWorks, RTLinux.

Advantages:

- High reliability and predictability.
- Suitable for embedded systems and critical applications.

Disadvantages:

- Limited multitasking capability.
- Complex design and expensive development.



7. Embedded Operating System

Designed for embedded systems, which are part of larger devices. Optimized for minimal resources and specific functions, often running on microcontrollers.

Examples: FreeRTOS, TinyOS.

Advantages:

- Efficient use of system resources.
- Fast and reliable for dedicated tasks.

Disadvantages:

- Difficult to upgrade or modify.
- Limited functionality compared to general-purpose OS.



8. Mobile Operating System

Developed for handheld devices like smartphones and tablets, focusing on efficient battery usage and touch interfaces. Manages hardware resources while supporting apps and user interactions.

Examples: Android, iOS.

Advantages:

- Optimized for mobile hardware and touchscreen use.
- Built-in app stores and software ecosystems.

Disadvantages:

- Limited multitasking compared to desktop OS.



9. Multiprocessing Operating System

Supports the execution of multiple processes simultaneously using multiple CPUs. The OS distributes tasks across different processors to improve speed and efficiency.

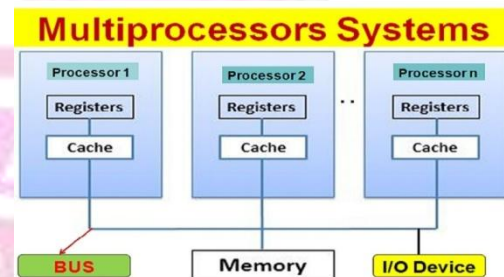
Examples: UNIX, Windows 10.

Advantages:

- Enhanced performance and speed.
- Better fault tolerance.

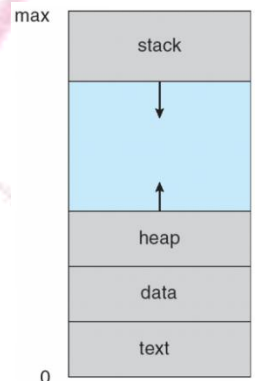
Disadvantages:

- Expensive hardware requirements.
- Complex system design and programming.



Process

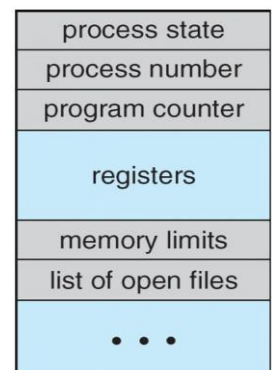
- A process is a program in execution. A process is more than a program code, which is some time known as the text section.
- It also includes the current activity as represented by the value of the "program Counter" and the Content of the processor's registers.
- A process generally also includes the process "stack", which contains temporary data (such as function parameters, return addresses & local variables), and a data section which contains a global variables.
- A process may also include a heap which is memory that is dynamically allocated during process runtime.
- A program is a passive entity, such as a file containing a list of instructions stored on disk (often called executable file) whereas process is an active entity with a Program Counter specifying the next instruction to execute & a set of associated resources.



Process Control Block

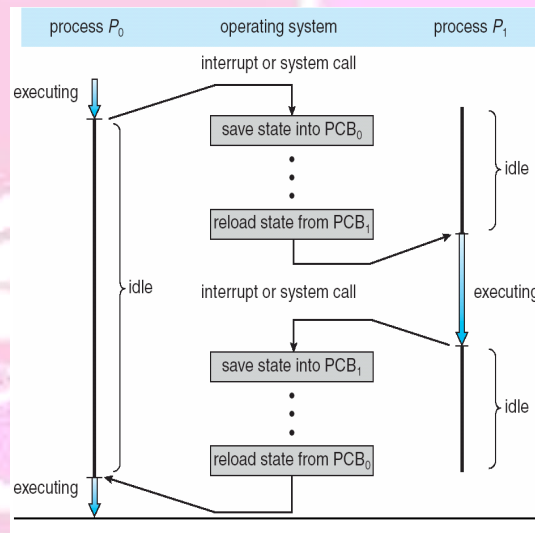
Each process is represented in the OS by a PCB also known as task control block, It contains many pieces of information associated with a specific process, including the following:-

- **Process state:** - The state may be new, ready, running, waiting, halted & so on.
- **Program Counter:** - The Counter indicates the address of the next instruction to be executed for this process.
- **CPU registers:** - The registers vary in number & type, depending on the computer architecture. They include accumulators, index registers, stack pointers & general purpose registers plus any condition Code information.
- Along with the Program Counter, this state information must be saved when an interrupt occurs,



to allow the process to be continued.

- **CPU scheduling information:** This information includes a process priority, pointers to scheduling queues & any other scheduling parameters.
- **Memory management information:** This information may include such information as the value of the base & limit registers, the page tables or the segment tables, depending on the memory system used by the OS.
- **Accounting information:** - This information includes the amount of CPU & real time used, time limits, account numbers job or process numbers & so on.
- **I/O states information:-** This information includes the list of I/O devices allocated to the process, a list of open files. & so on.



Thread

- A thread is a basic unit of CPU utilization, it comprises a thread ID, a Program Counter, a register set and a stack.
- It shares with other threads belonging to the same process its code section, data section & other resources, such as open files & signals.
- If a process has multiple threads of control, it can perform more than one task at a time.
- Most OS kernels are now multithreaded, several threads operate in the kernel, & each thread performs a specific task such as managing devices or interrupt handling.

Types of Threads :- There are 2 types of threads.

1. User level thread
2. Kernel level thread.

Operation on Processes

OS needs some way to create processes.

In UNIX the "ps" Command can be used to list the running processes.

In windows, the "task manager" can be used.

Process Creation:- Process creation is achieved through the fork() system call in Unix Operating System.

- The newly created process is called the child process and the process that initiated it (or the process when execution is started) is called the parent process.
- After the fork() system call, we have two processes - parent and child processes.
- The fork() system call returns either of the three values:-
 - Negative value to indicate an error, i.e., unsuccessful in creating the child process.
 - Returns a zero for child process.
 - Returns a positive value for the parent process. This value is the process ID of the newly created child process.
- There are also 2 possibilities in terms of the address space of the new process.
 1. The child process is a duplicate of the parent process (it has the same program & data as the parent).
 2. The child process has a program loaded into it.

Subscribe Infeepedia youtube channel for computer science competitive exams

Download Infeepedia app and call or wapp on 8004391758

Wait():-

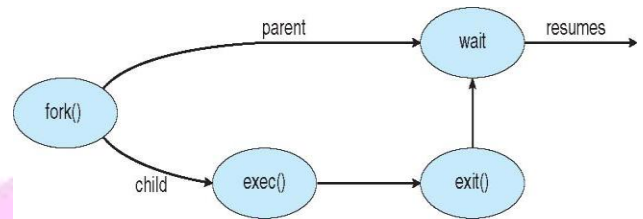
- If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.

Process Termination:- A process terminates when it finishes executing its final statement and ask the OS to delete it by using the "exit()" system call.

In windows to terminate the process. "TerminateProcess()" is used.

Process Can terminate by one of the following reasons:-

1. Normal exit (Voluntary)
2. Error exit (voluntary)
3. Fatal error (Involuntary)
4. Killed by another process (Involuntary)

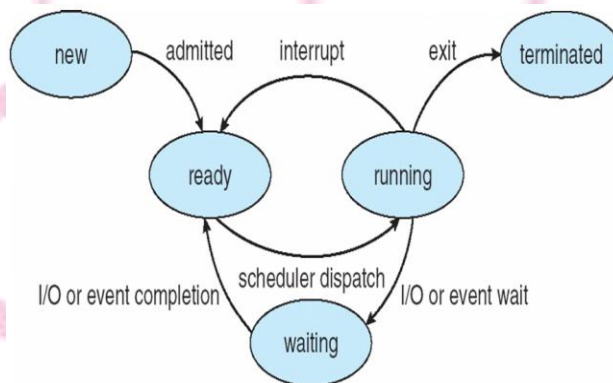


Process creation using fork () system diagram

Cascading termination :- Some systems do not allow a child to exist if its parent has terminated. In such system if a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon is referred as cascading termination is normally initiated by the OS.

Process State

- As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.
- Each process may be in one of the following state
 - **New:** The process is being created
 - **Running:** Instructions are being executed
 - **Waiting:** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
 - **Ready:** The process is waiting to be assigned to a processor.
 - **Terminated:** The process has finished execution.



Process state transition diagram

Some important terms

Arrival time:- Arrival time is the point of time in millisecond at which a process arrives at the ready queue to begin the execution.

Arrival time can be calculated as the difference of the completion time & the Turn around time of the process.

Arrival Time (A.T.) = Completion Time (CT) - Turn Around Time (TAT)

Burst Time (BT):- Burst time refers to the time required in milliseconds by a process for its execution.

The Burst time takes into consideration the CPU time of a process.

It is mainly called as the execution time or running time of the process.

CPU Burst:- when the CPU executes the process

I/O Burst:- when the Process is in waiting / blocked state to perform I/O operation.

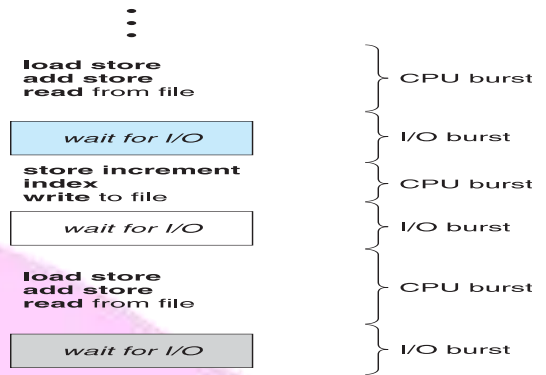
The I/O time is not taken into consideration.

The Process makes a transition from the running state to the completion state during this time frame.

Burst time can be calculated as the difference of the completion time of the process & the waiting time:

Burst time (B.T) = Completion Time (C.T) - Waiting Time (W.T)

Burst time (BT) = Turnaround time (TAT) - waiting time (W.T)



Response Time:- It is the time spent between the ready state and getting the CPU for the first time.

Response Time = Time at which the process gets the CPU for first time - Arrival time.

Waiting Time:- waiting time is the total time spent by the process in the ready State waiting for CPU.

The Waiting time is the total time taken by the process in the ready state.

Waiting Time. = Turnaround time - Burst time.

Exit time:- Exit time is a time when a process completes its execution & exit from the system.

Turnaround time:- Turnaround time is the total amount of time spent by the process from coming in the ready state for the first time to its completion.

Different CPU scheduling algorithms produce different turnaround time for the same set of processes, because the waiting time differs when we change the CPU scheduling algorithm.

Turnaround time = Burst time + waiting time.

Turnaround time = Exit time - Arrival time.

Throughput:- It is a way to find the efficiency of a CPU. It can be defined as the number of processes executed by the CPU in a given amount of time.

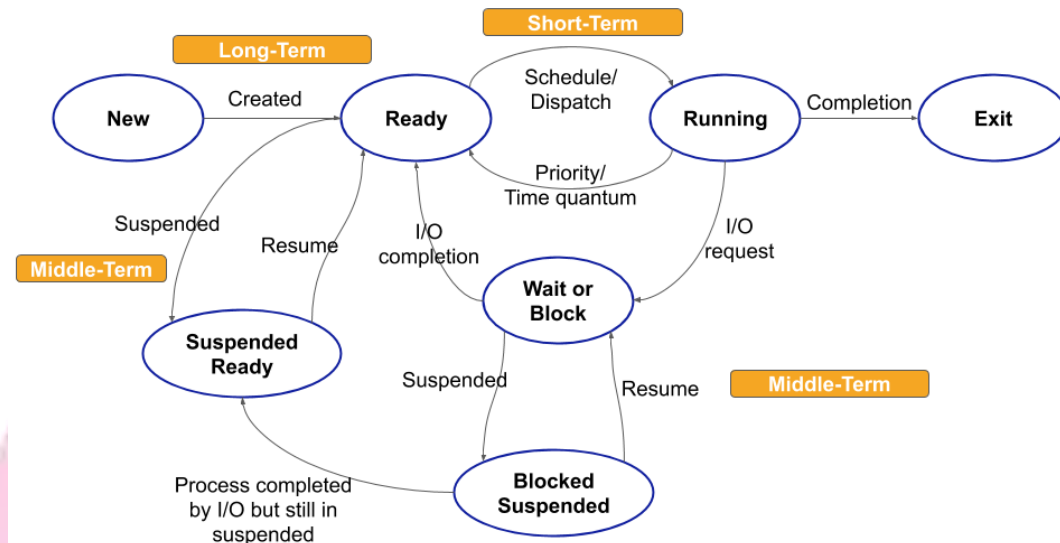
Dispatch latency:- The dispatcher should be as fast as possible since it is invoked during every process switch. The time it takes for the dispatcher to stop one process & start another running is known as the dispatch latency.

Process Scheduling

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- The objective time sharing is to switch the CPU among processes so frequently that users can interact with each program, while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.
- As we discuss about single processor system, there will never be more than one running process.
- If there are more processes the rest will have to wait (Job pool) until the CPU is free & Can be rescheduled.

Scheduling Queues

- Job Queue:-** As processes enter the system, they are put into a job queue (Job pooling), which consists of all processes in the system.
- Ready Queue:-** The processes that are residing in main memory and are ready & waiting to execute are kept on a list Called the ready queue.
- Device Queue:-** when a process is allocated the CPU, it executes for a while & after that the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue such as Input queue, Output queue etc.



Schedulers

There are 3 types of schedulers.

Schedulers are special system software which handled process scheduling in various ways.

1. **Long term scheduler or Job scheduler:-** In batch systems more processes are submitted than can be executed immediately.
 - These processes are spooled to a mass storage device (disk), where they are kept for later execution. The job scheduler selects processes from this pool & loads them to execute.
 - It Controls the degree of multiprogramming (the number of processes in memory).
 - It is important that the long term scheduler make a careful selection of both I/O bound process & CPU bound process
 - I/O bound processes are which use much of their time in input & output operation while CPU bound processes are which spend their time on CPU.
 - The Job scheduler increases efficiency by maintaining balance between the two.
2. **Short term or CPU scheduler:-** It is responsible for selecting one process from ready state for scheduling it on the running state.
 - The CPU scheduler is responsible for ensuring that there is no starvation owing to high burst time processes. It is responsible for loading the process selected by short term scheduler on CPU (ready to running state).
 - Dispatcher is responsible for:
 - Context switching
 - Switching to user mode
 - Jumping to the proper location in program the newly loaded program.
3. **Medium term scheduler:-** Some Operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling.
 - The key idea behind a medium term scheduler is that sometimes it can be advantageous to remove processes from memory (& from active contention for the CPU) and thus reduce the degree of multiprogramming.
 - Later the process can be reintroduced into memory & its execution can be continued where it left off. This scheme is called swapping. The process is swapped out & is later swapped in by the medium term scheduler.
 - It is responsible for suspending & resuming the process.
 - Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.
 - CPU scheduling is the basis of multiprogramming Operating System. By switching the CPU among processes, the OS can make the computer more productive.

CPU scheduler

- Whenever the CPU becomes idle, the OS must select one of the processes in the ready queue to be executed. The selection process is carried out by the short term scheduler (or CPU scheduler).
- The scheduler selects a process from the processes in memory that are ready to execute & allocated the CPU to that process.
- Note that the ready queue is not necessarily a FIFO queue; it can be FIFO queue, a priority queue, a tree, or simply an unordered linked list. The records in the ready queue are generally PCBs of the processes.
- CPU scheduling decisions may take place under the following 4 circumstances:-
 - a) When a process switches from the running state to the waiting state.
 - b) When a process switches from the running state to the ready state.
 - c) When a process switches from the waiting state to the ready state.
 - d) When a process terminates.

Note: For situations 1 & 4, there is no choice in terms of scheduling. A new process must be selected for execution when scheduling takes place only under circumstances 1 & 4. We say that the scheduling scheme is non-preemptive or cooperative otherwise it is preemptive for situation 2 & 3.

- It is desirable to maximize CPU utilization and throughput & to minimize turnaround time, waiting time, & response time.
- In most cases, we optimize the average measure. However under some circumstances, it is desirable to optimize the minimum or maximum values rather than the average.
- For example: - to guarantee that all users get good services, we may want to minimize the maximum response time.

Scheduling Algorithms

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

There are some following Algorithms: -

- 1) First Come First served scheduling
- 2) Shortest Job First scheduling (Non Preemptive version)
- 3) Priority Scheduling
- 4) Round Robin Scheduling
- 5) Shortest Remaining Time First scheduling (Preemptive version of SJF)
- 6) Longest Job First Scheduling (non-preemptive)
- 7) Longest Remaining time First scheduling (Preemptive)
- 8) Multilevel Queue scheduling
- 9) Multilevel Feedback Queue Scheduling.

1. First Come First served scheduling

- It is a non-preemptive scheduling & used in batch system with this scheme, the process that requests the CPU first is allocated the CPU first.
- FCFS is implemented & easily managed with a FIFO queue.
- When the process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

Disadvantages:-

- The average waiting time under the FCFS facility is quite long.
- The average waiting time under an FCFS policy depends on the CPU burst time and also arrival time.

Example1: consider the following set of processes that arrive at time 0 with the length of the CPU burst time given in ms

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1, P_2, P_3

The Gantt Chart for the schedule is:



Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

Average waiting time: $(0 + 24 + 27)/3 = 17$

2. Shortest job First Scheduling

- It is a non-preemptive & batch system scheduling. It is an optimal algorithm.
- It is an algorithm in which the process having the smallest execution time is chosen for the next execution.
- It significantly reduces the average waiting time for other processes awaiting execution..
- It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
- Preemptive version of SJF is shortest remaining time first (SRTF).
- SJF scheduling is associated with each job as a unit of time to complete.
- SJF is frequently used for long term scheduling.
- It reduces average waiting time over FCFS algorithm.

Disadvantage:-

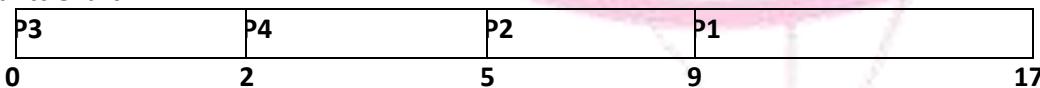
- Job Completion time must be known earlier but it is hard to predict.
- SJF Cannot be implemented for CPU scheduling for the short term. It is because there is no specific method to predict the length of the upcoming CPU burst.

Example:

Case 1: here process arrival time and burst time is given :-

Process	Burst Time	Arrival Time
P1	8	0
P2	4	0
P3	2	0
P4	3	0

Gantt Chart



Average waiting time = $(9+5+0+2) / 4 = 4$ ms

In this case the arrival time is 0 of all the processes so simply compare the burst time of the processes and execute in ascending order (shortest job first & longest job in the last).

Response Time	Waiting Time	Turnaround time
9	9	17
5	5	9
0	0	2
2	2	5

3. Shortest remaining time first scheduling

- It is a pre-emptive version of shortest job first scheduling.
- In this algorithm the processor is allocated to the job closest to completion.
- In the SRTF the short processes are handled very quickly.

Disadvantages:-

- CPU has to switch Content multiple times as if the shortest job occurs from the running one, So the overhead in context switching.
- CPU spends a lot time in switching the process which effect the utilization of CPU.
- SRTF may cause starvation as shorter process may keep coming and a long CPU burst process never gets CPU.

Example: SRTF algorithm (Preemptive)

here process arrival time and burst time is given :-

Process	Burst Time	Arrival Time
P1	8	2
P2	4	6
P3	5	0
P4	2	3
P5	3	1

Gantt Chart:

P3	P5	P4	P3	P2	P1	
0	1	4	6	10	14	22

Response Time	Waiting Time	Turnaround time
13	13	21
0	0	4
0	9	15
1	1	3
0	0	3

4. Round-Robin Scheduling

- It is designed for time-sharing system or interactive system.
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- Each process is assigned a time interval, called its time quantum, during which it is allowed to run.
- If the process is still running at the end of quantum, the CPU is preempted & given to another process.
- It is simple, easy to implement & starvation free as all processes get fair share of CPU.
- **Disadvantages:** - The drawback of Round robin scheduling is more overhead of context switching.
- Sometimes larger waiting time & response time so low throughput (for less time quantum).
- The performance of the RR also depends heavily on the size of time quantum.
- A time quantum is generally from 10 to 100 milliseconds in length.
- The ready queue is treated as a circular queue.
- If there are n processes in the ready queue & the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units.
- Each process must wait no longer than $(n-1)q$ time units until its next time quantum.
- **Example:** with 5 processes & a time quantum of 20 milliseconds, each process will get up to 20 milliseconds after every 100 milliseconds

Example1:- here process burst time is given with time quantum=2 ms:-

Process	Burst Time
P1	4
P2	5
P3	3
P4	6

Gantt Chart

P1	P2	P3	P4	P1	P2	P3	P4	P2	P4	
0	2	4	6	8	10	12	13	15	16	18

Response Time	Waiting Time	Turnaround time
0	6	10
2	11	16
4	10	13
6	12	18

5. Priority Scheduling

- It is a batch System and an interactive system scheduling.
- A priority is associated with each process, & the CPU is allocated to the process with the highest priority.
- Equal priority processes are scheduled in FCFS order.
- Priority scheduling can be either preemptive or non preemptive.
- when the priority is calculated during the execution of system then it is called dynamic priority otherwise if the priority is assigned before the execution is called priority scheduling.

Disadvantage :-

- A major problem with priority Scheduling algorithms is indefinite blocking & starvation.
- Priority scheduling can leave some low priority processes waiting indefinitely.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for long time.
- **Example of aging:-**If priorities range from 127 (low) to (0) high. We could increase the priority of a waiting process by 1 every 15 minutes.

6. Multilevel Queue Scheduling

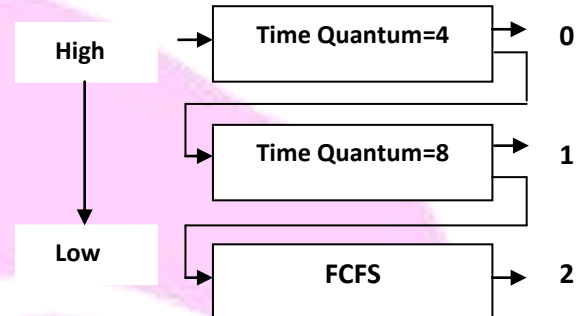
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues.
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm for Ex: Separate queues might be used for foreground & background processes.
- The foreground queue might be scheduled by the R.R. also while the background queue is scheduled by an FCFS algo.

Advantages: The processes are permanently assigned to the queue, so it has advantage of low scheduling overhead.

Disadvantages:- Some processes may starve for CPU if some higher priority queues are never becoming empty. It is inflexible in nature.

7. Multilevel Feedback Queue Scheduling

- It allows a process to move between queues. The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower priority queue. This scheme leaves I/O bound & interactive processes in the higher priority queues.
- In addition, a process that waits too long in a lower priority queue may be moved to a higher priority queue.
- This form of aging prevents starvation.
- Here queue 0 & queue 1 follow round robin & queue 3 follows FCFS.
- When Process start executing it first enter in queue queue0.
- In queue0 it executes 4 units if it complete its execution then the priority of process will not change but if it has to execute more, then its priority reduces & shifted to queue1. If it will not complete its execution in queue1 also then again priority reduces & shifted to queue queue2.



Interprocess Communication

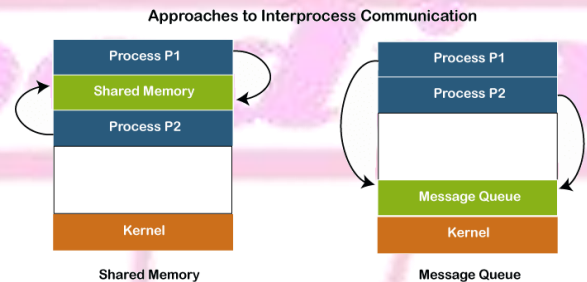
- Processes executing concurrently in the OS may be either independent or Cooperative processes.
- Independent Processes:** Processes that do not share resources.
- Cooperating Processes:** Processes that share data or resources.
- Example:** Producer-Consumer problem is best example of Cooperative process.

Reasons for providing an environment that allows process Cooperation:-

- Information sharing(data, resource etc)
- Computation Speedup
- Modularity
- Convenience

There are 2 fundamental models of interprocess Communication.

- Message Passing**
- Shared memory.**



Difference between message passing & shared memory

SNo.	Message Passing	Shared memory
1	Message passing is useful for exchanging smaller amount of data because no conflicts need be avoided	Shared memory is useful for larger amount of data because conflicts need be avoided.
2	Easy to implement for inter Computer Communication.	Complex to implement for inter computer Communication comparatively.
3	Slower than shared memory because message passing system typically implemented using System Calls and thus require the more time consuming task of kernel intervention.	Faster than message passing because no kernel intervention. system calls required only to establish shared memory regions once shared memory is establish all accesses are treated as routine memory accesses.
4	synchronization is achieved by message passing	Requires synchronization mechanisms (e.g., semaphores, mutexes).

Shared memory Systems

- IPC using shared memory requires Communicating processes to establish a region of shared memory.
- A shared memory region resides in the address space of the process creating the shared memory segment.
- They can exchange information by reading & writing data in the shared areas.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.
- Example of shared memory system:- Producer-Consumer problem.
- A producer process produces information that is consumed by a Consumer process.

Example1: - A compiler may produce assembly Code, which is consumed by an assembler. The assembler, in turn, may produce object modules, which is consumed by the loader.

Example2: - The producer- consumer problem also provides useful metaphor for the client-server paradigm. We can think of a server as a producer & a client as a consumer, web server produces HTML files and images which are consumed by the client web browser requesting the resource.

2 types of buffer can be used:-

1. **Unbounded buffer:-** It places no practical limit on the size of the buffer. The Consumer may have to wait for new items, but the producer can always produce new items.
2. **Bounded buffer:-** it is assumed a fixed size buffer. In this Case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Message Passing System

- Message passing provides a mechanism to allow processes to Communicate & to synchronize their actions without sharing the same address space & is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.
- **Example:** A chat program used on the www could be designed so that chat participants communicate with one another by exchanging messages.
- A message passing facility provides at least 2 operations:
Send (message)
receive (message)

Here are several methods for logically implementing a link & the send()/ receive () operations:-

1. Direct or indirect Communication
2. Synchronous or asynchronous Communication
3. Automatic on explicit buffering

Process Synchronization

- Cooperating processes can either directly share a logical address space (i.e. both code & data) or be allowed to share data only through files or messages.
- Concurrent access to shared data may result in data inconsistency. So we need different mechanisms to ensure that orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.
- **Race Condition:** Race Condition is an undesirable situation that occurs when a device or system attempts to perform 2 or more operations at the same time.

Example:- Two processes P1 & P2

Process P1	Process P2
1. read (A)	1. read (A)
2. A = A +2	2. A = A +4
3. write(A)	3. write(A)

Let us assume $A=10$

Now for process P1 $A=10$, $A=10+2=12$ And $A=12$ is output	Now for process P2 $A=10$, $A=10+4=16$ And $A=16$ is output
---	---

Here $A=10$, $P1(A)=12$ and $P2(A)=16$

If the CPU pre-empt after the 2nd statement of the process P1 and executes process P2 then the result will be incorrect. $A=10$, $P1=12$ and $P2=14$.

Here the process is not synchronized and the result depends on the sequences of instruction execution.

To resolve this race condition we should take care of synchronization and Coordination amongst Cooperating processes

The Critical section Problem

- A Critical section is the portion of a program or process in which the process may be updating or changing Common variables, updating a table, writing a file or manipulating any data and so on.
- In the multiprogramming & real time environment where multiple processes are dependent to each other & share some data, the shared data is featured as a critical section & no 2 processes can modify or access critical section simultaneously.
- A system Consisting of n processes $[P0, P1, P2, \dots, P_{n-1}]$, having a shared code segment, can execute simultaneously because the Shared Code is executed in Critical section.
- The important feature of the system is that when one process is executing in its critical section no other process is allowed to execute in its critical section
- The critical section problem is to design protocol that the processes can use to Cooperate.
- Each process must request permission to enter its critical section.
- The section of Code implementing the request is the entry section.
- The critical section may be followed by an exit section.
- The remaining code is the remainder section.

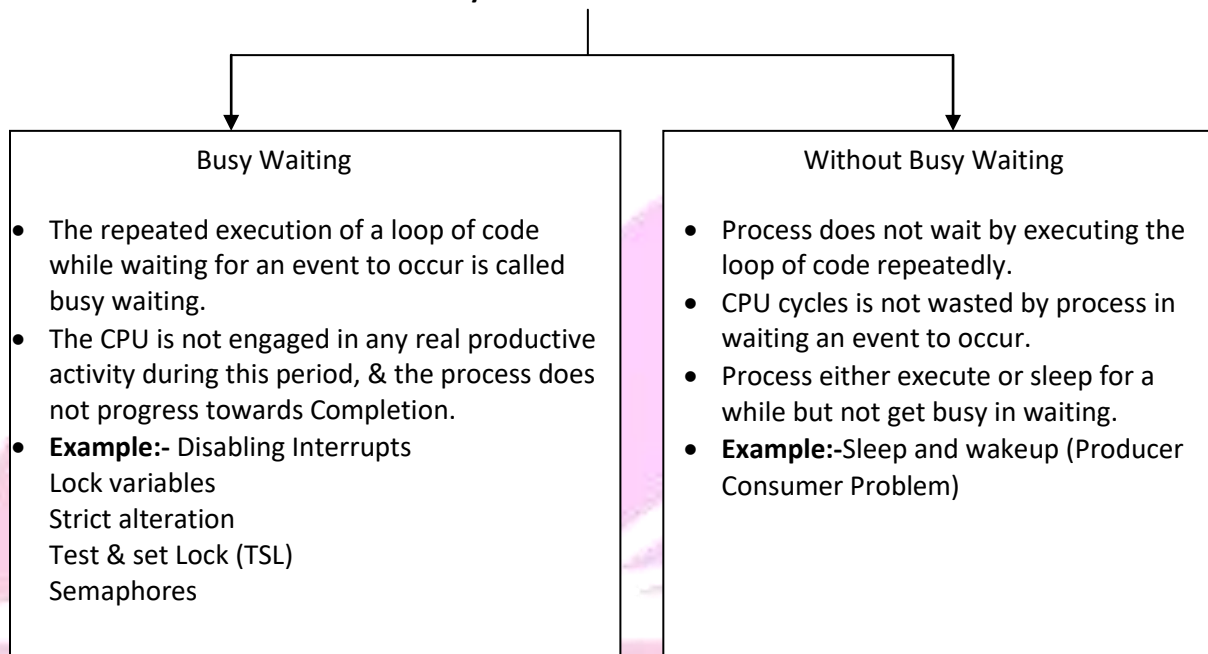
General structure of a typical process P_i

```
do
{
    entry section
    critical section
    exit section
    remainder section
}
while (true);
```

- The solution to the critical section problem must satisfy the following 4 requirements:
1. **Mutual exclusion:-** If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
 2. **Progress:-** If no process is executing in its critical section & some processes wish to enter their critical sections then only those processes that are not executing in their remainder Section can participate in deciding which will enter its critical section next. & this selection cannot be postponed indefinitely.
 3. **Bounded waiting:-** There exists a bound, or limit, on the a number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
 4. **Portability:-** Here we assume that each process is executing at a nonzero speed. However, we can make no assumption concerning the relative speed of the n processes & hardware.
- A process could run in user or Kernel mode.

Example: Peterson's solution

Process Synchronization mechanism

Semaphores

- Semaphores is an integer variables that are used to solve the critical section problem by using 2 atomic operations wait (P) and signal (v) that are used for process synchronization
- It is a synchronization tool.
- The wait () operation is also known as "Down()" & "P()".
- The signal () operation is also termed as "up()" & "v()".
- The definition of wait & signal is as follows:

wait (S) { while (s<=0); S--; }	Signal(s) { S++; }
--	------------------------------------

- All modifications to the integer value of the semaphore in the wait() and signal () operations must be executed indivisibly. i.e. when one process modifies the semaphore, no other process can simultaneously modify that same semaphore value.

Types of semaphores:- There are 2 main type of semaphores.

1. Counting Semaphores.
2. Binary Semaphores

1. Counting Semaphores:-

- The value of a Counting semaphore can range over an unrestricted domain ($-\infty$ to $+\infty$)
- Counting semaphores can be used to control access to a given resource consisting of a finite number of instances
- The semaphore is initialized to the number of resources available.
- Each process that wishes to use a resource performs a wait() operation on the semaphore (thereby decrementing the Count).
- when a process releases a resource, it performs a signal() operation (incrementing the Count)
- when the count for the semaphore goes to 0. All resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

2. Binary Semaphore -

- The value of a binary semaphore can range only between 0 and 1.
- Binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion.
- Binary semaphores to deal with critical section problem for multiple processes.
- The n processes share a semaphore, mutex, initialized to 1.

```

do
{
wait (mutex),      → entry section.
// critical section
Signal (mutex);    → exit section.
// remainder section.
}
while (true);

```

- The main disadvantage of the semaphore definition is that it requires busy waiting.
- while a process is in its critical section any other process that tries to enter its critical section must loop continuously in the entry code.
- This continual looping is clearly a problem in a real multiprogramming system where a single CPU shared among many processes.
- This type of semaphore is also called a spinlock because the process "spins" while waiting for the lock.
- Spinlocks do have advantages in that no Context switch is required when a process must wait on a lock and a Context switch may take considerable time. Thus when locks are expected to be hold for short times, spinlocks are useful.

Difference between Counting Semaphore and Binary Semaphore

SNo.	Counting Semaphore	Binary Semaphore
1	No mutual Exclusion	Mutual Exclusion
2	Any Integer value	value only 0 and 1
3	More than 1 slot	only one slot
4	Provide a set of processes	It has a mutual exclusion mechanism.

Problems of Synchronization solved by using Semaphore

We have a large class of concurrency Control problems. These problems are used to test the proposed synchronizations mechanisms.

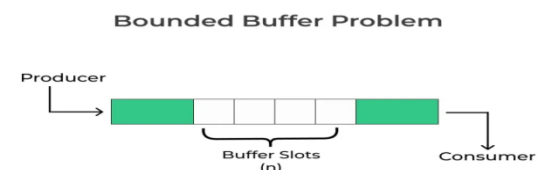
Here we will discuss the solution of the synchronization problems by using semaphore.

1. The Bounded - Buffer Problem:- It is also known as Producer - Consumer problem.**Problems in bounded buffer:-**

- The producer must not insert data when the buffer is full.
- The Consumer must not remove data when the buffer is empty.
- The producer & Consumer should not insert & remove data simultaneously.

Solution in bounded buffer problems:-**Here we are using 3 semaphores:-**

- m (mutex):- A binary semaphore which is used to acquire and release the lock.
- Empty:- A Counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty (number of empty slots.).
- Full:- A Counting semaphore whose initial value is 0.



2. Readers - writers Problem:-

There are 4 cases in reader-writer problem:

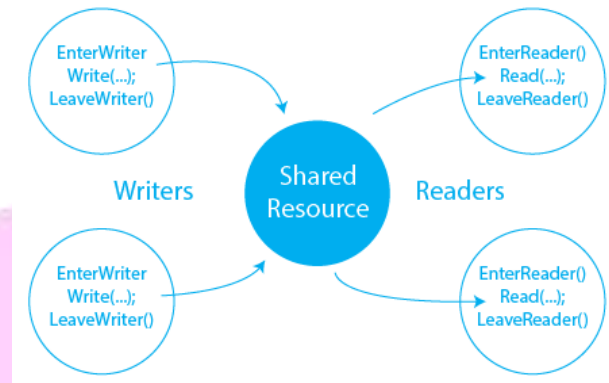
Case1: Reader-writer

Case2: writer – Reader

Case3: writer-writer

Case4: Reader - Reader

Problems occur in case1 to case3 and in case4 there is no problem.

**3. The Dining Philosophers Problem:-**

- In 1965 Dijkstra found this problem.
- Consider 5 philosophers (Processes) who spend their lives thinking & eating.
- Philosophers are seated around a circular table.
- The bowl of rice is placed in the centre of table and 5 chopsticks are laid on the table one chopstick beside every Philosopher.
- When a Philosopher thinks, she does not interact with her Colleagues.
- When a Philosopher gets hungry she tries to pick up the 2 chopsticks that are closest to her (left & right).
- A philosopher may pick up only one chopstick at a time.
- She cannot pick up a chopstick that is already in the hand of a neighbour.
- While eating Philosopher does not release any of the 2 chopstick, when she has finished eating, she puts down both of her chopsticks and starts thinking again.



One solution using semaphore:-

- a. Chopstick [5]:- It is an array of 5 binary semaphore items & each item's initial value is 1.
- b. A philosopher tries to grab a chopstick by executing a wait () operation on that semaphore.
- c. Philosopher releases her chopstick by executing signal operation on the appropriate semaphores.

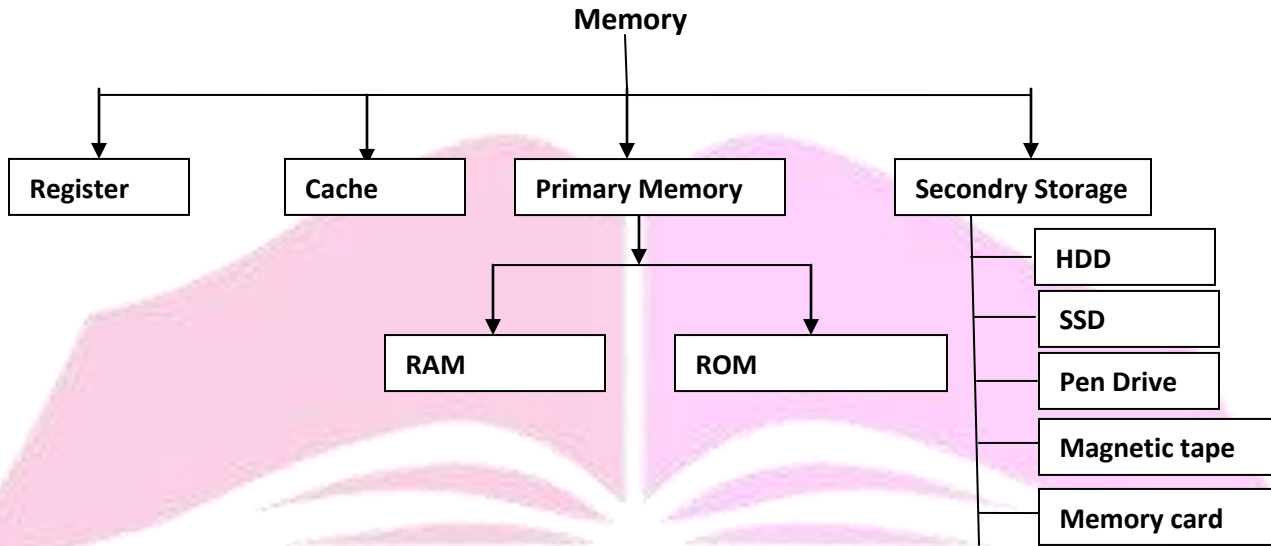
Some remedies to overcome to the deadlock problem.

1. Allow at most 4 philosophers to be sitting simultaneously at the table (not effective because it changes the scenario).
2. As we know every Philosopher grabs her left Chopstick first of them right chopstick one by one. here we will change the sequence of grabbed chopsticks for 5th (last) Philosopher. 5th Philosopher will grab right chopstick first of then left chopstick.
3. Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this, she must pick them up in critical section).
4. Use an asymmetric solution, that is, an odd philosopher picks up first her left chopstick & then her right chopstick where as an even philosopher picks up her right chopstick and then her left chopstick.

Note: To ensure a deadlock free solution to the dining-Philosopher Problem we use monitors. But still the starvation remains.

Monitors:-

- Monitors are a synchronization construct that were created to overcome the problems caused by semaphores such as timing errors.
- Monitors are abstract data types and contain shared data variables and procedures. The shared data variables cannot be directly accessed by a process and procedures are required to allow a single process to access the shared data variables at a time.
- Only one process can be active in a monitor at a time. Other processes that need to access the shared variables in a monitor have to line up in a queue and are only provided access when the previous process releases the shared variables.

Memory ManagementMemory Management

- Memory management is used to achieve a degree of multiprogramming and proper utilization of memory.
- Memory management is a process of controlling and coordinating computer memory.
- In memory management operating system assigns memory blocks to various running programs to optimize the overall performance of the system.

Use of memory management

- Operating system manages memory to allocate and de-allocate space before and after process execution.
- It is used to keep track of used memory space by processes.
- We can minimize fragmentation issues by memory management.
- It is used to proper utilization of main memory.
- It is used to maintain data integrity while executing of process.

Some Important termsLogical address Space:

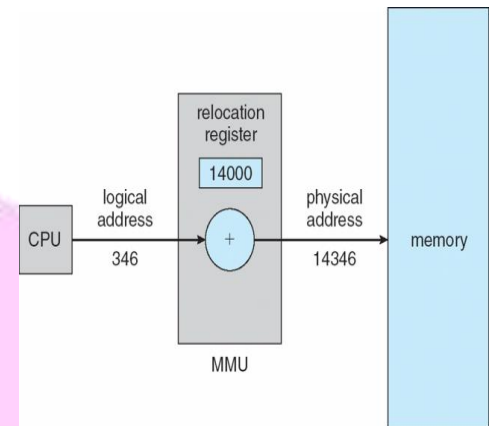
- Logical Address is generated by CPU while a program is running.
- The logical address is also known as Virtual Address.
- This address is used as a reference to access the physical memory location by CPU.
- The set of all logical addresses that are generated by any program is referred to as Logical Address Space
- The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

Physical Address Space

- An address in which a process is stored in the memory (i. e. loaded into the memory-address register of the memory) is commonly referred to as a Physical address.
- A physical address cannot be accessed by the user directly but the user can calculate the physical address with the help of a Logical address.
- The user never directly deals with the physical address but can access by its corresponding logical address.
- The set of all physical addresses corresponding to the logical addresses in a Logical address space is called Physical Address Space.

Memory management unit

- It is a hardware device that does the run-time mapping from the virtual address to the physical address.
- It is located within the Central Processing Unit.
- The Memory Mapping unit mainly converts the logical addresses into the physical addresses.

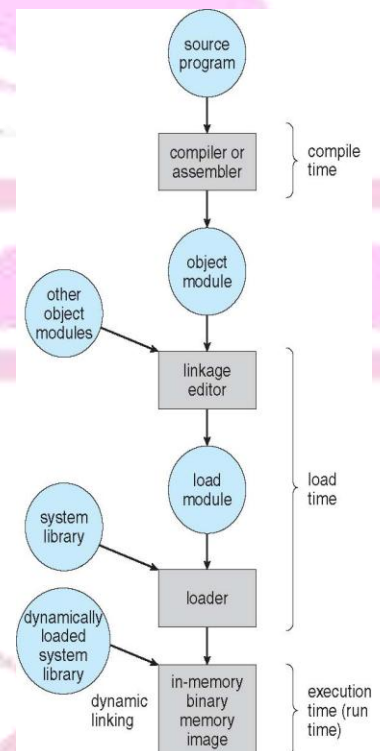
Address Binding

- Address binding is a mapping from one address space to another.

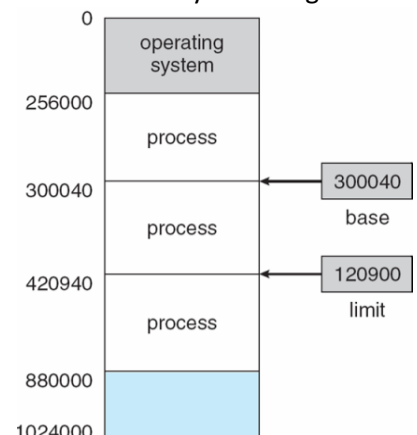
Types of Address Binding:-

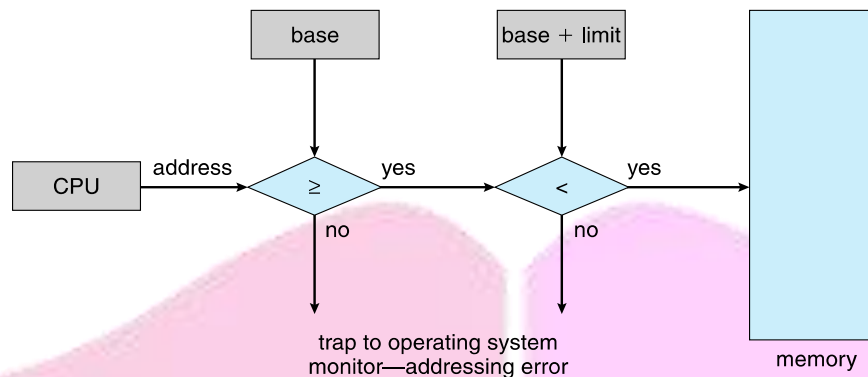
The binding of instructions and data to memory addresses can be done by three types:

1. Compile time
2. Load time
3. Execution time

Base and limit registers

- We first need to make sure that each process has a separate memory space. To do this, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
- We can provide this protection by using two registers, usually a base and a limit register.
- The base register holds the smallest legal physical memory address;
- The limit register specifies the size of the range.
- For example, if the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).





A base and limit registers define the logical address space

Static and Dynamic Loading

To execute a program it is necessary for the entire program and all data of a process to be in physical memory. To load a process into the main memory is done by a loader.

There are two different types of loading :-

- **Static loading:-** In static loading the entire program is loaded into a fixed address. It requires more memory space to load entire program at one time.
- **Dynamic loading:-** To utilize the memory, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. Main advantage of dynamic loading is that unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

Fragmentation

- When the processes are loaded and removed from the memory they create free space or hole in the memory and these small blocks cannot be allocated to new upcoming processes and results in inefficient use of memory. This is called memory fragmentation.
- Storage is fragmented into a large number of small holes.

Fragmentation is of 2 types:-

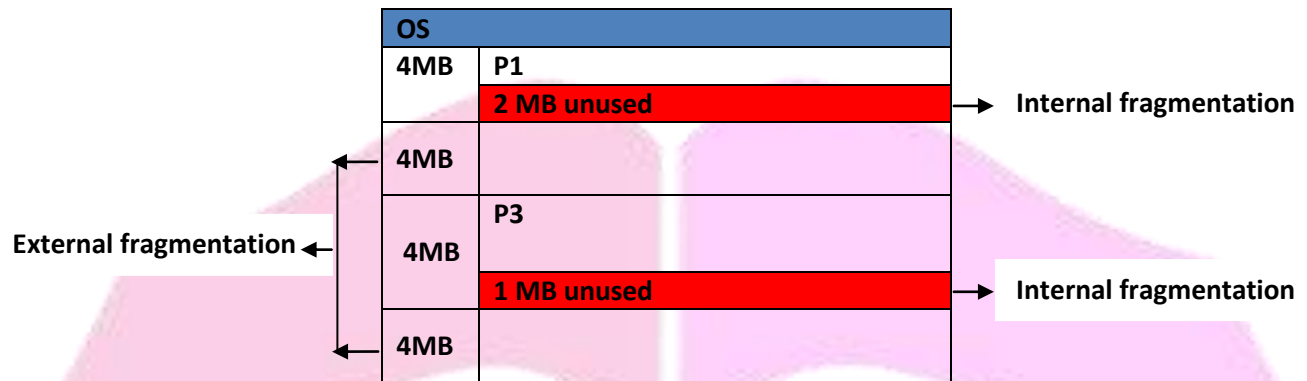
1. Internal Fragmentation:-

- The process is allocated a memory block of size more than the size of that process. Due to this some part of the memory is left unused and this cause internal fragmentation.
- In Fixed Partitioning internal fragmentation can be reduced by assigning the smallest partition, which is still good enough to carry the entire process.
- This problem can be removed if we use dynamic partitioning for allocating space to the process. In dynamic partitioning, the process is allocated only that much amount of space which is required by the process. So, there is no internal fragmentation.

2. External Fragmentation:-

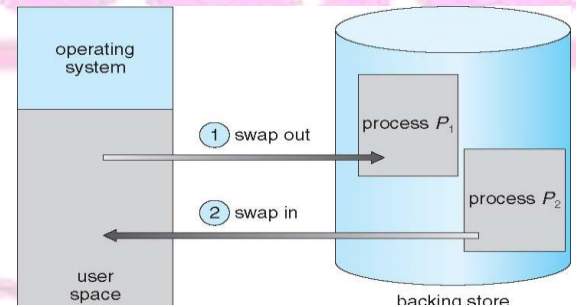
- In this fragmentation, although we have total space available that is needed by a process still we are not able to put that process in the memory because that space is not contiguous. This is called external fragmentation.
- This problem is occurring because we are allocating memory continuously to the processes. To remove external fragmentation we can use paging & segmentation (non-contiguous memory allocation techniques).
- Another way to remove external fragmentation is compaction. When dynamic partitioning is used for memory allocation then external fragmentation can be reduced by merging all the free memory together in one large block. This technique is also called defragmentation.

Example:- Suppose fixed partitioning (i.e. the memory blocks are of fixed sizes) is used for memory allocation in RAM. Processes P1, P2, and P3 have sizes 2MB, 3MB, and 6MB respectively. Some part of this RAM is occupied by the Operating System (OS). Suppose processes are allotted space as given in the figure then:



Swapping

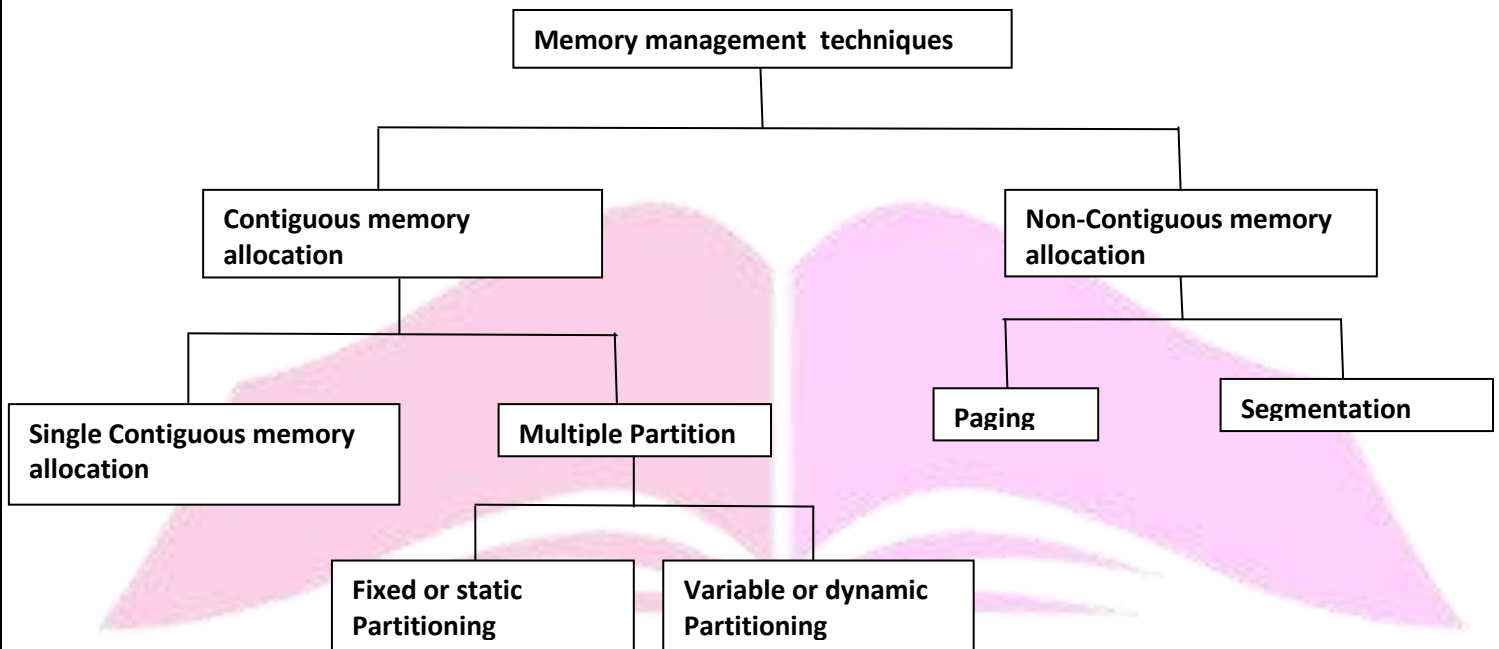
- Swapping is mechanisms in which a process can be swapped temporarily out of main memory (or move) to backing store and make that memory available to other processes. At some later time, the system swaps back the process from the backing store to main memory.
- A swapping allows more processes to be run and can be fit into memory at one time.
- Swapping requires a backing store.
- The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.
- Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.
- roll-out, roll in:-** A variant of swapping policy is used for priority-based scheduling algorithms. if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process. This variant of swapping policy is known as roll-out, roll in.



Swapping of two processes using a disk as a backing store

Memory Management Techniques

- The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.
- We can place the operating system in low memory and user processes in high memory.
- Operating systems is responsible for allocating and managing a computer's main memory.
- Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.
- There are two Memory Management Techniques:
 - Contiguous Memory allocation**
 - Non-Contiguous Memory allocation**



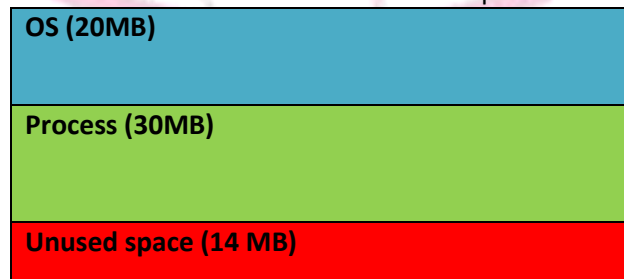
1. Contiguous memory allocation

- If a process is stored in a contiguous fashion in the physical memory blocks then it is called a contiguous memory allocation.
- It is easy to implement and access time of data is also efficient due to sequential data is stored.
- A process can be randomly accessed in contiguous memory allocation.
- A major disadvantage of contiguous memory allocation is the memory gets fragmented.

It can be of 2 types:-

a. Single Contiguous memory allocation:

- There is only one block of memory and there is no degree of multiprogramming concept.
- One process can reside in memory at one time.
- The user process is stored in a single section of the contiguous memory block and it is given to that process according to its requirement.
- The major issue with Single Contiguous memory allocation is internal fragmentation. If the process size is less than the memory size then the remaining space will be unused.
- Example:- Suppose we have a memory size of 64 MB in which OS space is of 20 MB and user space is of 44MB. And now suppose we have a process of 30 MB. Here we left with 14 MB unused space as internal fragmentation.



b. Multiple partition

We can divide multiple partitions in 2 types

1. Fixed or static partition

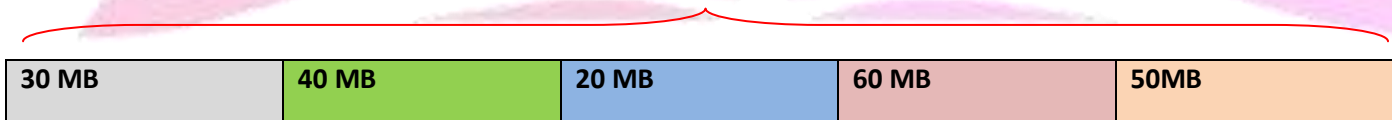
- In this partitioning, the memory is divided into multiple fixed blocks and the degree of multiprogramming depends on the number of memory partition (block).
- The number of partitions (non-overlapping) in main memory is fixed but the size of each partition may or may not be the same. As it is a contiguous allocation, hence no spanning is allowed.

Disadvantages:-

- The major problem is internal and external fragmentation of memory.
- Process size should not be greater than memory block.
- Limitation on degree of multiprogramming.
- Example:- Suppose we have a memory of 100MB. We can partition this memory as:-

Example:- Suppose we have a memory of 100MB. We can partition this memory as:-

200MB



Fixed partitioning with different size blocks

200MB

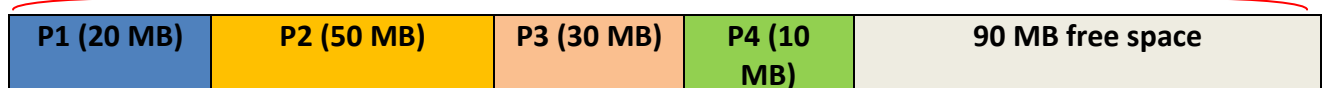


Fixed partitioning with same size blocks

2. Variable or dynamic partition

- To overcome the drawback of fixed partitioning i.e. internal fragmentation we use variable partitioning. In this partitioning scheme, the size of the partition is not declared initially.
- In variable partitioning allocation is done dynamically when the process arrives, the memory is allocated to the process according to its size.
- As partition is created according to the need of the process so in this partition scheme there is no internal fragmentation.
- Degree of multiprogramming is dynamic.
- There is no limitation on the size of the process as the memory space is allocated dynamic
- The major problem of the variable partitioning is external fragmentation.
- It is quite difficult to implement as the allocation of memory at run-time rather than during the system configuration. It is difficult for operating system to manage memory as allocation and de-allocation is done frequently.
- **Example:- suppose we have a memory space of 100 MB and there are 4 Processes P1, P2, P3 and P4 of sizes 20 MB, 50 MB, 30 MB and 10 MB respectively. So we will allocate memory dynamically as:-**

200MB



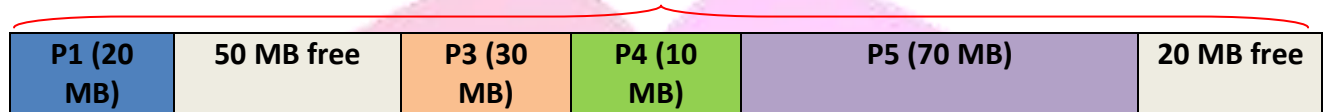
Variable partitioning

Subscribe Infeepedia youtube channel for computer science competitive exams

Download Infeepedia app and call or wapp on 8004391758

- Now suppose process P2 (50 MB) terminates and 2 more processes P5 and P6 sizes 70 MB and 60 MB respectively arrives.
- Here we have enough space free but we cannot allocate memory to both the processes due to external fragmentation. We can allocate memory to process P5 but unable to allocate memory to process P6 as we have two holes (50 MB and 20 MB) of memory available in non contiguous form.

200MB



Variable partitioning with external fragmentation

Algorithm for contiguous memory Partition allocation:- There are 3 algorithms are used to allocate memory to a process.

- First fit:-** Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
- Best fit:-** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- Worst fit:-** Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach in variable size partitioning.

2. Non contiguous Memory allocation

- In the non-contiguous memory allocation, the available free memory space is not at one place, the memory is divided into frames and the frames are may or may not be available in continuous manner.
- In this technique, we solve the major problem of external fragmentation.
- Different parts of the same process can be stored at different places in the memory.
- We can allocate memory in 2 types in non contiguous memory allocation.

1. Paging

2. Segmentation

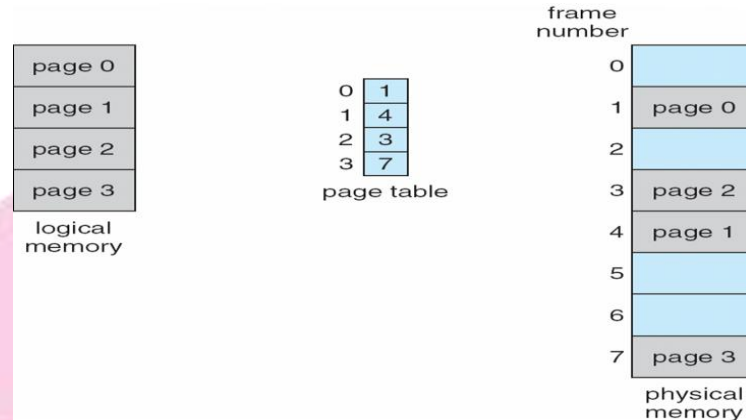
1. Paging

- Paging is a fixed size partitioning scheme.
- Paging avoids external fragmentation space and the need for compaction but still it suffers from internal fragmentation.
- When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store). The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as pages and are defined by the hardware.
- The partitions of main memory are called as frames and are defined by the hardware.
- Paging itself is a form of dynamic relocation. Every logical address is bound by the paging hardware to some physical address. Using paging is similar to using a table of base (or relocation) registers, one for each frame of memory.

Disadvantages of Paging

- It suffers from Internal fragmentation
- Page table requires additional memory.
- Multi-level paging may lead to memory reference overhead.

Example:- Logical memory is a place where the process address is stored. In this example page table contains the frame number of the corresponding page number.

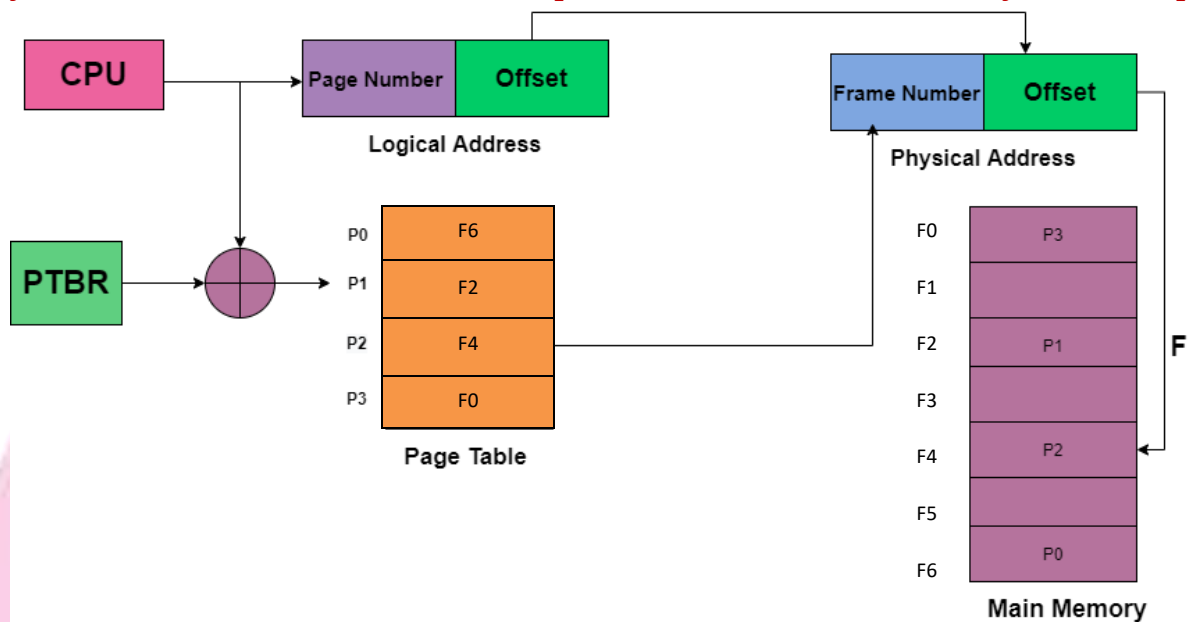


Translation of Logical Address into Physical Address

- As we know CPU always generates a logical address and we need physical address to access the main memory. However we map logical address to physical address using page table.
- The logical address generated by CPU always consists of two parts:
 - Page Number(p)**
 - Page Offset (d)**
- Page Number is used to specify the specific page of the process from which the CPU wants to read the data and it is also used as an index to the page table.
- Page offset is mainly used to specify the specific word on the page that the CPU wants to read it is also called displacement.
- If the size of logical address space is 2^m and page size is 2^n addressing units then the high order $m-n$ bits of logical address designates the page number and the n low-order bits designate the page offset.
- The logical address is as follows:



- The physical address consists of two parts:**
 - Frame Number(f)**
 - Page offset(d)**
- The frame number is combined with the page offset and forms the required physical address.
- Frame number is used to indicate the specific frame where the required page is stored.
- Page Offset indicates the specific word that has to be read from that page.
- The Page size (like the frame size) is defined with the help of hardware. It is important to note here that the size of the page is typically the power of 2 that varies between 512 bytes and 16 MB per page and it mainly depends on the architecture of the computer.



This diagram indicates the translation of the Logical address into the Physical address.

Page Table:-

- Page table is a data structure.
- It maps the page number referenced by the CPU to the frame number where that page is stored.
- The Page table mainly contains the base address of each page in the Physical memory (frame number). The base address is then combined with the page offset in order to define the physical memory address.
- Page table is stored in the main memory.
- Number of entries in a page table = Number of pages in which the process is divided.

Page Table Entry:-

- A page table entry contains several information about the page.
- The information contained in the page table entry varies from operating system to operating system.
- The most important information in a page table entry is frame number.

In general, each entry of a page table contains the following information-

Frame Number	Valid/Invalid bit	Protection	Reference	caching	Modify(Dirty) bit
Mandatory field		optional field			

1. Frame Number:-

- Frame number specifies the frame where the page is stored in the main memory.
- The number of bits in frame number depends on the number of frames in the main memory.
- Frame bit is also known as address translation bit.

2. Valid / Invalid Bit:

- This bit is also sometimes called as present / absent bit.
- This bit specifies whether that page is present in the main memory or not.
- If the page is not present in the main memory, then this bit is set to 0 otherwise set to 1.

NOTE

- If the required page is not present in the main memory, then it is called as Page Fault.
- A page fault requires page initialization.
- The required page has to be initialized (fetched) from the secondary memory and brought into the main memory.

Subscribe Infeepedia youtube channel for computer science competitive exams

Download Infeepedia app and call or wapp on 8004391758

3. Protection Bit:-

- This bit is also sometimes called as “Read / Write bit”.
- This bit is concerned with the page protection.
- It specifies the permission to perform read and write operation on the page.
- If only read operation is allowed to be performed and no writing is allowed, then this bit is set to 0.
- If both read and write operations are allowed to be performed, then this bit is set to 1.

4. Reference Bit:-

- Reference bit specifies whether that page has been referenced in the last clock cycle or not.
- If the page has been referenced recently, then this bit is set to 1 otherwise set to 0.

NOTE

- Reference bit is useful for page replacement policy.
- A page that has not been referenced recently is considered a good candidate for page replacement in LRU page replacement policy.

5. Caching Enabled / Disabled:-

- This bit enables or disables the caching of page.
- Whenever freshness in the data is required, then caching is disabled using this bit.
- If caching of the page is disabled, then this bit is set to 1 otherwise set to 0.

6. Modify bit:

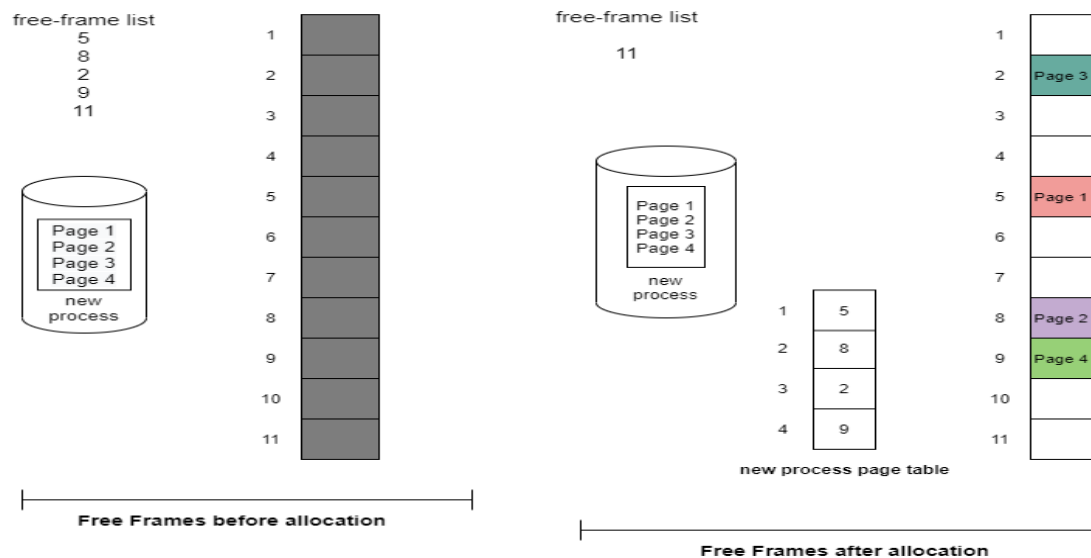
- This bit is also sometimes called as “Dirty bit”.
- This bit specifies whether that page has been modified or not.
- If the page has been modified, then this bit is set to 1 otherwise set to 0.

NOTE

- In case the page is modified,
- Before replacing the modified page with some other page, it has to be written back in the secondary memory to avoid losing the data.
- Dirty bit helps to avoid unnecessary writes.
- This is because if the page is not modified, then it can be directly replaced by another page without any need of writing it back to the disk.

Frame table:-

- The frame table is a data structure that keeps the information of which frames are allocated or which frames are available and many more things.
- This table mainly has one entry for each physical page frame.



Some important formula related to paging

These formulas are useful for solving the numerical problems based on paging.

Formulas for Main Memory:-

1. Frame size = Page size
2. Physical Address Space = Size of main memory
3. Size of main memory = Total number of frames x Page size
4. If number of frames in main memory = 2^n , then number of bits in frame number = n bits
5. If Page size = 2^n Bytes, then number of bits in page offset = n bits
6. If size of main memory = 2^n Bytes, then number of bits in physical address = n bits

Formulas for Process (logical memory):-

1. Virtual (logical) Address Space = Size of process
2. Number of pages in the process = Process size / Page size
3. If process size = 2^n Bytes, then number of bits in virtual address space = n bits

Formulas for Page Table:-

1. Size of page table = Number of entries in page table x Page table entry size
2. Number of entries in pages table = Number of pages in the process
3. Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

Note:-

- In general, if the given address consists of 'n' bits, then using 'n' bits, 2^n locations are possible.

Then, size of memory = $2^n \times$ Size of one location.

Question and answer related to paging

Ques1:- Calculate the number of bits required in the address for memory having size of 32 MB also find frame size when the number of frames is 2^{10} . Assume the memory is Byte addressable.

Solution:- given that:

$$\text{Size of memory} = 32 \text{ MB} = 2^5 * 2^{20} \text{ B} = 2^{25} \text{ B}$$

So the physical address = 25 bits

F	D
10 bit	15 bit

25 bit

$$\text{Number of frames} = 2^{10}$$

$$\text{Physical memory address} = \text{frame size} * \text{number of frames}$$

$$2^{25} \text{ B} = \text{frame size} * 2^{10} \text{ B}$$

$$\text{Frame size} = 2^{15} \text{ B} = 32 \text{ KB}$$

Ques2:- Consider a system with byte-addressable memory, 32 bit logical addresses and 28 bit physical address, 4 kilobyte page size and page table entries of 4 bytes each. Find the size of the page table in the system and also find the number of frames in the physical memory.

Solution:- Given-

Number of bits in logical address = 32 bits

Page size = 4KB

Page table entry size = 4 Bytes

P	d
20 bit	12 bit

LA 32 bit

Process Size = logical address space

Number of bits in logical address = 32 bits

Process Size = logical address space

Number of bits in logical address = 32 bits

Thus, logical address space = 2^{32} B

Process size = 2^{32} B = 4 GB

Number of pages in the process = Process size / Page size

= 4 GB / 4 KB = $(2^2 * 2^{30})$ B / $(2^2 * 2^{10})$ B = $2^{32} / 2^{12}$

= 2^{20} pages

Number of entries in page table = 2^{20} entries

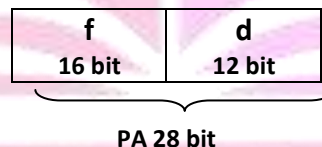
Page table size = Number of entries in page table x Page table entry size = $2^{20} \times 2^2$ Bytes = 4 MB

Physical address = 28 bit

Physical address space (memory size) = 2^{28} B = 256 MB

Page size = frame size = 2^{12} B

Frame numbers = 2^{16}

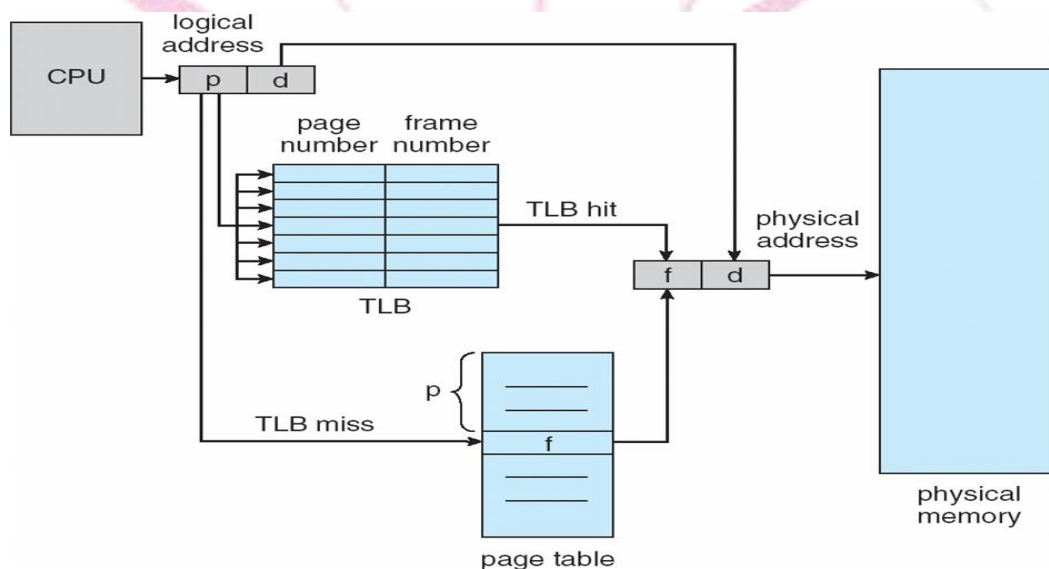


Translation look aside buffer (TLB)

- The major problem of paging with page table is the effective access time due to increased number of memory accesses. However, the page table is also stored in memory. To access data of the page, first, we need to access page table (which is stored in memory) and then access the memory again to access the data.
- TLB is associative, high-speed memory.
- A translation look aside buffer (TLB) is a hardware solution that tries to reduce the effective access time.
- It is a memory cache and the access time of TLB is very less as compared to the main memory.
- Whenever context switching occurs, the entire content of TLB is flushed and deleted; And TLB is then again updated with the currently running process.
- When a new process gets scheduled initially, TLB is empty. So, TLB misses are frequent; But with every access from the page table, TLB is updated and after some time, TLB hits increases and TLB misses reduces.

Disadvantages of TLB:-

- TLB can hold the data of only one process at a time. When context switches occur frequently, the performance of TLB degrades due to low hit ratio.
- As it is a special hardware, it involves additional cost.



Effective Access Time (EAT):-

In a single level paging using TLB, the effective access time is given as:-

$$\text{EAT} = \text{Hit ratio of TLB} * (\text{access time of TLB} + \text{access time of main memory}) + \text{Miss ratio of TLB} * (\text{access time of TLB} + 2 * \text{access time of main memory})$$

2. Segmentation

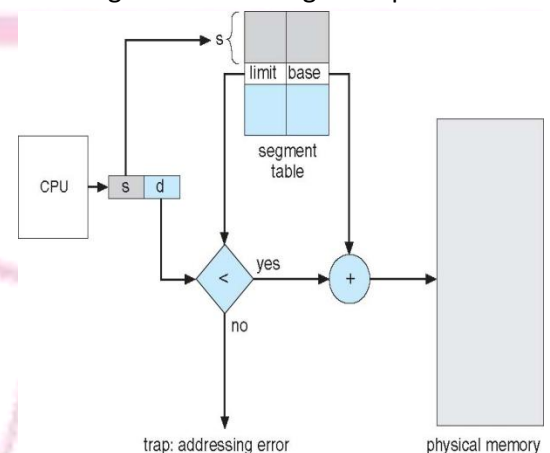
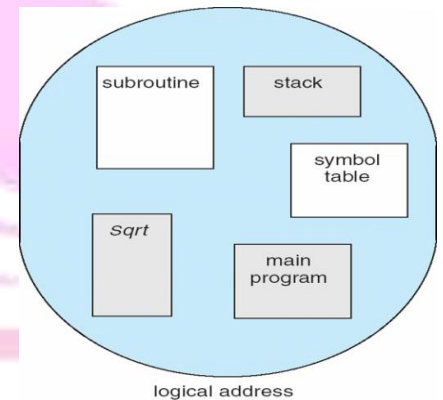
- Segmentation is a non contiguous memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.
- Paging is more close to the Operating system rather than the User. It divides all the processes into the form of pages regardless of the fact that a process can have some relative parts of functions which need to be loaded in the same page.
- Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.
- It is better to have segmentation which divides the process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.
- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program.
- Segment-table base register (STBR) points to the segment table's location in memory
- Segment-table length register (STLR) indicates number of segments used by a program;
- segment number s is legal if $s < \text{STLR}$

Advantages:

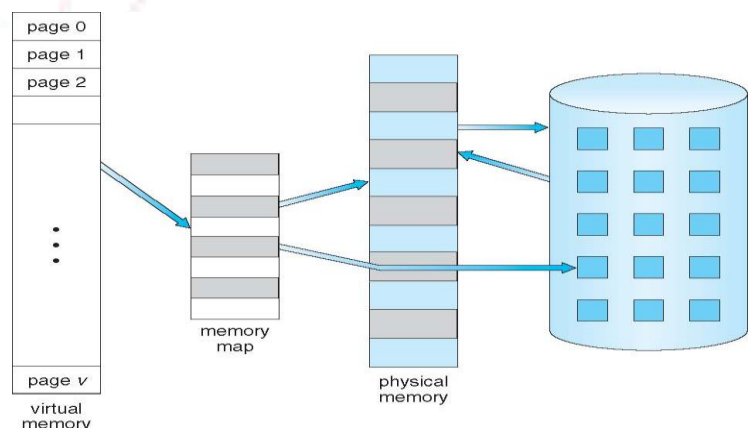
- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantages:

As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

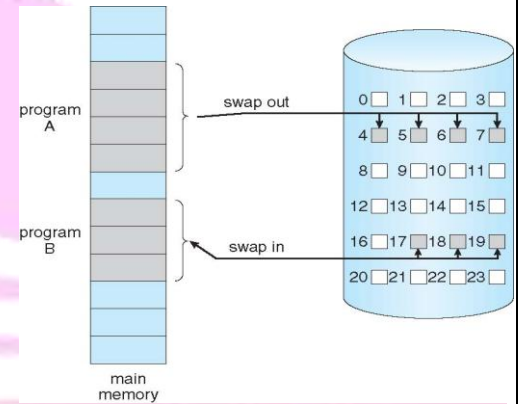
**Virtual Memory**

- Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. If the process size is larger than main (physical) memory, the process is stored in secondary memory and only some needed pages of the process is loaded in the main memory.
- Virtual memory is mostly implemented with demand paging and demand segmentation.
- Virtual memory also allows processes to share files easily and to implement shared memory.
- To execute an instruction it is required to be in physical memory but due to limited size of main memory we upload only those pages of the process which is necessary and being used.

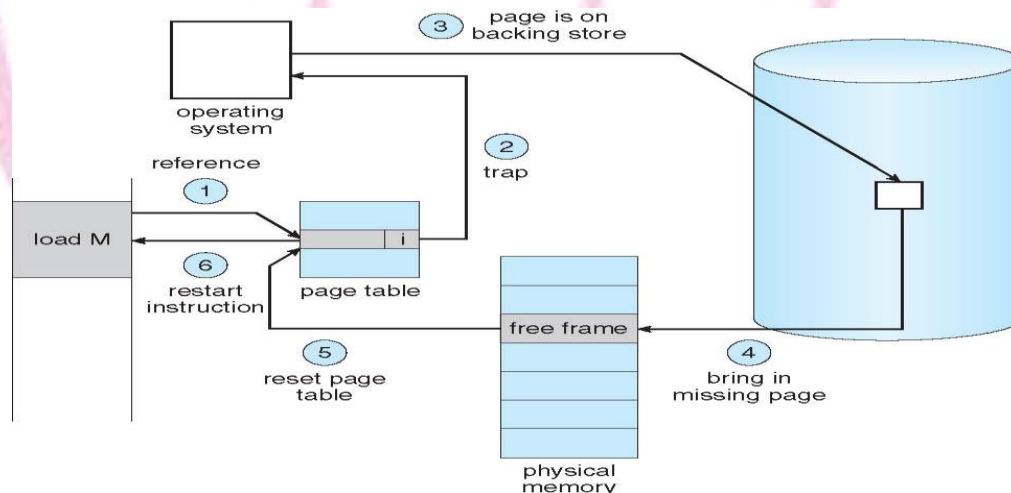


Demand paging

- Demand paging is a memory management scheme in which pages of data are loaded into physical memory only when they are required by a running program.
- Unlike pre-paging, where all necessary pages are loaded in advance, demand paging minimizes memory usage and improves efficiency by loading pages on-demand.
- Rather than swapping the entire process into memory, however, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed.
- We thus use pager, rather than swapper, in connection with demand paging.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process.
- The valid-invalid bit scheme is used.
- When this bit is set to "valid," the associated page is both legal and in memory.
- If the bit is set to "invalid," the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.

Steps in Demand Paging

- The CPU generates a virtual address to access a page.
- The page table is checked to see if the page is in memory.
- If the page is not present, a page fault occurs.
- The OS interrupts the process and determines the location of the page on the disk.
- The required page is fetched from secondary storage (disk) and loaded into physical memory.
- The page table is updated to mark the page as valid (present in memory).
- The process resumes execution.

**Steps in Handling a Page Fault****Advantages of Demand Paging**

1. Efficient Memory Use
2. Faster Process Startup
3. Supports Multiprogramming:

Disadvantages of Demand Paging

1. Page Fault Overhead
2. Disk I/O
3. Thrashing

Pure demand paging: never bring a page into memory until it is required.

Locality of reference:- Locality of reference, also known as the principle of locality, is the tendency of a processor to access the same set of memory locations repetitively over a short period of time. It results in reasonable performance from demand paging.

Performance of Demand Paging:- As long as we have no page faults, the effective access time is equal to the memory access time (denoted as ma). If, however, a page fault occurs, we must first read the relevant page from disk and then access the desired word.

Let p be the probability of a page fault ($0 < p \leq 1$).

if $p = 0$ no page faults

if $p = 1$, every reference is a fault

We would expect p to be close to zero—that is, we would expect to have only a few page faults. The effective access time is then

$$\text{Effective access time} = (1-p) * ma + p * \text{page fault time.}$$

Ques:- The average page-fault service time of 8 milliseconds and a memory access time of 200 nanoseconds, what is the effective access time in microseconds? If one access out of 1,000 causes a page fault.

Solution:- given that memory access time (ma)=200 ns,

Page fault service time= 8 ms = 8000000ns

$P = 1/1000$ (1 page fault out of 1000 access)

Effective access time (EAT) = $(1 - 1/1000) * (200) + 1/1000 * (8 \text{ milliseconds})$

= $(999/1000) \times 200 + 1/1000 \times 8,000,000$

= $199.8 + 8000$

= $8199.8 \text{ ns} = 8199.8 * (1/1000) \text{ microseconds}$

EAT = 8.2 microseconds.

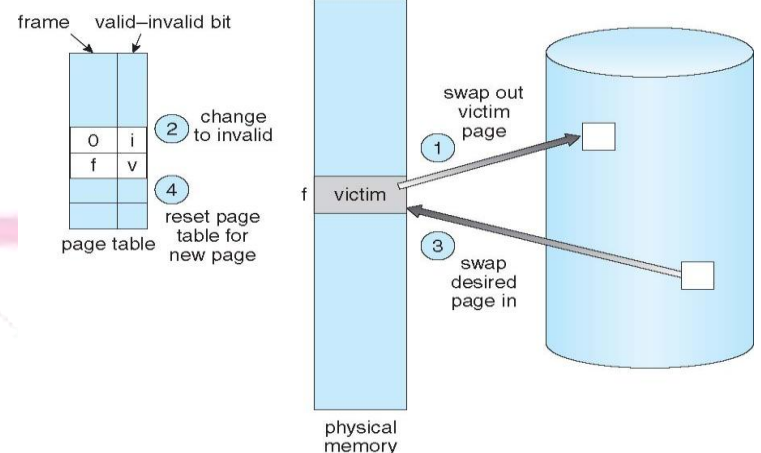
Note: The effective access time is directly proportional to the page-fault rate.

Page Replacement

- The process of replacement is sometimes called swap out or write to disk.
- If no frame is free, we find one that is not currently being used and free it.
- We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory.
- We can now use the freed frame to hold the page for which the process faulted.

We modify the page-fault service routine to include page replacement:

1. Find the location of the desired page on the disk.
2. Find a free frame:
 - a. If there is a free frame, use it.
 - b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
 - c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.



Page Replacement Algorithms

The page replacement algorithm helps to decide which pages must be swapped out from the main memory in order to create space for the referenced page. We use page replacement algorithm to reduce page-fault rate.

Various Page Replacement algorithms used in the Operating system are as follows;

1. FIFO page replacement algorithm
2. Optimal page replacement algorithm
3. Least recently used page replacement algorithm

1. First-In-First-Out (FIFO) Algorithm

For example:- reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5 this algorithm is implemented by memory with 3 frames
3 frames in memory

		3	3	3	2	2	2	2	2	4	4
	2	2	2	1	1	1	1	1	3	3	3
1	1	1	4	4	4	5	5	5	5	5	5
M	M	M	M	M	M	M	H	H	M	M	H

Page fault with 3 frames in memory = 9

Total page fault = 9 (page miss)

Page hit = 3

Page fault ratio = number of page fault / (total page miss + page hit) * 100

Page fault ratio = $(9/12) * 100 = 75\%$

Page hit = $(5/20) * 100 = 25\%$

Belady's Anomaly:- FIFO page-replacement algorithms suffers from Belady's anomaly, the page-fault rate may increase as the number of allocated frames increases.

reference string: 1,2,3,4,1,2,5,1,2,3,4,5 implement using 4 frames

4 frames in memory

			4	4	4	4	4	4	3	3	3
		3	3	3	3	3	3	2	2	2	2
	2	2	2	2	2	2	1	1	1	1	5
1	1	1	1	1	1	5	5	5	5	4	4
M	M	M	M	H	H	M	M	M	M	M	M

Page fault with 4 frame in memory = 10

Here we can see that when the number of frames increases, the number of page fault also increases. Which is called Belady's Anomaly.

2. Optimal page replacement algorithm

- In this algorithm, OS replaces the page that will not be used for the longest period of time in future. It is also called futuristic algorithm.
- This algorithm replaces the page which will not be referred for so long in future.
- This Algorithm does not suffers with Belady's Anomaly.
- **Example:-** let us consider a reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 for a memory with 3 frames.

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2		2		2								7		
	0	0	0		0		0		0								0		
		1	1		3		3		3								1		

page frames

Total page fault = 9 (page miss)

Subscribe Infeepedia youtube channel for computer science competitive exams

Download Infeepedia app and call or wapp on 8004391758

Page hit = 11

Page fault ratio = number of page fault/ (total page miss+ page hit) * 100

Page fault ratio= (9/20) * 100 = 45%

Page hit = (11/20)*100 = 55%

3. Least recently used (LRU) page replacement algorithm:

- This algorithm replaces the page which has not been referred for a long time.
- This algorithm is just opposite to the optimal page replacement algorithm.
- In this algorithm, we look at the past history and replace the page which is not used recently.
- This algorithm is better than FIFO but worst than optimal algorithm.
- This algorithm is used frequently because it uses past history rather than future.
- This Algorithm does not suffer with Belady's Anomaly.
- Example:- let us consider a reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 for a memory with 3 frames.

reference string																			
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0									
	0	0	0		0		0	0	3	3									
		1	1		3		3	2	2	2									
page frames																			

Total page fault= 12 (page miss)

Page hit = 8

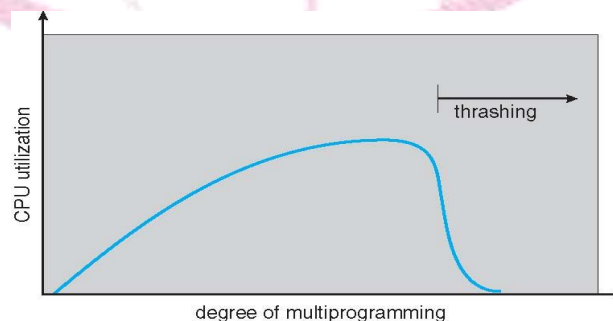
Page fault ratio = number of page fault/ (total page miss+ page hit) * 100

Page fault ratio= (12/20) * 100 = 60%

Page hit = (8/20)*100 = 40%

Thrashing

- In case, if the page fault and swapping happens very frequently at a higher rate, then the operating system has to spend more time swapping these pages. This state in the operating system is termed thrashing. Because of thrashing the CPU utilization is going to be reduced.
- During thrashing, the CPU spends less time on some actual productive work spend more time swapping. For example:- if any process does not have the number of frames that it needs to support pages in active use then it will quickly page fault. And at this point, the process must replace some pages. As all the pages of the process are actively in use, it must replace a page that will be needed again right away. Consequently, the process will quickly fault again, and again, and again, replacing pages that it must bring back in immediately. This high paging activity by a process is called thrashing.

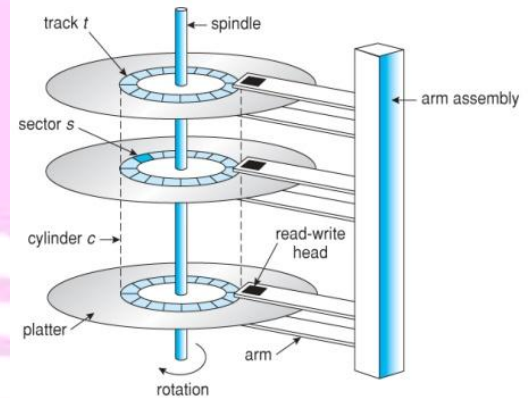


Secondary Storage structure

- files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors.
- Here we will discuss physical structure of magnetic disk.

Platter→surface→track→sector→data

- When the disk is in use the disk rotates at high speed, such as 7200 rpm (120 revolutions per second.) The rate at which data can be transferred from the disk to the computer is composed of several steps:
- **Seek time:-** the seek time also known as positioning time or random access time is defined as the time required by the read/write head to move from one track to another. Read write head moves forward and backward direction.
- **Rotation:-** time taken by the disk to rotate one full rotation around the spindle.
- **Rotation latency:-** The time taken by the platter to rotate and to reach the desired sector under the read-write head is called rotational latency. The average rotational latency for a disk is half the amount of time it takes for the disk to make one rotation.
- **Transfer time:-** data to be transfer/ transfer rate
- **Transfer rate:-** number of heads*capacity of one track*number of rotation in one second
- **Controller time:-** time taken by disk controller.
- **Queue time:-** time spend by a data in a queue.



Disk access time= Seek Time + Rotation Time + Transfer Time + Controller Time + Queue Time

Disk Scheduling

- As mentioned earlier, disk transfer speeds are limited primarily by seek times and rotational latency. When multiple requests are to be processed there is also some inherent delay in waiting for other requests to be processed.
- Bandwidth is measured by the amount of data transferred divided by the total amount of time from the first request being made to the last transfer being completed, (for a series of disk requests.)
- Both bandwidth and access time can be improved by processing requests in a good order.
- Whenever a process needs I/O to or from the disk, it issues a system call to the operating system.
- If the desired disk drive and controller are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive.
- For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the operating system chooses which pending request to service next.
- How does the operating system make this choice? Any one of several disk-scheduling algorithms can be used.
- The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

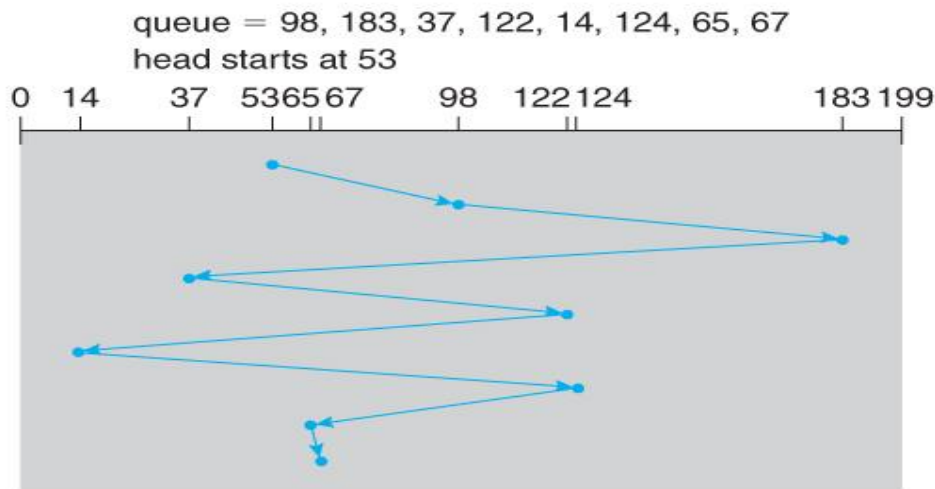
There are some following disk scheduling algorithm:-

1. FCFS Scheduling
2. SSTF Scheduling
3. Scan Scheduling
4. C-Scan Scheduling
5. Look Scheduling
6. C-Look Scheduling

1. First Come First Serve Scheduling

- It is the simplest Disk Scheduling algorithm but not very efficient. It services the I/O requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.
- If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders.

$$|(53-98)| + |(98-183)| + |(183-37)| + |(37-122)| + |(122-14)| + |(14-124)| + |(124-65)| + |(65-67)| = 640$$

**Advantages-**

- It is simple, easy to understand and implement.
- It does not cause starvation to any request.

Disadvantages

- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

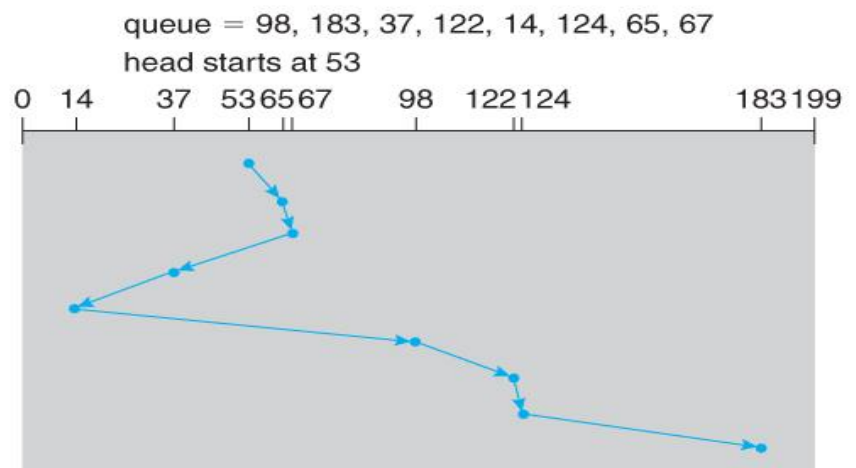
2. Shortest Seek Time First Scheduling

- The SSTF algorithm selects the request with the least seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
- Shortest Seek Time First scheduling is more efficient, but may lead to starvation if a constant stream of requests arrives for the same general area of the disk.
- For example request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67.

From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next.

Continuing, we service the request at cylinder 14 then 98, 122, 124, and finally 183. This scheduling method results in a total head movement of only 236 cylinders-little more than one-third of the distance needed for FCFS scheduling of this request queue.

$$|(53-65)| + |(65-67)| + |(67-37)| + |(37-14)| + |(14-98)| + |(98-122)| + |(122-124)| + |(124-183)| = 236$$



Advantages:-

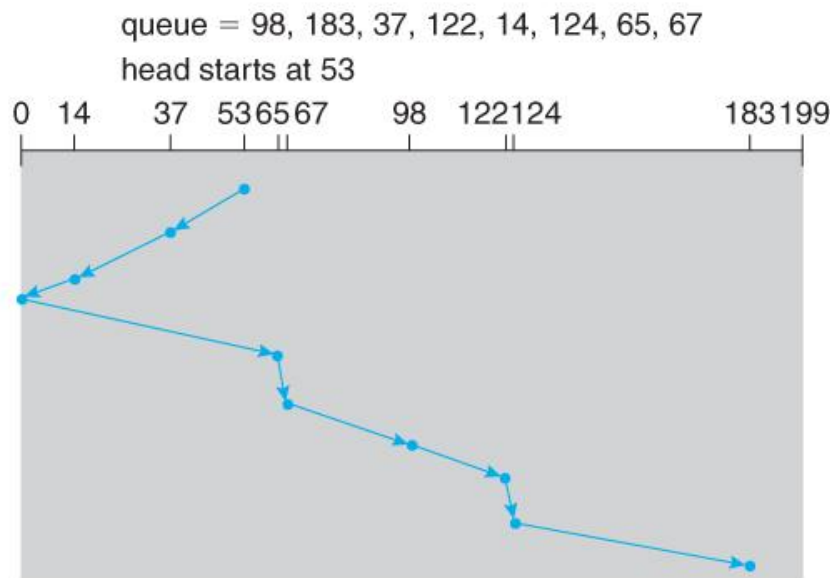
- It reduces the total seek time as compared to FCFS.
- It provides increased throughput.
- It provides less average response time and waiting time.

Disadvantages:-

- There is an overhead of finding out the closest request.
- The requests which are far from the head might starve for the CPU.
- It provides high variance in response time and waiting time.
- Switching the direction of head frequently slows down the algorithm.

3. SCAN Scheduling

- The SCAN algorithm is also known as the elevator algorithm moves back and forth from one end of the disk to the other, similarly to an elevator processing requests in a tall building.
- In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path.
- **Example: Here it is given that head is at 53 and disk arm moves towards 0.**
 $(53-37) + (37-14) + (14-0) + (0-65) + (65-67) + (67-98) + (98-122) + (122-124) + (124-183) = 236$

**Advantages-**

- It is simple, easy to understand and implement.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages-

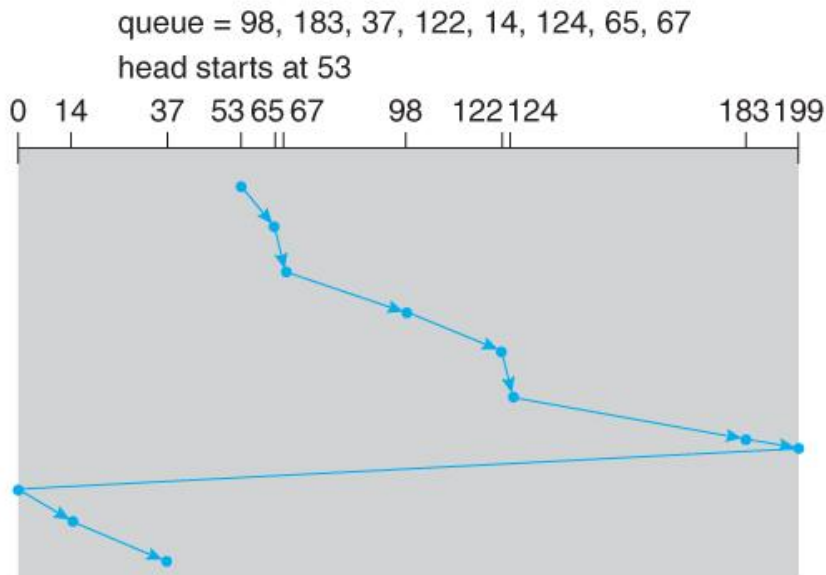
- It causes long waiting time for the cylinders just visited by the head.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

4. C-SCAN Scheduling

The Circular-SCAN algorithm is improved version of SCAN scheduling by treating all requests in a circular queue fashion. Once the head reaches the end of the disk, it returns to the other end without processing any requests, and then starts again from the beginning of the disk.

Example: Here it is given that head is at 53 and disk arm moves towards 199.

$$(53-65) + (65-67) + (67-98) + (98-122) + (122-124) + (124-183) + (183-199) + (199-0) + (0-14) + (14-37) = 382$$



Advantages-

- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
- It provides uniform waiting time.
- It provides better response time.

Disadvantages-

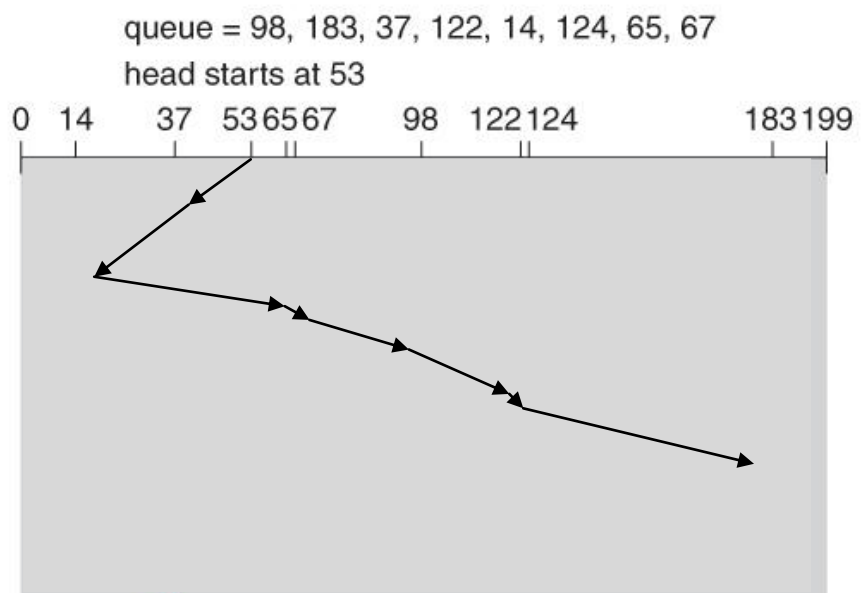
- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

5. LOOK Scheduling

- LOOK Algorithm is an improved version of the SCAN Algorithm.
- Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
- After reaching the last request at the other end, head reverses its direction.
- It then returns to the first request at the starting end servicing all the requests in between.

Example: Here it is given that head is at 53 and disk arm moves towards 0.

$$(53-37) + (37-14) + (14-65) + (65-67) + (67-98) + (98-122) + (122-124) + (124-183) = 208$$



Advantages-

- It does not cause the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages-

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head.

6. C-Look Scheduling

Circular-LOOK Algorithm is an improved version of the LOOK Algorithm.

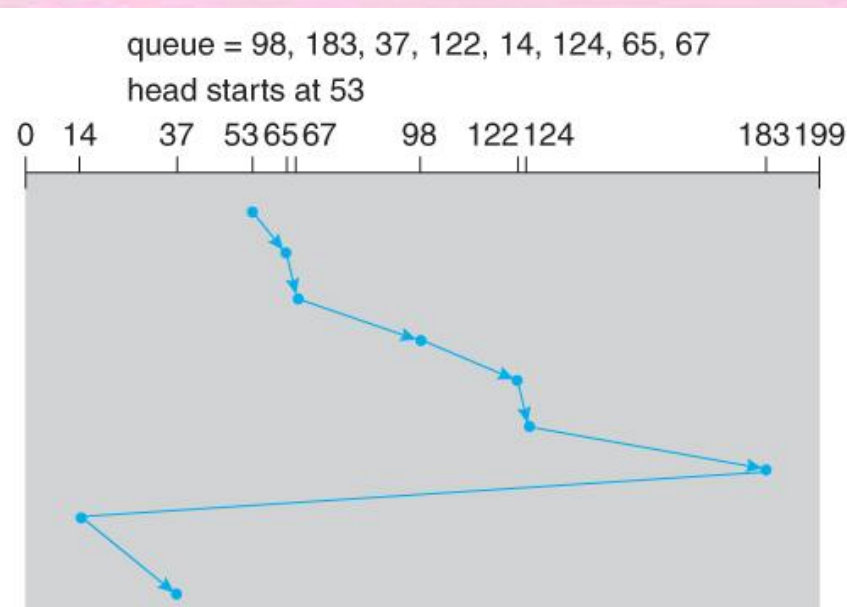
Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.

After reaching the last request at the other end, head reverses its direction.

It then returns to the first request at the starting end without servicing any request in between and the same process repeats.

Example: Here it is given that head is at 53 and disk arm moves towards 199.

$$(53-65) + (65-67) + (67-98) + (98-122) + (122-124) + (124-183) + (183-14) + (14-37) = 322$$

**Advantages-**

- It does not causes the head to move till the ends of the disk when there are no requests to be serviced.
- It reduces the waiting time for the cylinders just visited by the head.
- It provides better performance as compared to LOOK Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages-

- There is an overhead of finding the end requests.

Difference between SCAN Algorithm and LOOK Algorithm:-

The main difference between SCAN Algorithm and LOOK Algorithm is-

- SCAN Algorithm scans all the cylinders of the disk starting from one end to the other end even if there are no requests at the ends.
- LOOK Algorithm scans all the cylinders of the disk starting from the first request at one end to the last request at the other end.

Subscribe Infeepedia youtube channel for computer science competitive exams

Download Infeepedia app and call or wapp on 8004391758

Redundant Arrays of Independent Disks (RAID)

RAID combines multiple disks to improve performance, data redundancy, or both.

Why Data Redundancy?

- Redundant storage increases reliability by backing up data across multiple disks.
- In case of disk failure, data can be retrieved from another disk, unlike in non-RAID systems where a single disk failure risks data loss.

Characteristics of RAID

- **Fault Tolerance:** Ability to survive disk failures without data loss.
- **Performance:** Enhances read/write speeds compared to a single disk.

How RAID Works

- Data is stored across multiple disks to allow overlapping input/output operations, improving performance and fault tolerance.
- Appears as a single logical drive to the operating system.

RAID Techniques

- **Disk Mirroring:** Copies identical data onto multiple drives for redundancy.
- **Disk Striping:** Spreads data across multiple disks in units (stripes), improving performance by allowing parallel access.
- **Combination of Mirroring and Striping:** Used in advanced RAID levels for both performance and redundancy.

In a Single-User System:

Small stripes (e.g., 512 bytes) ensure faster access by spanning records across disks.

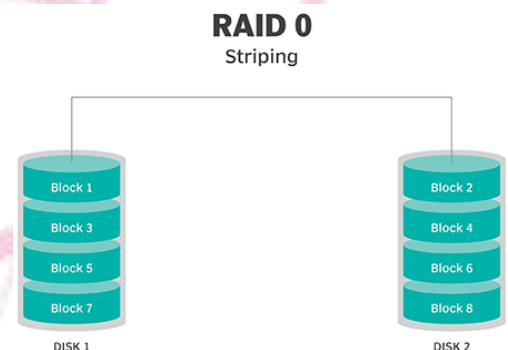
In a Multi-User System:

Larger stripe sizes hold typical records to allow overlapped disk I/O, improving performance.

Levels of RAID

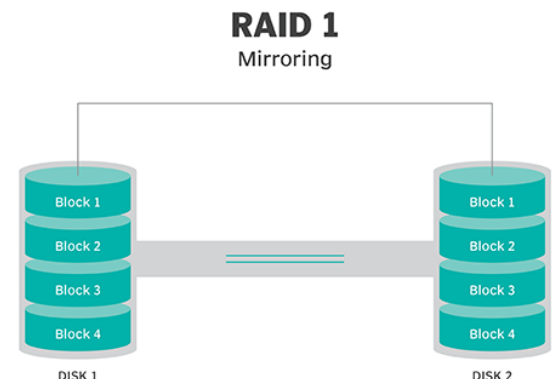
1. RAID-0 (Striping)

- No redundancy, only striping.
- Best performance but no fault tolerance.



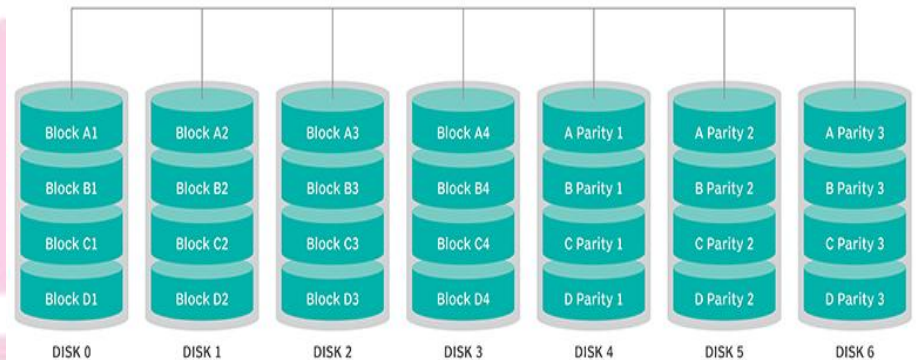
2. RAID-1 (Mirroring)

- Data is mirrored across at least two drives.
- Improved read performance, same write performance as a single disk.
- No striping.

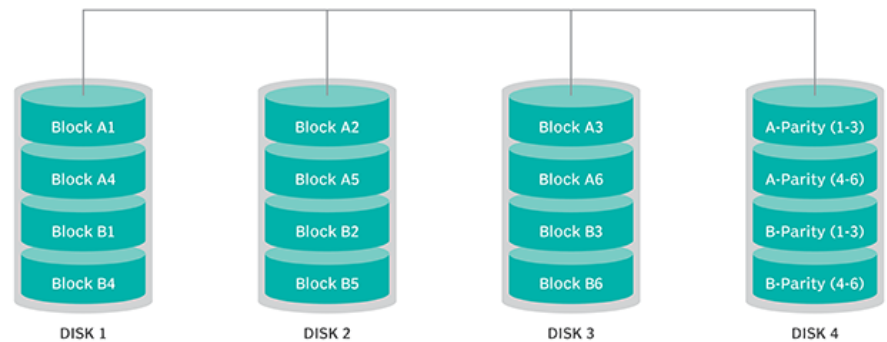


3. RAID-2

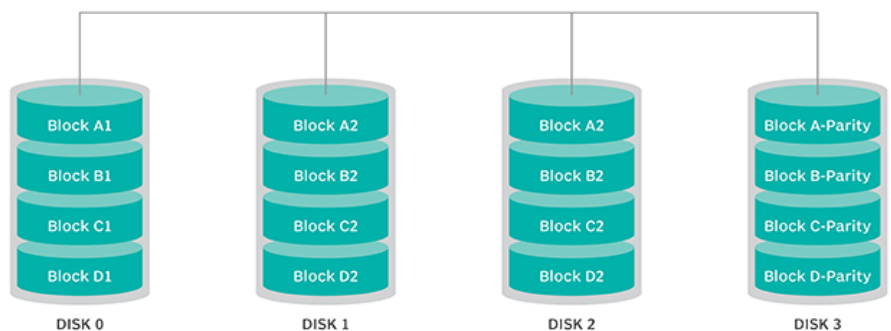
- Uses striping with error checking and correction (ECC) information on separate disks.
- Employs Hamming code parity.
- No advantages over RAID-3; rarely used.

RAID 2**4. RAID-3**

- Striping with one dedicated drive for parity information.
- Uses ECC for error detection and recovery.
- All drives are addressed in I/O operations, limiting I/O overlap.
- Best for single-user systems with large, sequential data.

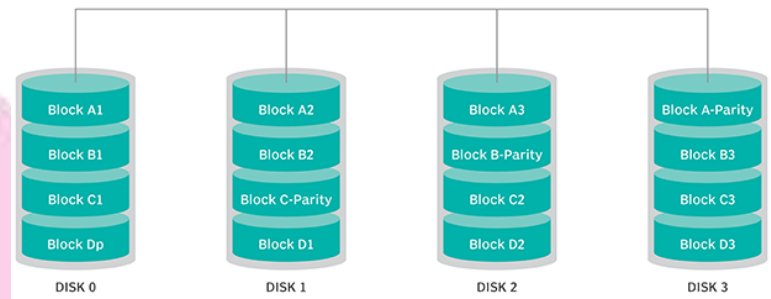
RAID 3
Parity on separate disk**5. RAID-4:**

- Uses large stripes with parity stored on a dedicated drive.
- Allows overlapped I/O for read operations.
- No overlapped I/O for write operations due to parity updates.

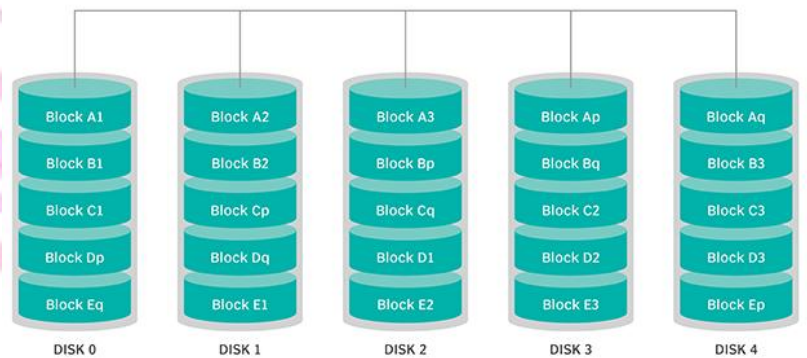
RAID 4

6. RAID-5

- Parity is striped across all drives, allowing operation even if one drive fails.
- Provides better performance than a single drive but lower than RAID-0.
- Requires at least three disks; five disks recommended for better performance.

RAID 5**7. RAID-6:**

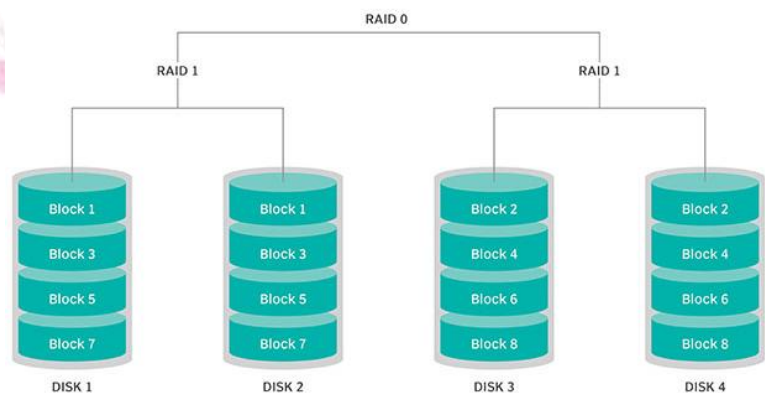
- Similar to RAID-5 but with a second parity block for extra protection.
- Can tolerate up to two disk failures.
- Slower write performance compared to RAID-5 due to additional parity.

RAID 6**8. RAID-10 (RAID 1+0)**

Combines RAID-1 (mirroring) and RAID-0 (striping).
Offers high performance with redundancy, but at a higher cost.
Data is mirrored and the mirrors are striped for improved speed and fault tolerance.

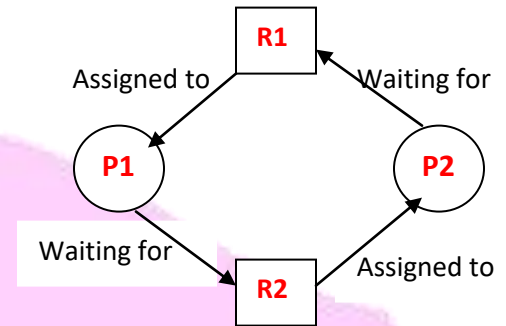
RAID 10 (RAID 1+0)

Stripe + Mirror



Deadlocks in Operating System

- Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.



- Example 1:** when 2 trains are coming towards each other none of the train can move once they come in front of each other. This is the situation of deadlock now both trains will get stuck and moves nowhere. Similar situation may occur in the Operating Systems when 2 or more processes hold some resources & wait for resources held by others.

Resource instances:- If we say resource printer has 2 instances it means there are 2 printers (as a resource). If a system has 2 CPU then the resource type a CPU has 2 instances. So resource instance tells the number of similar resources. There are multiple types of resources such as Memory space, CPO cycles, files, I/O devices & so on.

Under the normal mode of operation a process may utilize a resource in the Operating System only the following sequence:

- Request:** The process requests the resources. If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
- Use:** The process can operate on the resource (Example-if resource is printer, the process can print on the printer)
- Release:** The process releases the resource.

Some Necessary Conditions for deadlock

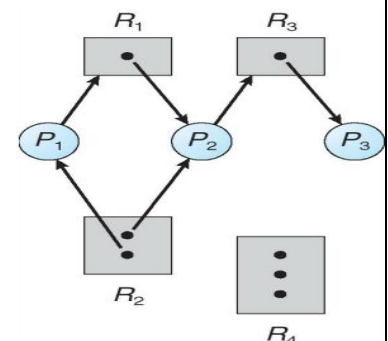
Deadlock can arise if the following 4 Conditions hold simultaneously in the system.

- Mutual exclusion:** At least one resource must be held in a non-sharable mode, i.e. only one process at a time can use the resource if another process requests that resource, the requesting process must be delayed until the resource has been released.
- Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
- No Preemption:** Resources cannot be preempted; i.e. a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- Circular wait:** A set $\{p_0, p_1, p_2, \dots, p_n\}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for a resource held by P_2 ... P_{n-1} is waiting for resource held by P_n and is waiting for a resource held by p_0



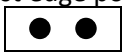
All 4 Condition must hold for a deadlock to occur. The circular-wait conditions imply the hold and wait Condition, so the 4 Conditions are not completely independent.

Resources allocation graph : Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph.

- This graph has set of vertices V & set of edges E .
- The set of vertices V is partitioned into two different types of nodes:
 $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the -active processes in the system, and $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- The directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$; It signifies that processes P_i has requested an instance of resource type R_j and is currently waiting for that resource.
- A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$; It signifies



that an instance of resource type R_j has been allocated to Process P_i .

- A directed edge $P \rightarrow R$ is called a request edge.
- A directed edge $R \rightarrow P$ is called an assignment edge.
- Pictorially, we represent each process P_i as a circle 
- Each resource type R_i as a rectangle 
- Since resource type R_i , may have more than one instance, we represent each such instance as a dot within the rectangle.
- Note that a request edge points to only the rectangle R_j , whereas an assignment edge must also designate one of the dots in the rectangle. 
- When process P requests an instance of resource type R_i , a request edge is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge.
- When the process no longer needs access to the resource, it releases the resource; as a result, the assignment edge is deleted.

In the resource-allocation graph:

- If the graph contains no cycles, then no process in the system is deadlocked.
- If the graph does contain a cycle, then a deadlock may exist.

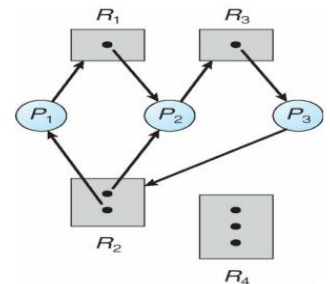
Case1:- If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock.

Example:- that process P_3 requests an instance of resource type R_2 . Since no resource instance is currently available, a request edge $P_3 \rightarrow R_2$ is added to the graph.

At this point, two minimal cycles exist in the system:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



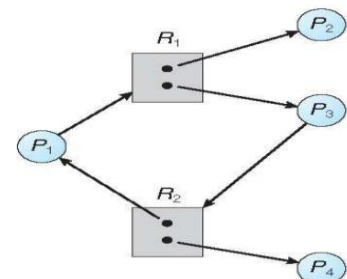
Processes P_1 , P_2 , and P_3 are deadlocked. Process P_2 is waiting for the resource R_3 , which is held by process P_3 . Process P_3 is waiting for either process P_1 or process P_2 to release resource R_2 . In addition, process P_1 is waiting for process P_2 to release resource R_1 .

Case2:- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

Example:- In this example, we also have a cycle:

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

However, there is no deadlock. Observe that process P_4 may release its instance of resource type R_2 . That resource can then be allocated to P_3 , breaking the cycle.



Methods for Handling Deadlocks

We can deal with the deadlock problem in one of three ways:-

1. Deadlock prevention and avoidance
2. Deadlock detection and recovery
3. Ignore the problem altogether

Deadlock Prevention

We can prevent the occurrence of a deadlock by eliminating any of the 4 necessary conditions (mutual exclusion, Hold & wait, no Preemption & circular wait).

1. Mutual Exclusion

- Mutual exclusion is required for non-sharable resources (e.g., a printer).
- Sharable resources (e.g., read-only files) do not require mutual exclusion and cannot cause deadlocks.
- Denying mutual exclusion for non-sharable resources is impractical as they are inherently non-sharable.

2. Hold and Wait

- To prevent hold-and-wait, the system ensures processes do not hold resources while requesting others:

Method 1: A process requests all required resources before execution starts.

Disadvantage:

- Resources are held longer than necessary, leading to low utilization.

Method 2: A process releases all currently held resources before requesting new ones.

Disadvantage:

- Starvation may occur if popular resources are frequently in demand.

3. No Preemption

To prevent the no-preemption condition:

- If a process requests a resource that is unavailable, all currently held resources are released (preempted).
- Preempted resources are added to the waiting list and reallocated when the process can regain all its required resources.
- Applicable Resources: Resources whose state can be saved (e.g., CPU registers, memory).
- Non-Applicable Resources: Non-preemptable resources (e.g., printers, tape drives).

4. Circular Wait:

One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of priority.

1. One protocol to ensure that the circular wait condition never holds is "Impose a linear ordering of all resource types."

Then, each process can only request resources in an increasing order of priority.

For example, set priorities for $R1 = 1$, $R2 = 2$, $R3 = 3$, and $R4 = 4$. With these priorities, if process P wants to use $R1$ and $R3$, it should first request $R1$, then $R3$.

2. Another protocol is "Whenever a process requests a resource R_i , it must have released all resources R_j with priority $(R_j) \geq$ priority (R_i) ."

If these two protocols are used, then the circular-wait condition cannot hold.

Disadvantage:

May result in low device utilization and reduced system throughput.

Deadlock Avoidance

- Deadlock avoidance requires prior knowledge of resource usage.
- Each process must declare the maximum resources it may need.
- The system dynamically examines the resource-allocation state (available, allocated resources, and maximum demands) to prevent circular wait conditions.
- Algorithms ensure the system never enters a deadlocked state.

Safe State:-

- A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.
- A system is in a safe state only if there exists a safe sequence.
- A safe state is not a deadlocked state. Conversely, a deadlocked state is an unsafe state.
- Not all unsafe states are deadlocks, however an unsafe state may lead to a deadlock.

Example:- we consider a system with 12 magnetic tape drives and 3 processes: P0, P1, and P2.

Process P0 requires 10 tape drives, process P1 requires 4 tape drives, and process P2 requires 9 tape drives. Suppose that, at time t0, process P0 is holding 5 tape drives, process P1 is holding 2 tape drives, and process P2 is holding 2 tape drives. (Thus, there are 3 free tape drives.)

Process	Maximum need	Allocation
P0	10	5
P1	4	2
P2	9	2

The sequence <P1, P0, P2> is a safe sequence.

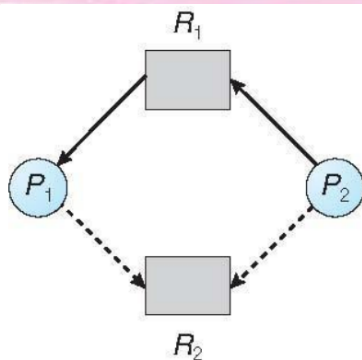
Explanation: - Process P1 can immediately be allocated all its tape drives and then return them (the system will then have five available tape drives); then process P0 can get all its tape drives and return them (the system will then have ten available tape drives); and finally process P2 can get all its tape drives and return them (the system will then have all twelve tape drives available).

A system can go from a safe state to an unsafe state. If the sequence get changed in this example.

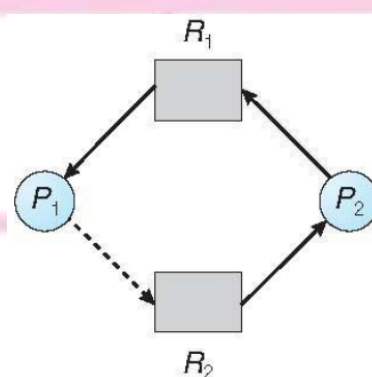
Deadlock Avoidance Algorithms

1. Resource-Allocation-Graph Algorithm:

- Used when each resource type has only one instance.
- Introduces claim edges (dashed lines) indicating future resource requests.
- When a process requests a resource, the claim edge is converted to a request edge.
- When a resource is released, the assignment edge reverts to a claim edge.



Resource-allocation graph for deadlock avoidance.



An unsafe state in a Resource-allocation graph

Banker's Algorithm

- Handles multiple instances of resource types.
- Ensures the system remains in a safe state after resource allocation.
- Requires processes to declare maximum resource needs in advance.
- Resources are allocated only if they maintain system safety.

Key Data Structures:

Available: Vector indicating available instances of each resource type.

Max: Matrix defining the maximum demand of each process.

Allocation: Matrix showing current resource allocation to processes.

Need: Matrix indicating remaining resource needs, calculated as:

$$\text{Need} = \text{Max} - \text{Allocation}$$

There are 2 algorithm works for Banker's Algorithm.

1. Safety Algorithm:-

We can now present the algorithm for finding out whether or not a system in a safe state.

2. Resource-Request Algorithm:-

Next, we describe the algorithm for determining whether requests can be safely granted. Let $Request_i$ be the request vector for process P_i . If $Request_i[j] = k$, then process P_i wants k instances of resource type R_j . When a request for resources is made by process P_i , the following actions are taken:

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$Available = Available - Request_i;$
 $Allocation = Allocation + Request_i;$
 $Need_i = Need_i - Request_i;$

If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $Request_i$ and the old resource-allocation state is restored.

Example: Consider a system that contains five processes P_1, P_2, P_3, P_4, P_5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation			Max	A	B	C	Available		
	A	B	C					A	B	C
P1	0	1	0	7	5	3		3	3	2
P2	2	0	0	3	2	2				
P3	3	0	2	9	0	2				
P4	2	1	1	2	2	2				
P5	0	0	2	4	3	3				

Need = Max - Allocation

Process	Need		
	A	B	C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

We execute the banker's algorithm to find the safe state and the safe sequence like $\langle P_2, P_4, P_5, P_1, P_3 \rangle$

Advantages of Banker's Algorithm

- Supports multiple resource types to meet process requirements.
- Processes provide prior information about resource needs and usage duration.
- Helps the OS manage and control resource allocation efficiently.
- The Max attribute ensures each process is aware of its maximum allowable resources.

Disadvantages of Banker's Algorithm

- Works with a fixed number of processes; no new processes can be added during execution.
- Processes cannot modify their maximum resource requirements during execution.
- Processes must declare maximum resource needs in advance.
- Resource requests are granted within a finite time, potentially extending up to a year.

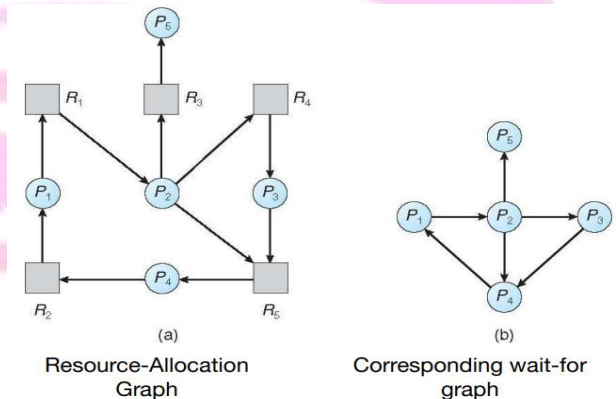
Deadlock Detection

If neither prevention nor avoidance is used, the system detects and resolves deadlocks using:

1. An algorithm to check if a deadlock has occurred.
2. An algorithm to recover from deadlocks.

Algorithms for Deadlock Detection**1. Wait-for Graph:**

- Applicable when each resource has a single instance.
- Derived by removing resource nodes from the resource-allocation graph and collapsing edges.
- An edge $P_i \rightarrow P_j$ implies P_i is waiting for P_j to release a needed resource.
- Deadlock exists if the wait-for graph contains a cycle.
- The system maintains and periodically checks the wait-for graph for cycles to detect deadlocks.
- Cycle detection requires $O(n^2)$ operations, where n is the number of processes.

**Banker's Algorithm for deadlock detection**

- The wait for graph scheme is not applicable to a resource-allocation system with multiple instances of each resource type.
- When multiple instance of each resources type is available then we use Banker's Algorithm for deadlock detection. The following data structures are defined for Banker's Algorithm.
- **Available:-** A vector of length m indicates the number of available resources of each type.
- **Allocation:-** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- **Request:-** An $n \times m$ matrix indicates the current request of each process. If $\text{Request}[i][j]$ equals k , then process P_i , is requesting k more instances of resource type R_j .
- Banker's algorithm includes a Safety Algorithm / Deadlock Detection Algorithm

Usage of Deadlock Detection Algorithm

- **Frequent Invocation:** Necessary if deadlocks occur often, as resources remain idle and deadlock cycles may grow.
- **Per Request Invocation:** Detects deadlocks immediately, including the process causing the issue, but incurs high computational overhead.
- **Periodic Invocation:** Reduces overhead by running the algorithm at intervals (e.g., hourly or when CPU utilization drops below 40%).

Recovery from Deadlock

Once a deadlock is detected, the system can:

1. Manual Intervention: Inform the operator to handle the deadlock.
2. Automated Recovery:
 - **Process Termination:** Abort processes to break the deadlock.
 - **Resource Preemption:** Preempt resources from deadlocked processes.

Process Termination

1. Abort All Deadlocked Processes
 - Breaks the deadlock cycle completely.
 - High cost: Partial computations are lost, requiring recomputation.
2. Abort One Process at a Time
 - Terminate one process and recheck for deadlock using the detection algorithm.
 - High overhead: Multiple checks increase computation time.
 - Risk of data inconsistency: Processes updating files or using printers may leave these in an incorrect state, requiring reset.

Factors for Selecting Processes to Abort

- Process priority.
- Time already computed and remaining computation time.
- Type and amount of resources held (ease of preemption).
- Additional resources needed for completion.
- Number of processes to be terminated.
- Interactive vs. batch process nature.
- Focus is on minimizing termination cost while resolving the deadlock.

Resource Preemption

Deadlocks can be resolved by preempting resources from processes and reallocating them to break the deadlock cycle.

Key Issues in Resource Preemption:**1. Selecting a Victim**

- **Determine which process to preempt based on cost factors like:**
 1. Number of resources held by the process.
 2. Execution time already consumed.
- **Aim to minimize the overall cost of preemption.**

2. Rollback

- Preempted processes must roll back to a safe state to restart.
- Simplest approach: Total rollback (abort and restart).
- Efficient approach: Partial rollback to the point necessary to resolve deadlock (requires detailed state tracking).

3. Starvation Prevention

- Ensure a process is selected as a victim only a limited number of times.
- Incorporate the number of rollbacks into the cost calculation to avoid repeated victimization.