

STYUDY MATRIAL

OF

OPERATING SYSTEM

Prepared by
Smt.Pranati Pattnaik
Sr.Lecturer in CA
Govt.Polytechnic , Bhubaneswar

Theory 1- OPERATING SYSTEM

4th Semester IT

Unit -1 Introduction

An **operating system (OS)** is system software that manages computer hardware, software resources, and provides common services for computer programs.

An **Operating System (OS)** is an interface between computer user and computer hardware. An **operating system** is software which performs all the basic tasks **like file management, memory management, process management, handling input and output**, and controlling peripheral devices such as disk drives and printers.

An operating system performs two basically unrelated functions

1. The operating system as a Resource manager.

A user process accesses several hardware and software resources during its execution. Resource management performs the following:

- (a) Time management. (CPU, DISK Scheduling)
- (b) Space management. (Primary and Secondary memory)
- (c) Process Synchronization and Deadlock handling.
- (d) Accounting and Status Information.

2. The operating system as a Virtual machine or extended machine

The operating system hides the truth and low level details about hardware from the programmer and provides users with a much friendlier interface to the machine. The function of operating system is to present user with equivalent of an **extended machine or virtual machine** that is easier to program than the underlying hardware.

The user of the virtual machine have the illusion that each one of them is the sole user of the machine, even though the machine may be operating in a multiuser environment. It performs the following tasks

- (a) Procession creation and management, File manipulation, Interrupt handling, I/O Operation
- (b) Error detection and handling
- (c) Protection and security

We can say that operating system is a Resource Manager as well as a Virtual Machine Manager.

Unit-2 Process Scheduling

2.1 Process Concepts

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process.

Process Life Cycle

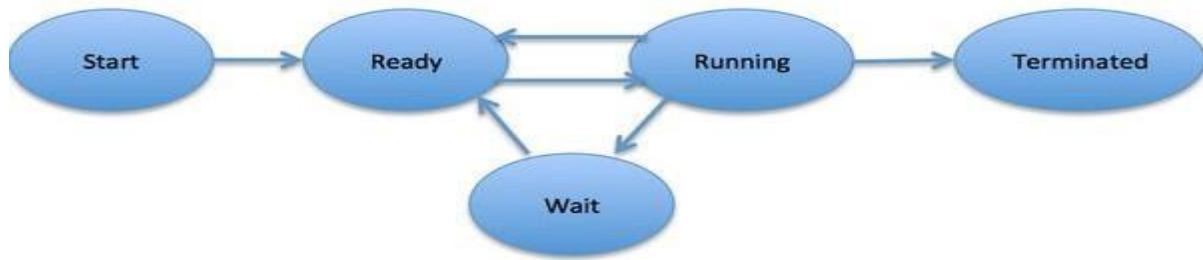
When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

Process life cycle in OS is one of the five states in which a **process** can be starting from the time it has been submitted for execution, till the time when it has been executed by the system.

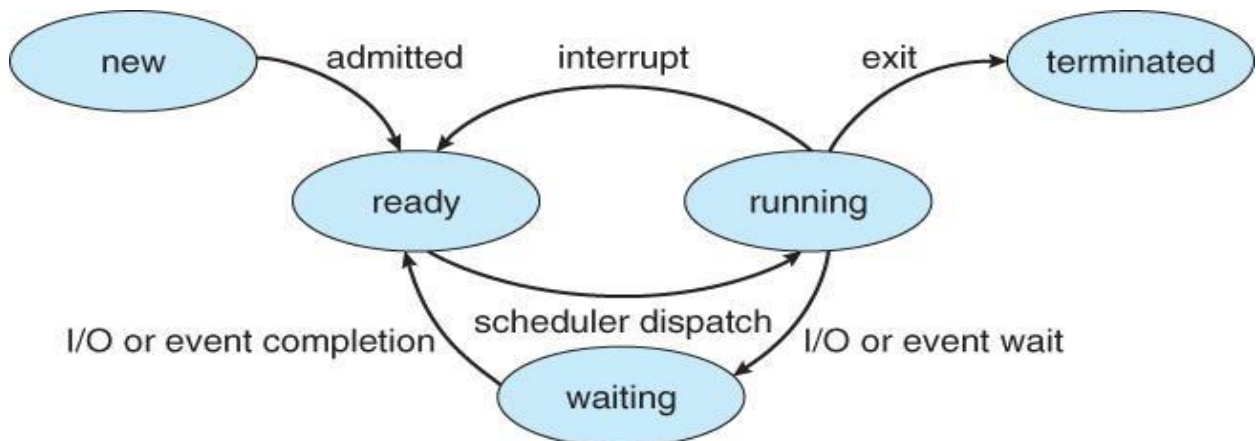
A **process** can be in any of the following states

In general, a process can have one of the following five states at a time.

1. **Start**: This is the initial state when a process is first started/created
2. **Ready**: The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3. **Running**: Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4. **Waiting**: Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5. **Terminated or Exit**: Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



- First, the process is "created" by being loaded from a secondary storage device into main memory. After that the process scheduler assigns it the "waiting" state.
- While the process is "waiting", it waits for the scheduler to do a so-called context switch. The context switch loads the process into the processor and changes the state to "running" while the previously "running" process is stored in a "waiting" state.
- If a process in the "running" state needs to wait for a resource (wait for user input or file to open, for example), it is assigned the "blocked" state. The process state is changed back to "waiting" when the process no longer needs to wait (in a blocked state).
- Once the process finishes execution, or is terminated by the operating system, it is no longer needed. The process is removed instantly or is moved to the "terminated" state. When removed, it just waits to be removed from main memory.



Process Control

Process control is the ability to monitor and adjust a **process** to give a desired output. It is used in industry to maintain quality and improve performance.

Process Control Block is a data structure that contains information of the **process** related to it. The process control block is also known as a task control block, entry of the process table, etc. Another responsibility of an operating system is to collect all information that it needs about a particular process into a data structure called **Process Control Block**.

Structure of the Process Control Block

The process control stores many data items that are needed for efficient process management. When a process is created, the operating system creates a corresponding PCB and when it is terminated, its PCB is released to the pool of free memory locations from which new PCB are drawn.

diagram –



Process Control Block (PCB)

The following are the data items –

1. Process State

This specifies the process state i.e. new, ready, running, waiting or terminated.

2. Process Number or ID

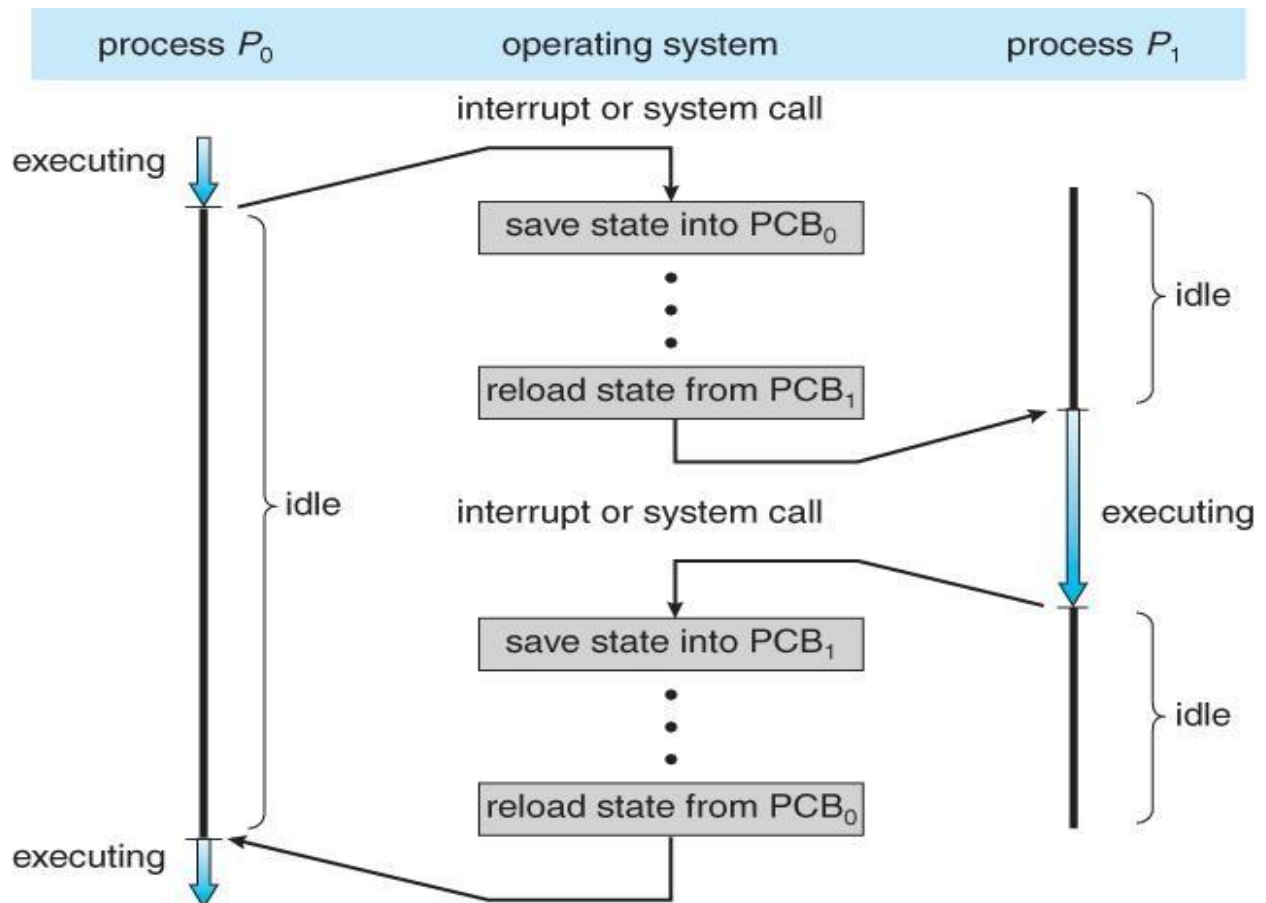
It is a unique process number or process identifier that identifies each process. This shows the number of the particular process.

3. Program Counter

This contains the address of the next instruction that needs to be executed in the process.

4. Registers

This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc. Whenever a processor switches over from one process to another process, information about the current status of the old process is saved in the register along with the program counter so that the process be allowed to continue correctly afterwards.



This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.

5. List of Open Files

These are the different files that are associated with the process

6. CPU Scheduling Information

The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.

7. Memory Management Information

The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.

8. I/O Status Information

This information includes the list of I/O devices used by the process, the list of files etc.

9. Accounting information

The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information.

Location of the Process Control Block

The process control block is kept in a memory area that is protected from the normal user access. This is done because it contains important process information. Some of the operating systems place the PCB at the beginning of the kernel stack for the process as it is a safe location.

Interacting processes

Process Interaction is a model of managing parallel or concurrent **processes** by defining how data between these **processes** is exchanged and how the **processes** are synchronized with each other. ... Such a framework should provide the ability to use data about interacting **processes** for performing necessary operations and tasks.

Inter process communication

Inter process communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

A diagram that illustrates inter process communication is as follows –



Inter-process communication or **inter process communication (IPC)** refers specifically to the mechanisms an [operating system](#) provides to allow the [processes](#) to manage shared data. The Processes may be running on single or multiple computers connected by a network.

Typically, applications can use IPC, categorized as [clients and servers](#), where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in [distributed computing](#).

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more **processes** or programs. ... Since every single user request may result in multiple **processes** running in the **operating system**, the **process** may require to **communicate** with each other.

IPC is a facility provided by an OS using **which cooperating processes** can communicate with each other. IPC is based on the use of share variables i.e the

variables that can be referenced by more than one process or message passing. Synchronization is often necessary when processes communicate. Processes are executed with high speed. But for communication one process must perform some action such as setting the value of a variable or sending some message that the other detects.

Concurrent process is the appearance of simultaneous execution of multiple processes. It includes

- Communication among process
- Sharing and competing among resources
- Synchronization of activities
- Allocation of process time
- Allocation of memory

It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.

Processes in the system are of two types:

- Independent Processes
- Cooperating Processes

1. Independent processes are those processes which neither affect nor get affected by other processes running in the system. In other words, Any process that does not share data with other processes is an independent process.

2. Cooperating processes are those if they can affect or get affected by the other processes running in the system. In other words, a process that shares data with other processes is a cooperating process.

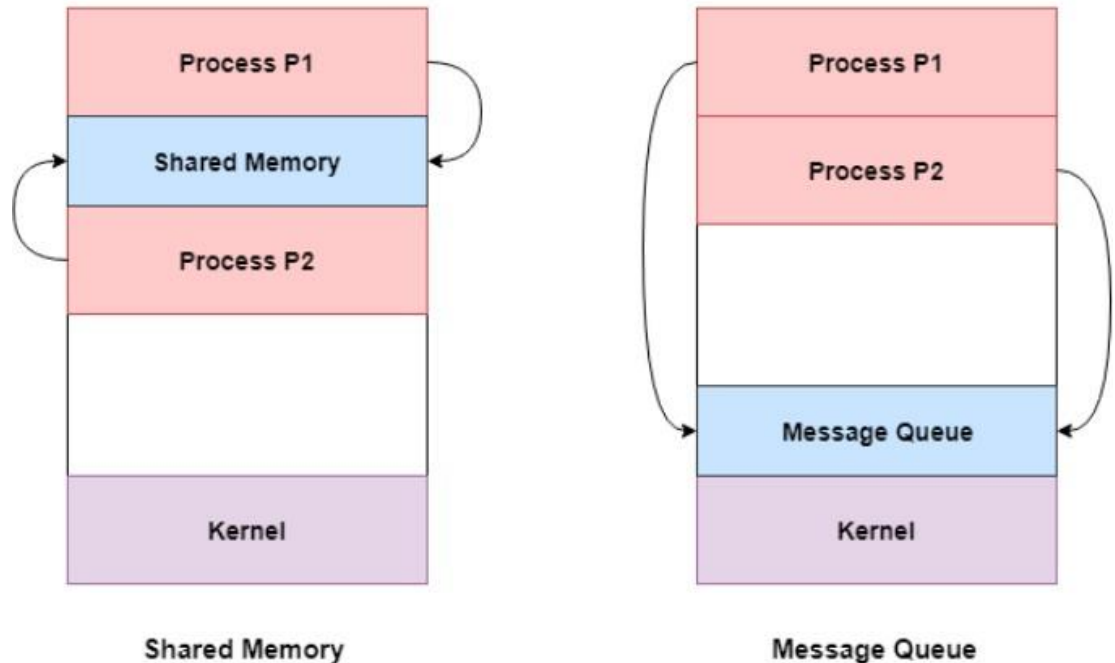
Inter-process communication is useful for creating *cooperating* processes. The cooperating processes are important because they provide:

1. **Information sharing among processes:** Since several users may be interested in the same piece of information. We must provide environment to allow concurrent access to such information.
2. **Increase in computational speed:** To run the task faster, we must break into subtasks, each of which can be executing in parallel order. Speedup can be achieved only if the computer has multiple processing elements.
3. **It provides modularity:** We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.
4. **It provides convenience in accessing data:** Even an individual user can work on many tasks at the same time. For example a user can be editing, printing and compiling in parallel.

There are two fundamental models of inter process communication

- (A) Shared memory
- (B) Message passing

Approaches to Interprocess Communication



(A). Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. Windows operating systems use shared memory.

Advantages of Shared Memory

1. Fastest bidirectional communication method
2. Can be used with any number of processes.
3. It saves resources.

Disadvantages of Shared Memory

1. Data inconsistency occurs like the Lost update. It needs a concurrency control mechanism
2. Lack of data protection

(B). Message passing

It is the second method for inter process communication. It provides two operations for processes to communicate.

1. Send (message)
2. Receive (message)

Message passing is slower as compared to shared memory method. This is because it makes use of system calls to provide communication between processes.

Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for inter process communication and are used by most operating systems.

Section 2.3 Process Scheduling

The **process scheduling** is the activity of the **process** manager that handles the removal of the running **process** from the CPU and the selection of another **process** on the basis of a particular strategy.

Process Scheduling is an OS task that schedules processes of different states like ready, waiting, and running.

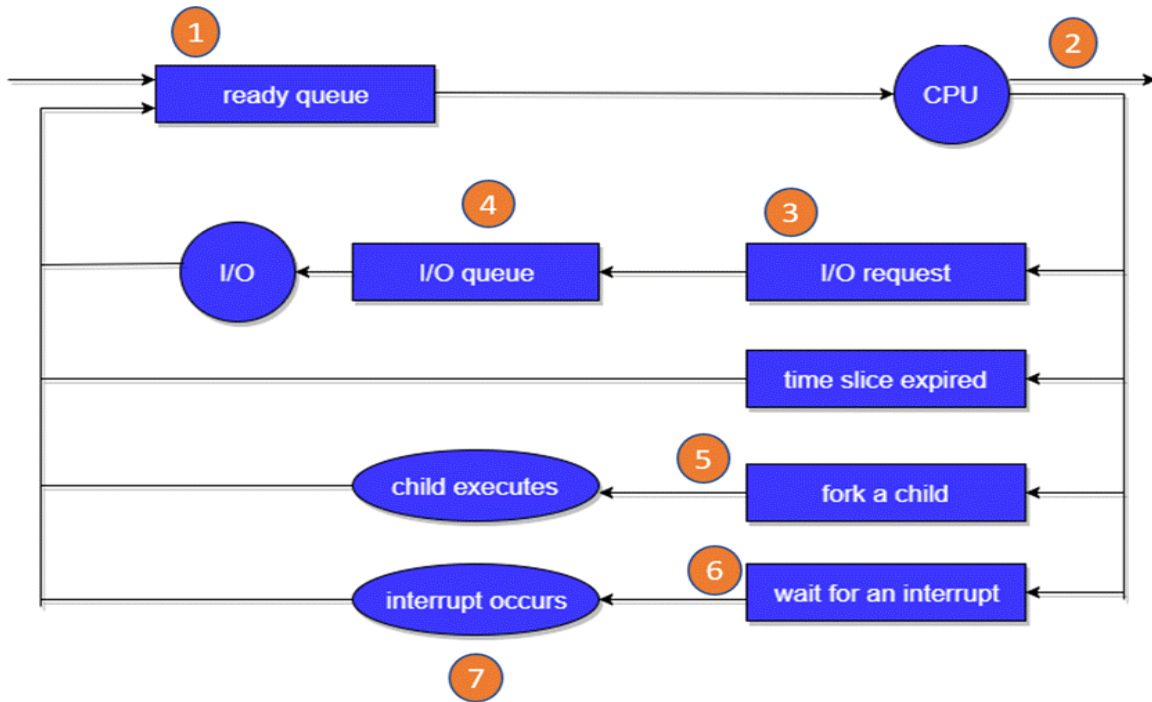
Process scheduling allows OS to allocate a time interval of CPU execution for each process. Another important reason for using a process scheduling system is that it keeps the CPU busy all the time. This allows you to get the minimum response time for programs.

Process Scheduling Queues

Process Scheduling Queues help you to maintain a distinct queue for each and every process states and PCBs. All the process of the same execution state are placed in the same queue. Therefore, whenever the state of a process is modified, its PCB needs to be unlinked from its existing queue, which moves back to the new state queue.

Three types of operating system queues are:

1. **Job queue** – It helps you to store all the processes in the system.
2. **Ready queue** – This type of queue helps you to set every process residing in the main memory, which is ready and waiting to execute.
3. **Device queues** – It is a process that is blocked because of the absence of an I/O device.



In the above-given Diagram,

- Rectangle represents a queue.
- Circle denotes the resource.
- Arrow indicates the flow of the process.

1. Every new process first put in the Ready queue. It waits in the ready queue until it is finally processed for execution. Here, the new process is put in the ready queue and wait until it is selected for execution or it is dispatched.
2. One of the processes is allocated the CPU and it is executing.
3. The process should issue an I/O request
4. Then, it should be placed in the I/O queue.
5. The process should create a new subprocess.
6. The process should be waiting for its termination.
7. It should remove forcefully from the CPU, as a result interrupt. Once interrupt is completed, it should be sent back to ready queue.

Context Switch

In computing, a **context switch** is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point. This allows multiple processes to share a single central processing unit (CPU) and is an essential feature of a multitasking operating system.

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point

as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

Example:

Process 1 is switched out and Process 2 is switched in because of an interrupt or a system call. Context switching involves saving the state of Process 1 into PCB1 and loading the state of process 2 from PCB2. After some time again a context switch occurs, and Process 2 is switched out and Process 1 is switched in again. This involves saving the state of Process 2 into PCB2 and loading the state of process 1 from PCB1

Schedulers

A scheduler is a type of system software that allows you to handle process scheduling. The scheduler is an operating system module that selects the next jobs to be admitted into the system and the next process to run.

Type of Process Schedulers

A scheduler is a type of system software that allows you to handle process scheduling.

There are mainly three types of Process Schedulers:

1. Long Term
2. Short Term
3. Medium Term

Long Term Scheduler

Long term scheduler is also known as a **job scheduler**. This scheduler regulates the program and select process from the queue and loads them into memory for execution. It also regulates the degree of multi-programing.

However, the main goal of this type of scheduler is to offer a balanced mix of jobs, like Processor, I/O jobs., that allows managing multiprogramming.

Medium Term Scheduler

Medium-term scheduling is an important part of **swapping**. It enables you to handle the swapped out-processes. In this scheduler, a running process can become suspended, which makes an I/O request.

A running process can become suspended if it makes an I/O request. A suspended processes can't make any progress towards completion. In order to

remove the process from memory and make space for other processes, the suspended process should be moved to secondary storage.

Short Term Scheduler

Short term scheduling is also known as **CPU scheduler**. The main goal of this scheduler is to boost the system performance according to set criteria. This helps you to select from a group of processes that are ready to execute and allocates CPU to one of them. The dispatcher gives control of the CPU to the process selected by the short-term scheduler.

Section 2.3 Job Scheduling

Job scheduling is the process of allocating **system** resources to many different tasks by an **operating system** (OS). The **system** handles prioritized **job** queues that are awaiting CPU time and it should determine which **job** to be taken from which queue and the amount of time to be allocated for the **job**. This type of scheduling makes sure that all jobs are carried out fairly and on time. Most OSs like Unix, Windows, etc., include standard job-scheduling abilities.

Scheduling Criteria

A scheduler algorithm is evaluated against some widely accepted performance criteria as follows:

(a). CPU utilization : It is defined as the average fraction of time during which CPU is busy, executing either user programs or system modules.

(b). Throughput: It is defined as the average amount of work completed per unit time.

(c). Turn Around Time (TAT) : It is defined as the total time elapsed from the time the job is submitted to the time the job is completed.

$TAT = (\text{Process finish time} - \text{Process arrival time})$

(d). Waiting Time (WT): It is defined as the total time spent by the job while waiting in suspended state or ready state in a multiprogramming environment.

$WT = (\text{Turn around time} - \text{Processing time})$

e). Response time(RT): It is an amount to time in which the request was submitted until the first response is produced.

Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)

2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling



First Come First Serve

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

Characteristics of FCFS method:

- It offers non-preemptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.

Shortest Remaining Time

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

Characteristics of SRT scheduling method:

- This method is mostly applied in batch environments where short jobs are required to be given preference.
- This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
- Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

Priority Based Scheduling

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

Round-Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

Characteristics of Round-Robin Scheduling

- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
- It is a real time system which responds to the event within a specific time limit.

Shortest Job First

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

Characteristics of SJF Scheduling

- It is associated with each job as a unit of time to complete.
- In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
- It is Implemented with non-preemptive policy.
- This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.

- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Multiple-Level Queues Scheduling

This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc. However, this is not an independent scheduling OS algorithm as it needs to use other types of algorithms in order to schedule the jobs.

Characteristic of Multiple-Level Queues Scheduling:

- Multiple queues should be maintained for processes with some characteristics.
- Every queue may have its separate scheduling algorithms.
- Priorities are given for each queue.

Section 2.4 Process Synchronization

It is the task phenomenon of coordinating the execution of **processes** in such a way that no two **processes** can have access to the same shared data and resources.

- It is a procedure that is involved in order to preserve the appropriate order of execution of cooperative processes.
- In order to synchronize the processes, there are various synchronization mechanisms.
- Process Synchronization is mainly needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or any data at the same time.

Example: Process A changing the data in a memory location while another process B is trying to read the data from the **same** memory location. There is a high probability that data read by the second process will be erroneous.



Memory

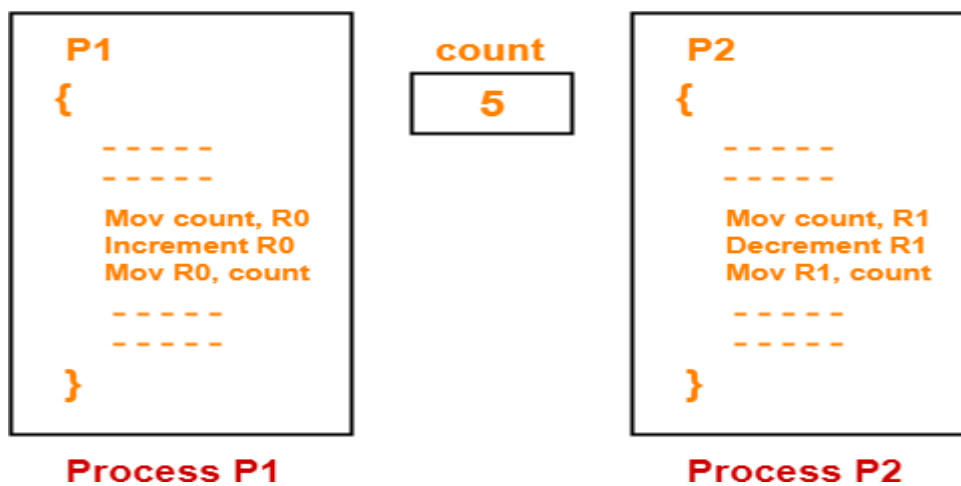
Race Condition

At the time when more than one process is either executing the same code or accessing the same memory or any shared variable; In that condition, there is a

possibility that the output or the value of the shared variable is wrong so for that purpose all the processes are doing the race to say that my output is correct. This condition is commonly known as a **race condition**.

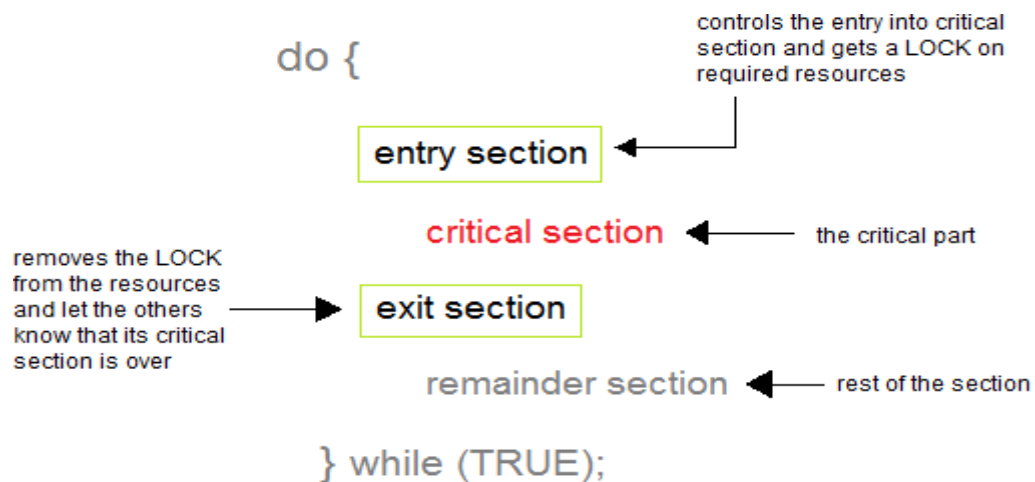
As several processes access and process the manipulations on the same data in a concurrent manner and due to which the outcome depends on the particular order in which the access of data takes place.

Mainly this condition is a situation that may occur inside the **critical section**. Race condition in the critical section happens when the result of multiple thread execution differs according to the order in which the threads execute. But this condition in critical sections can be avoided if the critical section is treated as an atomic instruction. Proper thread synchronization using locks or atomic variables can also prevent race conditions.



Critical section

A section of code or a set of operations, in which process may be changing shared variables, updating a common file or a table etc. is known as the critical section of that process. A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, **only one process must be executing its critical section**. If any other process also wants to execute its critical section, it must wait until the first one finishes. The entry to the critical section is mainly handled by `wait()` function while the exit from the critical section is controlled by the `signal()` function.



Entry Section

In this section mainly the process requests for its entry in the critical section.

Exit Section

This section is followed by the critical section.

The solution to the Critical Section Problem

A solution to the critical section problem must satisfy the following four conditions:

1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time. No two contending processes should be simultaneously executing inside their CS.

2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting

No process should have to wait forever to enter its CS. After a process makes a request for getting into its critical section, there is a limit for how many other processes

can get into their critical section, before this process's request is granted. So after the limit is reached, the system must grant the process permission to get into its critical section.

4. No assumption

No assumption should be made about relative speeds and priorities of contenting processes.

Semaphores

Semaphore is simply a variable that is **non-negative** and shared between threads. It is another algorithm or solution to the critical section problem. It is a signaling mechanism and a thread that is waiting on a semaphore, which can be signaled by another thread.

A Semaphore uses two atomic operations, 1)wait(), and 2) signal ()for the process synchronization.

A Semaphore is an integer variable, which can be accessed only through two operations *wait()* and *signal()*.

Example

```
WAIT ( S ) :  
{while ( S <= 0 );  
//no-op  
S = S - 1;  
}
```

The definition of signal () as follows: SIGNAL (S):

There are two types of semaphores:

1. **Binary Semaphores or Mutex lock**
2. **Counting Semaphores**

- **Binary Semaphores:** They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.

- **Counting Semaphores:** They can have any value and are not restricted over a certain domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

We can use binary semaphores to deal with the critical section problem for multiple processes. The n processes share a semaphore, mutex, initialized to 1. Each process P_i is organised as follows

```

Do {
Wait (mutex);
    // critical section
Signal (mutex);
    // remainder section
} While (TRUE);

```

Section 2.5 Principle of Concurrency

Concurrency is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads running in parallel. The running process threads always communicate with each other through shared memory or message passing.

Both interleaved and overlapped processes can be viewed as examples of concurrent processes, they both present the same problems. The relative speed of execution cannot be predicted. It depends on the following:

- The activities of other processes
- The way operating system handles interrupts
- The scheduling policies of the operating system

Problems in Concurrency:

- **Sharing global resources** – Sharing of global resources safely is difficult. If two processes both make use of a global variable and both perform read and write on that variable, then the order in which various read and write are executed is critical.
- **Optimal allocation of resources** – It is difficult for the operating system to manage the allocation of resources optimally.

- **Locating programming errors** –
It is very difficult to locate a programming error because reports are usually not reproducible.
- **Locking the channel** –
It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.

Advantages of Concurrency:

- **Running of multiple applications** –
It enable to run multiple applications at the same time.
- **Better resource utilization** –
It enables that the resources that are unused by one application can be used for other applications.
- **Better average response time** –
Without concurrency, each application has to be run to completion before the next one can be run.
- **Better performance** –
It enables the better performance by the operating system. When one application uses only the processor and another application uses only the disk drive then the time to run both applications concurrently to completion will be shorter than the time to run each application consecutively.

Drawbacks of Concurrency :

- It is required to protect multiple applications from one another.
- It is required to coordinate multiple applications through additional mechanisms.
- Additional performance overheads and complexities in operating systems are required for switching among applications.
- Sometimes running too many applications concurrently leads to severely degraded performance.

3.0 MEMORY MANAGEMENT

INTRODUCTION

Memory Management is the process of controlling and coordinating **computer memory**, assigning portions known as blocks to various running programs to optimize the overall performance of the system. It is the most important function of an operating system that manages primary memory.

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

To summarize the functions of memory management as follows:

- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also makes sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

Logical and Physical Address

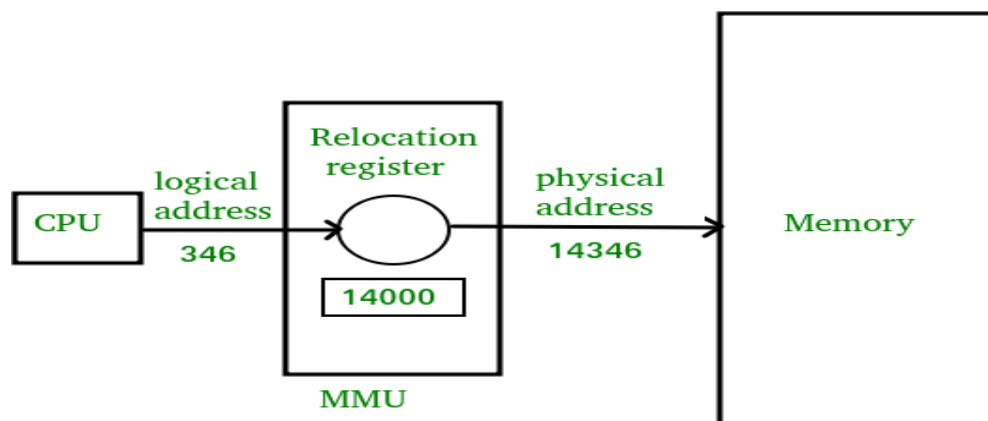
The **physical address** refers to a location in the memory. It allows access to data in the main memory. A physical address is not directly accessible to the user program hence, a logical address needs to be mapped to it to make the address accessible. This mapping is done by the **MMU**. Memory Management Unit (MMU) is a hardware component responsible for translating a logical address to a physical address.

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, **the logical address must be mapped to the physical address by MMU (Memory Management Unit) before they are**

used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

A **logical address or virtual address** is an address that is generated by the CPU during program execution. A logical address doesn't exist physically. The logical address is used as a reference to access the physical address. A logical address usually ranges from zero to maximum (max). The user program that generates the logical address assumes that the process runs on locations between 0 to the max. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective. This **logical address**(generated by CPU) combines with the **base address** generated by the MMU to form the **physical address**.

The diagram below explains how the mapping between logical and physical addresses is done.



(Fig Logical vs physical address space)

1. The CPU generates the logical address (here, 346).
2. The MMU will generate the base address (here, 14000) which is stored in the Relocation Register.
3. The value of Relocation Register (here, 14000) is added to the logical address to get the physical address. i.e. $14000+346= 14346$ (Physical Address).

Differences Between Logical and Physical Address in Operating System

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.

3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differ from each other in run-time address binding method.
5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

3.1 Memory allocation techniques

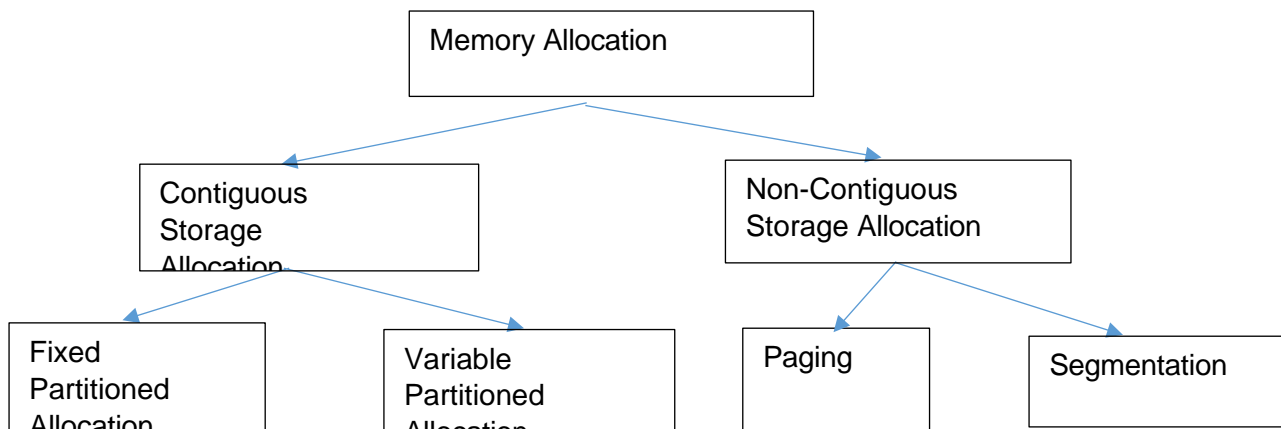
The two fundamental **methods** of **memory allocation**

- (a) static **memory allocation**.
- (b) Dynamic **memory allocation**

Static **method** assigns the **memory** to a process, before its execution. On the other hand, the dynamic **memory allocation method** assigns the **memory** to a process, during its execution.

Memory allocation are of two types

- A. Contiguous storage allocation
- B. Non-Contiguous storage allocation



A. Contiguous memory allocation

In the Contiguous Memory Allocation, each process is contained in a single contiguous section of memory. In this memory allocation, all the available memory space remains together in one place which implies that the freely available memory partitions are not spread over here and there across the whole memory space.

In **Contiguous memory allocation** which is a memory management technique, whenever there is a request by the user process for the memory then a single section of the contiguous memory block is given to that process according to its requirement. Contiguous Memory allocation is achieved just by dividing the memory into blocks of different sizes to accommodate programs. Partitioning is of two types

- (a) Fixed partition allocation
- (b) Variable partition allocation

(A) **Fixed Partition Allocation Scheme**

In a multiprogramming environment, several programs reside in primary memory at a time and the CPU passes its control rapidly between these programs. There are two types of partitioning when partitions are created.

- (i) Static Partitioning
- (ii) Dynamic Partitioning

In **Static partitioning** scheme, the system divides the memory into fixed-size partitions. The partitions may or may not be the same size. The size of each partition is fixed as indicated by the name of the technique and it cannot be changed.

In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.

Whenever any process terminates then the partition becomes available for another process. It is important to note that these partitions are allocated to the processes as they arrive and the partition that is allocated to the arrived process basically depends on the algorithm followed.

Example: In fixed size partition the memory is divided into 6 partitions. The 1st region is reserved for OS. The rest 5 regions are for user programs. Three partitions are occupied by P1, P2 and P3. The second and last partitions are free

Lower memory area	Operating System (200K)	0
	FREE (200K)	200
	P ₁ (200K)	400
	P ₂ (300K)	600
	P ₃ (100K)	900
	FREE	1000

Fixed size partitions

Once the partitions are defined, the OS keeps track of status of the memory partitions and this is done through a data structure called **partition description table (PDT)**

Partition number	Starting Address of Partition	Size of partition	Partition status
1	0K	200K	Allocated
2	200K	200K	Free
3	400K	200K	Allocated
4	600K	300K	Allocated
5	900K	100K	Allocated
6	1000K	100K	Free

There are 03 most common strategies to allocate free partitions to the new processes.

- (i) First-fit
- (j) Best fit
- (k) Worst fit

In **Dynamic partitioning** scheme, the size and the no. of partitions are decided during the run time by the operating system.

Advantages of Fixed-size Partition Scheme

1. This scheme is simple and is easy to implement
2. It supports multiprogramming as multiple processes can be stored inside the main memory.
3. Management is easy using this scheme.
4. It requires no special costly hardware.
5. It makes efficient utilisation of processor and I/O devices.

Disadvantages of Fixed-size Partition Scheme

1. Internal Fragmentation

Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as Internal fragmentation

2. Limitation on the size of the process

If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory. Due to this, a condition is imposed on the size of the process and it is: the size of the process cannot be larger than the size of the largest partition.

3. External Fragmentation

It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

4. Degree of multiprogramming is less

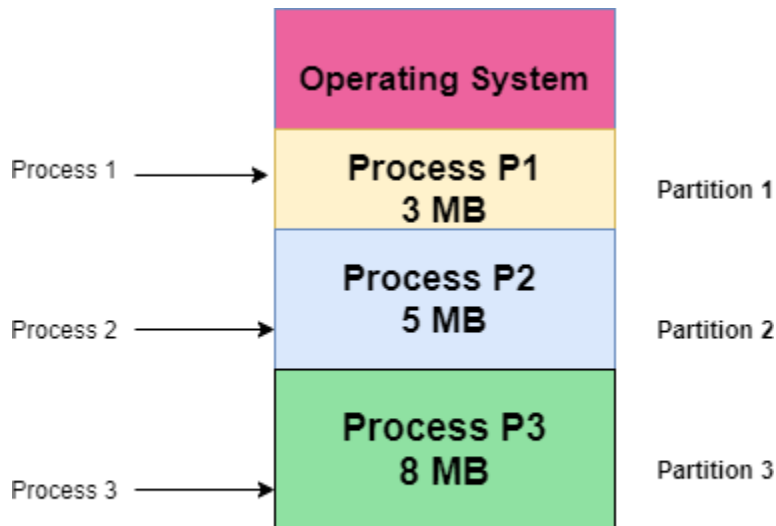
In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.

(B) Variable Partition Allocation Scheme

This scheme is also known as **Dynamic partitioning** and is came into existence to overcome the drawback i.e internal fragmentation that is caused by **Static partitioning**. In this partitioning, scheme allocation is done dynamically.

The size of the partition is not declared initially. Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process. Thus the size of each partition is equal to the size of the process.

As partition size varies according to the need of the process so in this partition scheme there is no **internal fragmentation**.



Size of Partition = Size of Process

Advantages of Variable-size Partition Scheme

Some Advantages of using this partition scheme are as follows:

1. **No Internal Fragmentation** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
2. **Degree of Multiprogramming is Dynamic** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
3. **No Limitation on the Size of Process** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

Disadvantages of Variable-size Partition Scheme

Some Disadvantages of using this partition scheme are as follows:

1. **External Fragmentation** As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation. Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let's there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that

process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.

2. **Difficult Implementation** The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration. As we know that OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.

B. Non-Contiguous memory allocation

In the **non-contiguous memory allocation** the available free memory space are scattered here and there and all the free memory space is not at one place. So this is time-consuming. In the **non-contiguous memory allocation**, a process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement. This technique of **non-contiguous memory allocation** reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.

Non-contiguous memory allocation is of different types,

1. Paging
2. Segmentation
3. Segmentation with paging

i) Paging

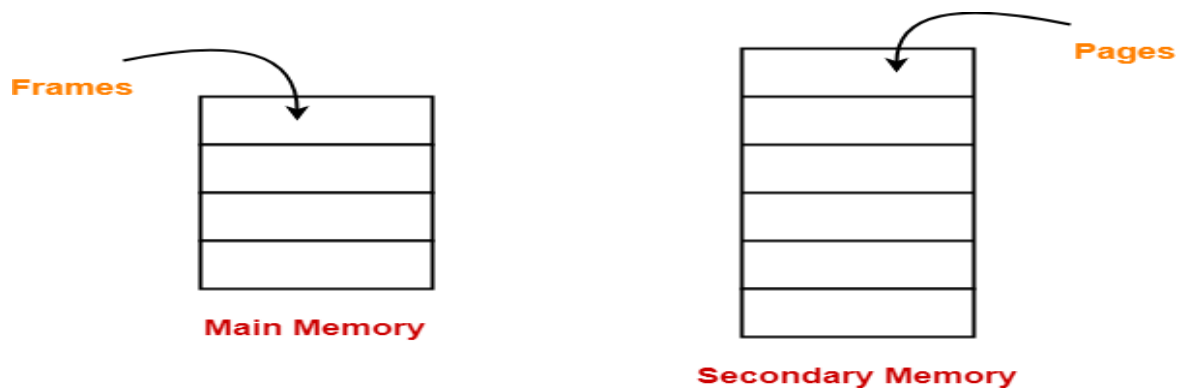
A non-contiguous policy with a fixed size partition is called paging. Paging is a memory management technique that permits a program's memory to be non-contiguous into the physical memory and thereby allowing program to be allocated physical memory whenever it is possible.

A computer can address more memory than the amount of physically installed on the system. These program generated address are called as logical or virtual address and they form the virtual address space. This extra memory is actually called virtual memory. Paging technique is very important in implementing virtual memory.

Secondary memory is divided into equal size partition (fixed) called **pages**. Every process will have a separate page table. The entries in the page table are the number of pages a process.

The physical memory is conceptually divided into no. of fixed size blocks called as **frames or page frames**. At each entry either we have an invalid pointer which means the page is not in main memory or we will get the corresponding frame number. When the frame number is combined with instruction of set D than we will get the corresponding physical address. Size of a page table is generally very large so cannot be accommodated inside the PCB, therefore, PCB contains a register value PTBR(page table base register) which leads to the page table.

- **Paging is a fixed size partitioning scheme.**
- **In paging, secondary memory and main memory are divided into equal fixed size partitions.**
- **The partitions of secondary memory are called as pages.**
- **The partitions of main memory are called as frames.**



- **Each process is divided into parts where size of each part is same as page size.**
- **The size of the last part may be less than the page size.**
- **The pages of process are stored in the frames of main memory depending upon their availability.**

Example-

- **Consider a process is divided into 4 pages P_0 , P_1 , P_2 and P_3 .**
- **Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-**

P1
P3
P0
P2

Main Memory

Translating Logical Address into Physical Address-

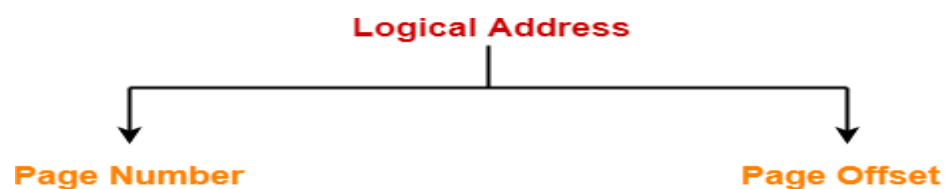
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Page Number
2. Page Offset



- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

Step-02:

For the page number generated by the CPU,

- **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

Step-03:

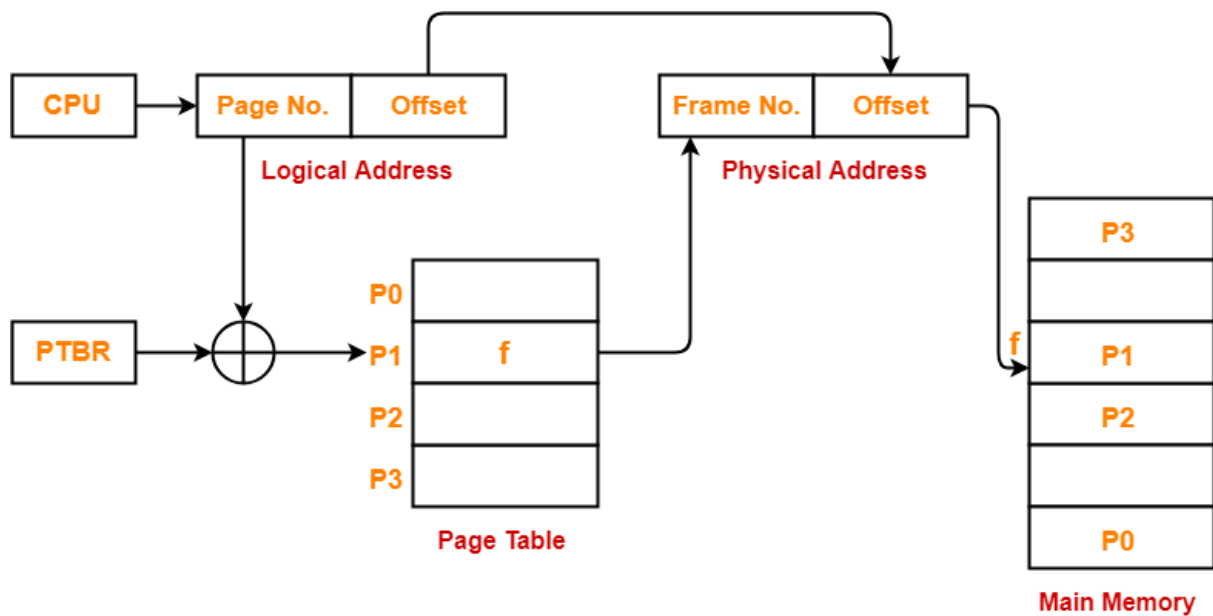
- The frame number combined with the page offset forms the required physical address.



- Frame number specifies the specific frame where the required page is stored.
- Page Offset specifies the specific word that has to be read from that page.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages:

1. It is independent of external fragmentation.
2. It supports time sharing system.
3. It achieves a high degree of Multiprogramming.

Disadvantages:

1. It makes the translation very slow as main memory access two times.
2. A page table is a burden over the system which occupies considerable space.

ii) Segmentation

Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process. The details about each segment are stored in a table called as segment table.

Segmentation is a programmer view of the memory where instead of dividing a process into equal size partition we divided according to program into partition called segments. A segment can be defined as a logical grouping of instructions such as Subroutine, array or data area. Every program is a collection of these segments.

Segmentation is a memory management scheme which supports the programmer's view in memory. Programmers never think of their programs as a linear array of words. Rather they think of their programs as a collection of logically related entities such as subroutine, procedures, functions, global or local data area, stack etc.

It is better to have segmentation which divides the process into the segments. Each segment contains same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

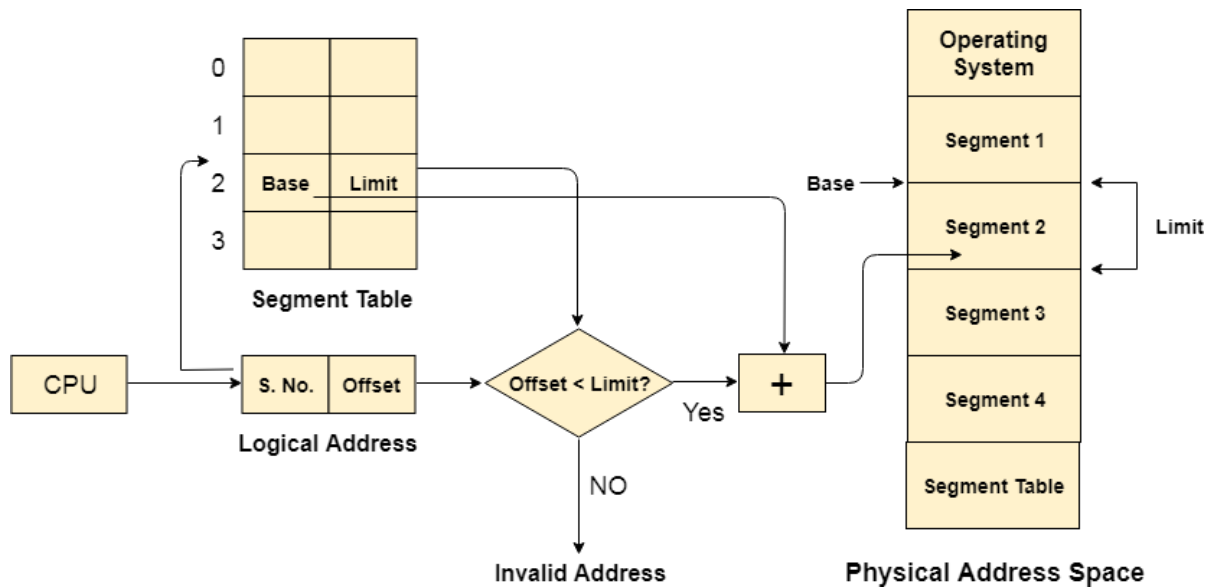
The translation is the same as paging but paging segmentation is independent of internal fragmentation but suffers from external fragmentation. Reason of external fragmentation is program can be divided into segments but segment must be contiguous in nature.

CPU generates a logical/virtual address which contains two parts:

1. Segment Number
2. Offset

The Segment number is mapped to the segment table. A segment table is used by the MMU. It contains an entry for each segment of a process. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.



Advantages of Segmentation

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compare to the page table in paging.

Disadvantages

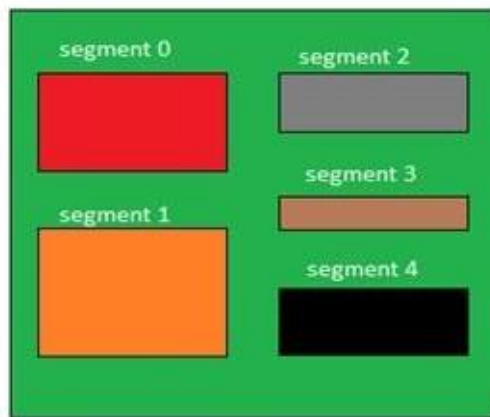
1. It can have external fragmentation.
2. it is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

iii) Segmentation with paging

In segmentation with paging, we take advantages of both segmentation as well as paging. It is a kind of multilevel paging but in multilevel paging, we divide a page table into equal size partition but here in segmentation with paging, we divide it according to segments. All the properties are the same as that of paging because segments are divided into pages. Each segment in this scheme is divided into pages and each segment maintains a page table. The logical address is divided into 3 parts

- i. Segment no S
- ii. Page number P
- iii. Offset or displacement D

Logical View of Segmentation

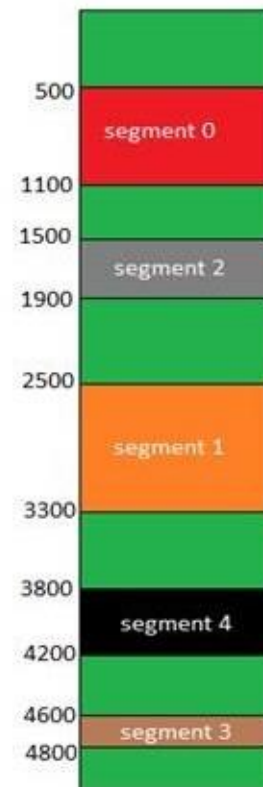


Logical Address Space

Segment Number

	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table



Physical Address Space

3.2 Swapping

Swapping is a mechanism in which a process can be **swapped** temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Swap space is the portion of virtual memory that is on the hard disk, **used when** RAM is full. **Swap** space can be useful to computer in various ways: It can be **used as** a single contiguous memory which reduces i/o operations to read or write a file. Applications which are not **used or** are **used** less can be kept in **swap** file.

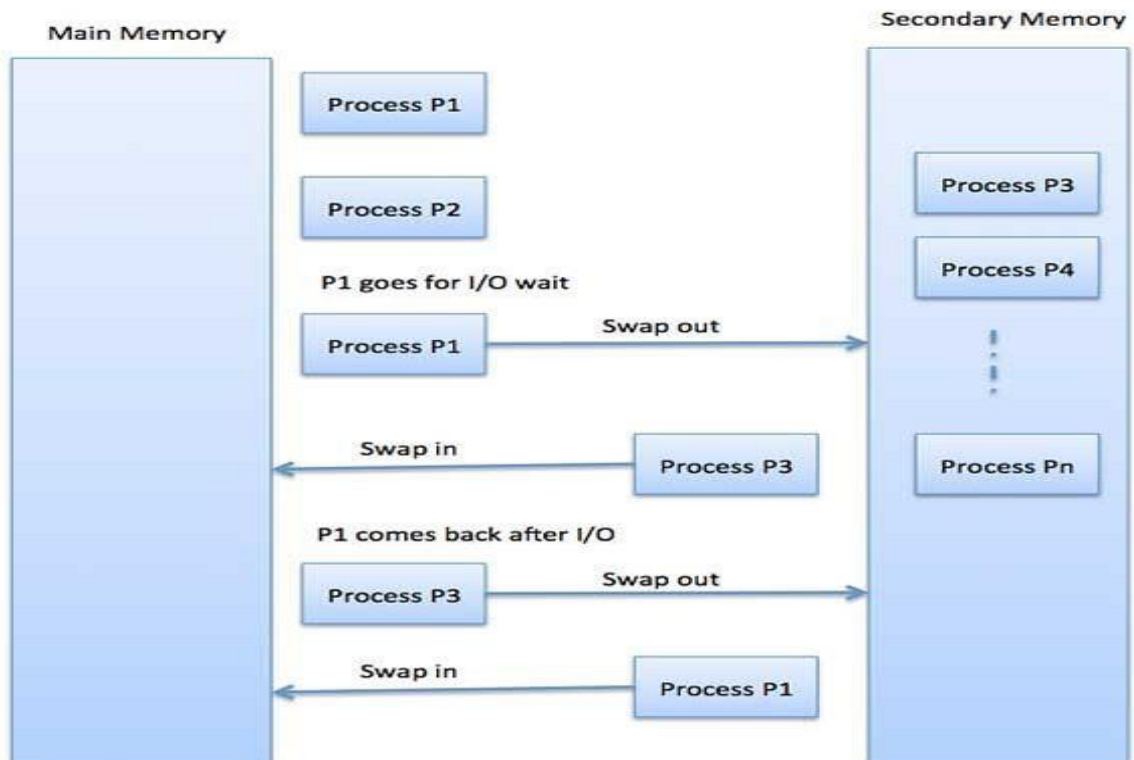
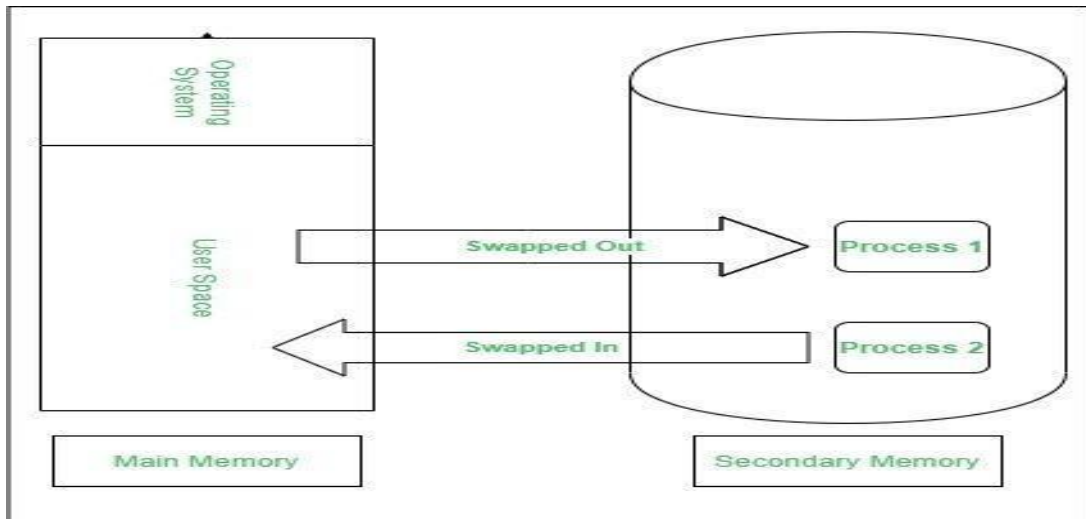
As additional RAM is required, the state of the physical **memory** page is mapped to the **swap** space, enabling a form of virtual (non-physical RAM) **memory** capacity. In other words, the main purpose of **swapping** in **memory management** is to enable more usable **memory** than held by the computer hardware.

The concept of swapping has divided into two more concepts: Swap-in and Swap-out.

- Swap-out is a method of removing a process from RAM and adding it to the hard disk.

- Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.



Advantages of Swapping

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

Dis-Advantages of Swapping

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

3.3 Virtual Memory

Virtual memory is an area of a computer system's secondary memory storage space (such as a [hard disk](#) or [solid state drive](#)) which acts as if it were a part of the system's [RAM](#) or primary memory.

Virtual memory is a feature of an operating system that enables a computer to be able to compensate shortages of physical **memory** by transferring pages of data from random access **memory** to disk storage. This process is done temporarily and is designed to work as a combination of **RAM** and space on the hard disk.

Virtual memory is an area of a computer system's secondary **memory** storage space (such as a hard disk) which acts as if **it** were a part of **the** system's **RAM** or primary **memory**. This frees up space in **RAM**, which can then be used to accommodate data which **the** system needs to access imminently.

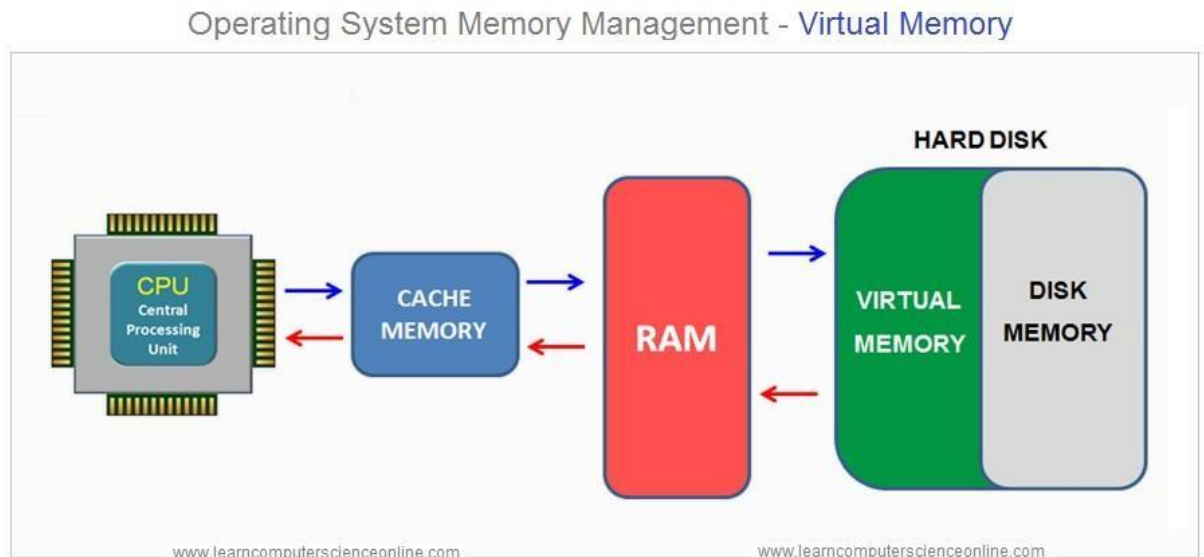
Virtual Memory is a storage allocation scheme in which secondary **memory** can be addressed as though it were part of main **memory**. It maps **memory** addresses used by a program, called **virtual** addresses, into physical addresses in computer **memory**.

Need for Virtual Memory

- Virtual memory was developed when physical RAM was very expensive, and RAM is still more expensive per Gigabyte than storage media such as [hard disks and solid-state drives](#). For that reason, it is much less costly to use a

combination of physical RAM and virtual memory than to equip a computer system with more RAM.

- Since using virtual memory (or increasing virtual memory) has no extra financial cost (because it uses existing storage space) it offers a way for a computer to use more memory than is physically available on the system.



Types of virtual memory: Paging and Segmentation

Virtual memory can be managed in a number of different ways by a system's operating system, and the two most common approaches are paging and segmentation.

1. Virtual Memory Paging

In a system which uses paging, RAM is divided into a number of blocks – usually 4k in size – called pages. Processes are then allocated just enough pages to meet their memory requirements. That means that there will always be a small amount of memory wasted, except in the unusual case where a process requires exactly a whole number of pages.

During the normal course of operations, pages (i.e. memory blocks of 4K in size) are swapped between RAM and a page file, which represents the virtual memory.

2. Virtual Memory Segmentation

Segmentation is an alternative approach to memory management, where instead of pages of a fixed size, processes are allocated segments of differing length to exactly meet their requirements. That means that unlike in a paged system, no memory is wasted in a segment.

Segmentation also allows applications to be split up into logically independent address spaces, which can make them easier to share, and more secure.

Advantages of Virtual Memory

- Allows more applications to be run at the same time.
- Allows larger applications to run in systems that do not have enough physical RAM alone to run them.
- Provides a way to increase memory which is less costly than buying more RAM.
- Provides a way to increase memory in a system which has the maximum amount of RAM that its hardware and operating system can support.

Disadvantages of Virtual Memory

- Does not offer the same performance as RAM.
- Can negatively affect the overall performance of a system.
- Takes up storage space which could otherwise be used for long term data storage.

3.4 Demand paging

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required.

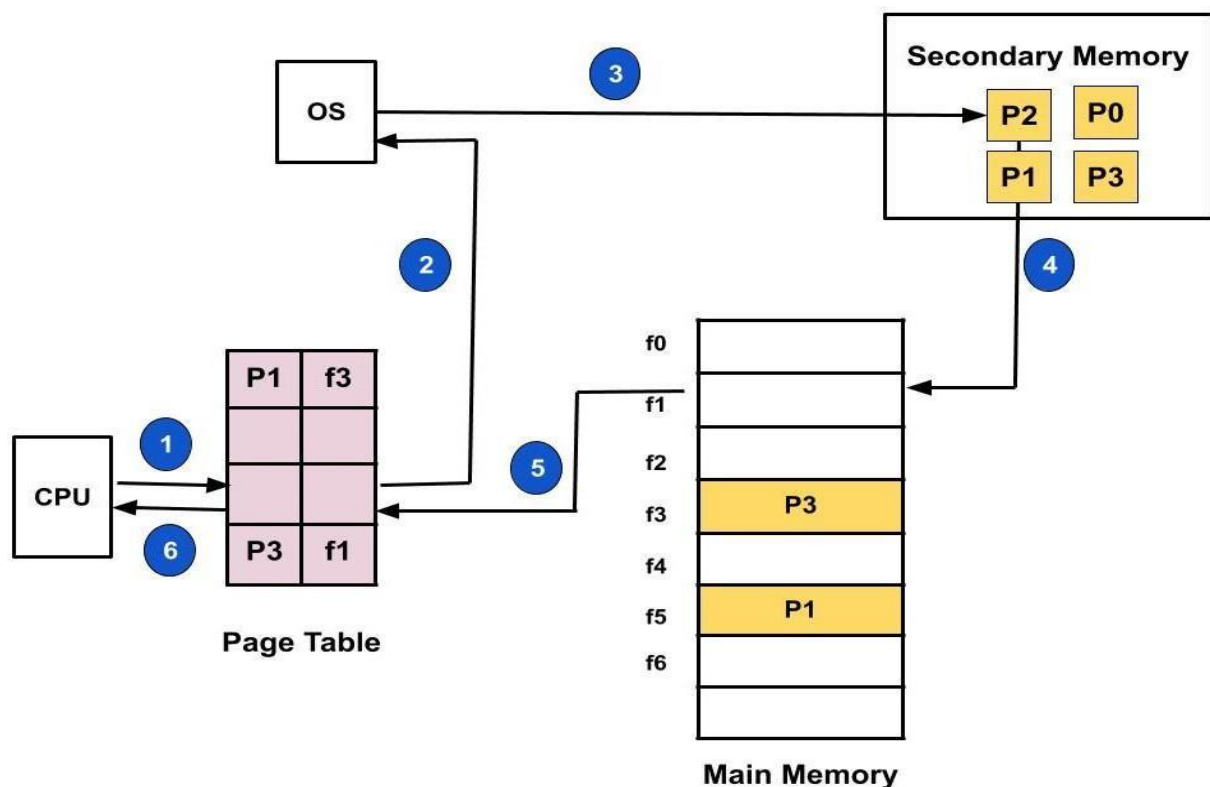
Demand paging is a technique used in virtual memory systems where the pages are brought in the main memory only when required or demanded by the CPU.

A **page fault** occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

Page fault is a type of exception raised by computer hardware when a running program accesses a memory page that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process. Logically, the page may be accessible to the process, but requires a mapping to be added to the process page tables, and may additionally require the actual page contents to be loaded from a backing store such as a disk.

How does demand paging work?

Lets us understand this with the help of an example. Suppose we have to execute a process P having four pages as P0, P1, P2, and P3. Currently, in the page table, we have page P1 and P3.



1. Now, if the CPU wants to access page P2 of a process P, first it will search the page in the page table.
2. As the page table does not contain this page so it will be a **trap** or **page fault**. As soon as the trap is generated and context switching happens and the control goes to the operating system.

3. The OS system will put the process in a waiting/ blocked state. The OS system will now search that page in the backing store or secondary memory.
4. The OS will then read the page from the backing store and load it to the main memory.
5. Next, the OS system will update the page table entry accordingly.
6. Finally, the control is taken back from the OS and the execution of the process is resumed.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Page Fault Service time

So whenever a page fault occurs all the above steps(2–6) are performed. This time taken to service the page fault is called the **Page fault service time**.

Effective Memory Access time

When the page fault rate is '**p**' while executing any process then the effective memory access time is calculated as follows:

$$\text{Effective Memory Access time} = (p) * (s) + (1-p) * (m)$$

where p is the page fault rate.

s is the page fault service time.

m is the main memory access time.

Page Replacement Algorithms

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

There are three Page Replacement Algorithms:

1. First In First Out (FIFO) Algorithm
2. Optimal page replacement Algorithm
3. Least recently used (LRU) Algorithm

1. First In First Out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. Here Replace a page that page is the oldest page of all the pages of main memory. When a page needs to be replaced page in the front of the queue is selected for removal.

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find number of page faults.

Page reference 1, 3, 0, 3, 5, 6, 3

1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots → 3 Page Faults.
 when 3 comes, it is already in memory so → 0 Page Faults.
 Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. → 1 Page Fault.
 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 → 1 Page Fault.
 Finally when 3 come it is not available so it replaces 0 1-page fault

Page fault rate = No. of page faults / No. of bits in the reference string.
 $= 6/7$

Example 2:

The page reference string: 0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7 for a memory with 3 frames.

Table FIFO Behaviour

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	3*	3	3	2*	2	2	1*	1	1	4*	4	4	7*
1		1*	1	1	0*	0	0	3*	3	3	2*	2	2	5*	5	5
2			2*	2	2	1*	1	1	0*	0	0	3*	3	3	6*	6
	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

It has 16-page faults. The symbol "*" indicates the new page in the memory. The symbol "m" indicate page fault. Page fault ratio = 16/16=100%

2. Optimal Page replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. Here Replace a page that will not be used for longest period of time

Example-2:

Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

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

Miss Miss Miss Miss Hit Miss Hit Miss Hit Hit Hit Hit Hit Hit Hit

Total Page Fault = 6

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults

0 is already there so → 0 Page fault.

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future. → 1 Page fault.

0 is already there so → 0 Page fault..

4 will takes place of 1 → 1 Page Fault.

Now for the further page reference string → 0 Page fault because they are

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

Page fault rate = No. of page faults / No. of bits in the reference string.
= 6/14

Example-2

The page reference string: 1 2 3 2 5 6 3 4 6 3 7 3 1 5 3 6 3 4 2 4 3 4 5 1 for a memory with 3 frames.

Table FIFO Behaviour

Frame	1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
0	* 1	1	1	1	1	1	1	1	1	1	1	1	* 1	1	1	1	1	1	* 2	2	2	2	2	* 1
1		*2	2	* 2	2	* 6	6	6	* 6	6	6	6	6	6	6	* 6	6	* 4	4	* 4	4	* 4	4	4
2			* 3	3	3	* 3	3	3	* 3	3	* 3	3	3	* 3	3	* 3	3	3	3	* 3	3	3	3	3
3				* 5	5	5	* 4	4	4	* 7	7	7	* 5	5	5	5	5	5	5	5	5	5	* 5	5
	m	m	m	h	m	m	h	m	h	h	m	h	h	m	h	h	h	m	m	h	h	h	h	m

Here “m” is for miss and “h” for hit. Further “*” indicates a new page in memory.
 Page fault = 11/24

3. Least Recently Used
 In this algorithm page will be replaced which is least recently used. Replace a page that has not been used for the longest period of time.

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.

Page reference: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3
 No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.
 Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots — **> 4**
Page faults
 0 is already their so —> **0** **Page fault.**

when 3 came it will take the place of 7 because it is least recently used → 1

Page fault

0 is already in memory so → 0 **Page fault.**

4 will take place of 1 → 1 **Page Fault**

Now for the further page reference string → 0 **Page fault** because they are already available in the memory.

Page fault rate = No. of page faults / No. of bits in the reference string.
 = 6/14

Example 2:

The page reference string: 0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7 for a memory with 3 frames.

Table Behaviour

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	3*	3	3	2*	2	2	1*	1	1	4*	4	4	7*
1		1*	1	1	0*	0	0	3*	3	3	2*	2	2	5*	5	5
2			2*	2	2	1*	1	1	0*	0	0	3*	3	3	6*	6
	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

Page fault 16/16=100%

3.0 MEMORY MANAGEMENT

INTRODUCTION

Memory Management is the process of controlling and coordinating **computer memory**, assigning portions known as blocks to various running programs to optimize the overall performance of the system. It is the most important function of an operating system that manages primary memory.

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

To summarize the functions of memory management as follows:

- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also makes sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

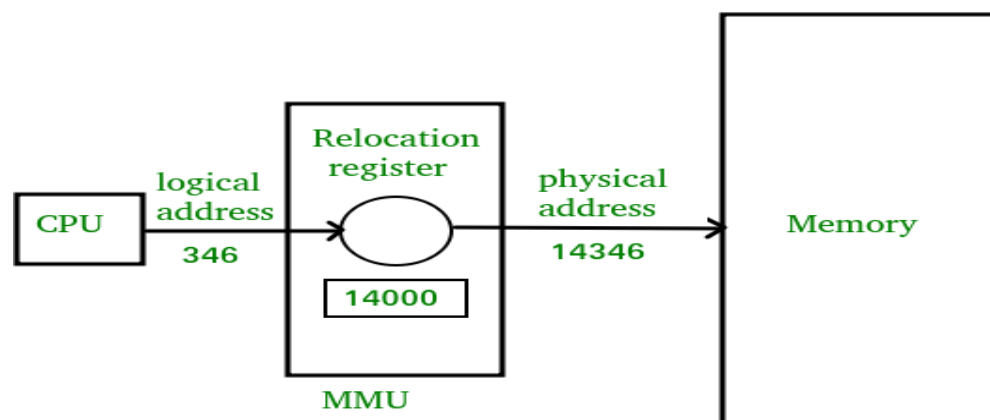
Logical and Physical Address

The **physical address** refers to a location in the memory. It allows access to data in the main memory. A physical address is not directly accessible to the user program hence, a logical address needs to be mapped to it to make the address accessible. This mapping is done by the **MMU**. Memory Management Unit (MMU) is a hardware component responsible for translating a logical address to a physical address.

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, **the logical address must be mapped to the physical address by MMU (Memory Management Unit) before they are used**. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

A **logical address or virtual address** is an address that is generated by the CPU during program execution. A logical address doesn't exist physically. The logical address is used as a reference to access the physical address. A logical address usually ranges from zero to maximum (max). The user program that generates the logical address assumes that the process runs on locations between 0 to the max. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective. This **logical address**(generated by CPU) combines with the **base address** generated by the MMU to form the **physical address**.

The diagram below explains how the mapping between logical and physical addresses is done.



(Fig Logical vs physical address space)

4. The CPU generates the logical address (here, 346).
5. The MMU will generate the base address (here, 14000) which is stored in the Relocation Register.
6. The value of Relocation Register (here, 14000) is added to the logical address to get the physical address. i.e. $14000+346= 14346$ (Physical Address).

Differences Between Logical and Physical Address in Operating System

6. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
7. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
8. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.

9. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differ from each other in run-time address binding method.
10. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

3.1 Memory allocation techniques

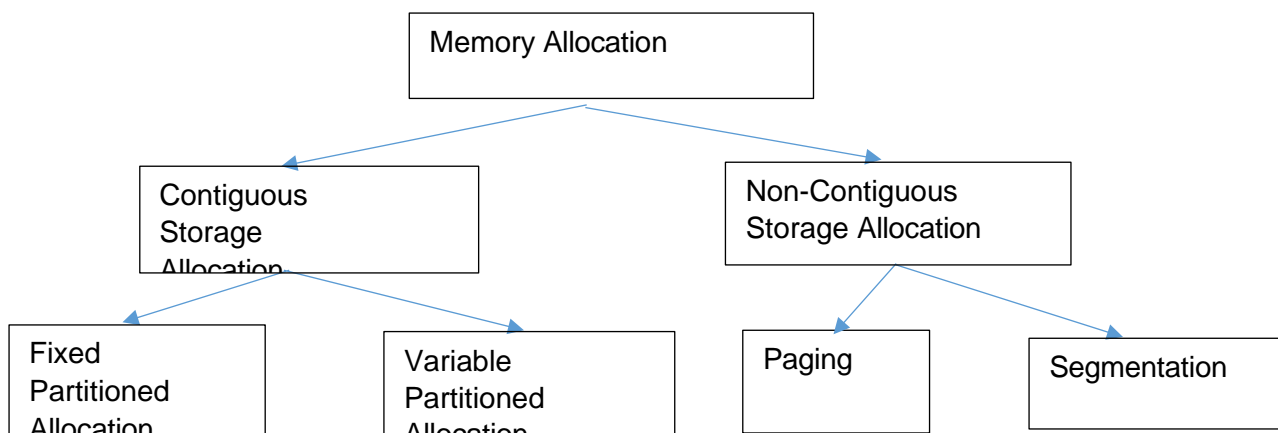
The two fundamental **methods** of **memory allocation**

- (c) static **memory allocation**.
- (d) Dynamic **memory allocation**

Static **method** assigns the **memory** to a process, before its execution. On the other hand, the dynamic **memory allocation method** assigns the **memory** to a process, during its execution.

Memory allocation are of two types

- C. Contiguous storage allocation
- D. Non-Contiguous storage allocation



A. Contiguous memory allocation

In the Contiguous Memory Allocation, each process is contained in a single contiguous section of memory. In this memory allocation, all the available memory

space remains together in one place which implies that the freely available memory partitions are not spread over here and there across the whole memory space.

In **Contiguous memory allocation** which is a memory management technique, whenever there is a request by the user process for the memory then a single section of the contiguous memory block is given to that process according to its requirement. Contiguous Memory allocation is achieved just by dividing the memory into blocks of different sizes to accommodate programs. Partitioning is of two types

- (a) Fixed partition allocation
- (b) Variable partition allocation

(A) **Fixed Partition Allocation Scheme**

In a multiprogramming environment, several programs reside in primary memory at a time and the CPU passes its control rapidly between these programs. There are two types of partitioning when partitions are created.

- (i) Static Partitioning
- (ii) Dynamic Partitioning

In **Static partitioning** scheme, the system divides the memory into fixed-size partitions. The partitions may or may not be the same size. The size of each partition is fixed as indicated by the name of the technique and it cannot be changed.

In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.

Whenever any process terminates then the partition becomes available for another process. It is important to note that these partitions are allocated to the processes as they arrive and the partition that is allocated to the arrived process basically depends on the algorithm followed.

Example: In fixed size partition the memory is divided into 6 partitions. The 1st region is reserved for OS. The rest 5 regions are for user programs. Three partitions are occupied by P1, P2 and P3. The second and last partitions are free

Lower memory area	Operating System (200K)	0
	FREE (200K)	200
	P ₁ (200K)	400
	P ₂ (300K)	600
	P ₃ (100K)	900
	FREE	1000

Fixed size partitions

Once the partitions are defined, the OS keeps track of status of the memory partitions and this is done through a data structure called **partition description table (PDT)**

Partition number	Starting Address of Partition	Size of partition	Partition status
1	0K	200K	Allocated
2	200K	200K	Free
3	400K	200K	Allocated
4	600K	300K	Allocated
5	900K	100K	Allocated
6	1000K	100K	Free

There are 03 most common strategies to allocate free partitions to the new processes.

- (l) First-fit
- (m) Best fit
- (n) Worst fit

In **Dynamic partitioning** scheme, the size and the no. of partitions are decided during the run time by the operating system.

Advantages of Fixed-size Partition Scheme

6. This scheme is simple and is easy to implement
7. It supports multiprogramming as multiple processes can be stored inside the main memory.
8. Management is easy using this scheme.
9. It requires no special costly hardware.
10. It makes efficient utilisation of processor and I/O devices.

Disadvantages of Fixed-size Partition Scheme

1. Internal Fragmentation

Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as Internal fragmentation

2. Limitation on the size of the process

If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory. Due to this, a condition is imposed on the size of the process and it is: the size of the process cannot be larger than the size of the largest partition.

3. External Fragmentation

It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

4. Degree of multiprogramming is less

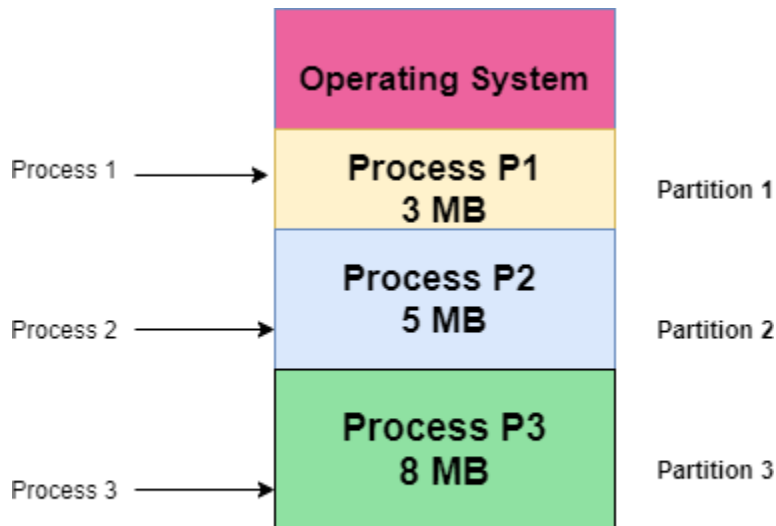
In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.

(B) Variable Partition Allocation Scheme

This scheme is also known as **Dynamic partitioning** and is came into existence to overcome the drawback i.e internal fragmentation that is caused by **Static partitioning**. In this partitioning, scheme allocation is done dynamically.

The size of the partition is not declared initially. Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process. Thus the size of each partition is equal to the size of the process.

As partition size varies according to the need of the process so in this partition scheme there is no **internal fragmentation**.



Size of Partition = Size of Process

Advantages of Variable-size Partition Scheme

Some Advantages of using this partition scheme are as follows:

4. **No Internal Fragmentation** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
5. **Degree of Multiprogramming is Dynamic** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
6. **No Limitation on the Size of Process** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

Disadvantages of Variable-size Partition Scheme

Some Disadvantages of using this partition scheme are as follows:

3. **External Fragmentation** As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation. Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let's there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that

process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.

4. **Difficult Implementation** The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration. As we know that OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.

B. Non-Contiguous memory allocation

In the **non-contiguous memory allocation** the available free memory space are scattered here and there and all the free memory space is not at one place. So this is time-consuming. In the **non-contiguous memory allocation**, a process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement. This technique of **non-contiguous memory allocation** reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.

Non-contiguous memory allocation is of different types,

4. Paging
5. Segmentation
6. Segmentation with paging

i) Paging

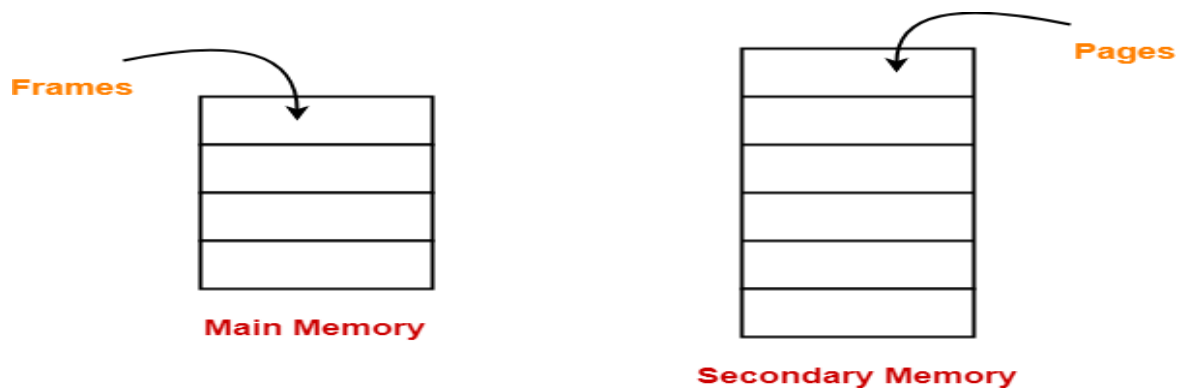
A non-contiguous policy with a fixed size partition is called paging. Paging is a memory management technique that permits a program's memory to be non-contiguous into the physical memory and thereby allowing program to be allocated physical memory whenever it is possible.

A computer can address more memory than the amount of physically installed on the system. These program generated address are called as logical or virtual address and they form the virtual address space. This extra memory is actually called virtual memory. Paging technique is very important in implementing virtual memory.

Secondary memory is divided into equal size partition (fixed) called **pages**. Every process will have a separate page table. The entries in the page table are the number of pages a process.

The physical memory is conceptually divided into no. of fixed size blocks called as **frames or page frames**. At each entry either we have an invalid pointer which means the page is not in main memory or we will get the corresponding frame number. When the frame number is combined with instruction of set D then we will get the corresponding physical address. Size of a page table is generally very large so cannot be accommodated inside the PCB, therefore, PCB contains a register value PTBR(page table base register) which leads to the page table.

- **Paging is a fixed size partitioning scheme.**
- **In paging, secondary memory and main memory are divided into equal fixed size partitions.**
- **The partitions of secondary memory are called as pages.**
- **The partitions of main memory are called as frames.**



- **Each process is divided into parts where size of each part is same as page size.**
- **The size of the last part may be less than the page size.**
- **The pages of process are stored in the frames of main memory depending upon their availability.**

Example-

- **Consider a process is divided into 4 pages P_0 , P_1 , P_2 and P_3 .**
- **Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-**

P1
P3
P0
P2

Main Memory

Translating Logical Address into Physical Address-

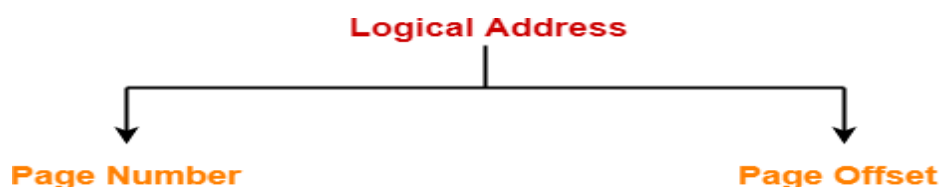
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

3. Page Number
4. Page Offset



- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

Step-02:

For the page number generated by the CPU,

- **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

Step-03:

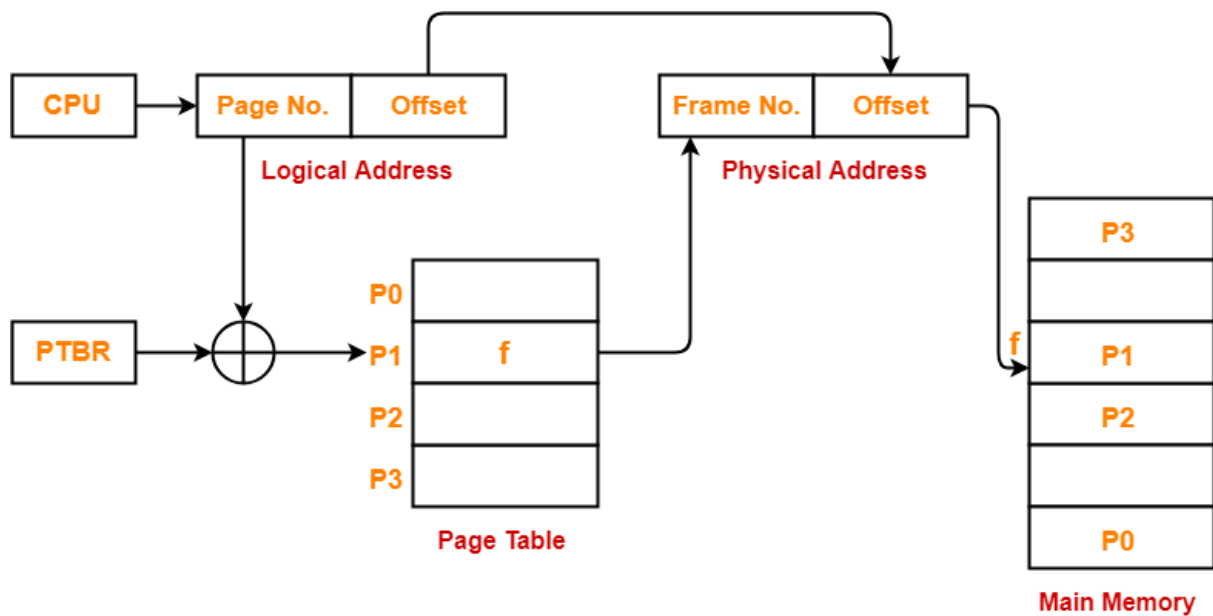
- The frame number combined with the page offset forms the required physical address.



- Frame number specifies the specific frame where the required page is stored.
- Page Offset specifies the specific word that has to be read from that page.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages:

1. It is independent of external fragmentation.
2. It supports time sharing system.
3. It achieves a high degree of Multiprogramming.

Disadvantages:

3. It makes the translation very slow as main memory access two times.
4. A page table is a burden over the system which occupies considerable space.

ii) Segmentation

Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process. The details about each segment are stored in a table called as segment table.

Segmentation is a programmer view of the memory where instead of dividing a process into equal size partition we divided according to program into partition called segments. A segment can be defined as a logical grouping of instructions such as Subroutine, array or data area. Every program is a collection of these segments.

Segmentation is a memory management scheme which supports the programmer's view in memory. Programmers never think of their programs as a linear array of words. Rather they think of their programs as a collection of logically related entities such as subroutine, procedures, functions, global or local data area, stack etc.

It is better to have segmentation which divides the process into the segments. Each segment contains same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

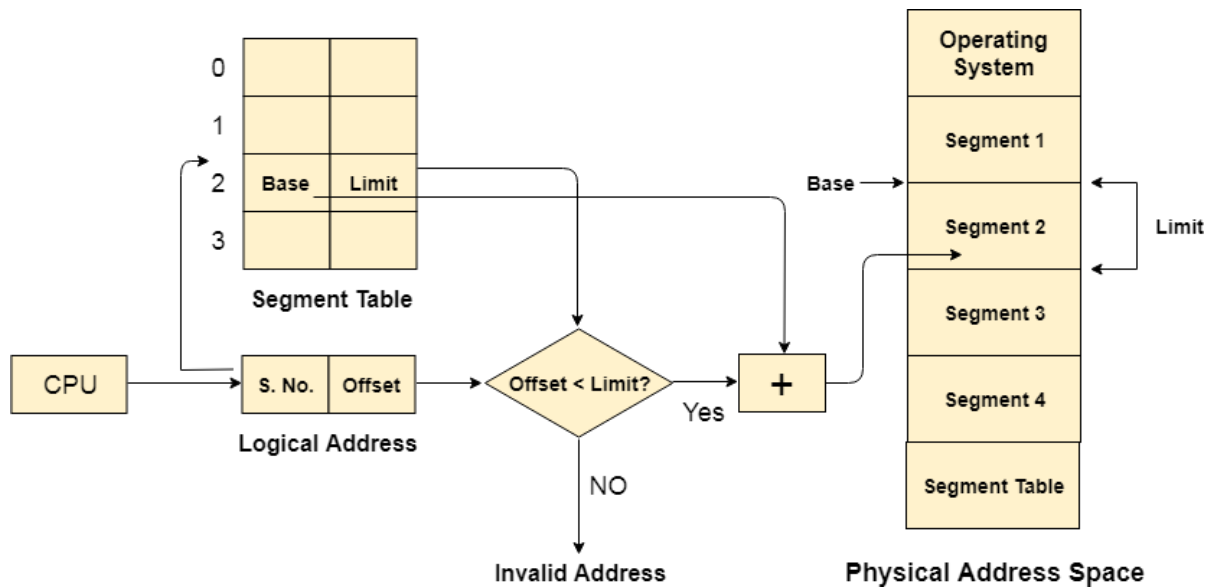
The translation is the same as paging but paging segmentation is independent of internal fragmentation but suffers from external fragmentation. Reason of external fragmentation is program can be divided into segments but segment must be contiguous in nature.

CPU generates a logical/virtual address which contains two parts:

3. Segment Number
4. Offset

The Segment number is mapped to the segment table. A segment table is used by the MMU. It contains an entry for each segment of a process. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.



Advantages of Segmentation

6. No internal fragmentation
7. Average Segment Size is larger than the actual page size.
8. Less overhead
9. It is easier to relocate segments than entire address space.
10. The segment table is of lesser size as compare to the page table in paging.

Disadvantages

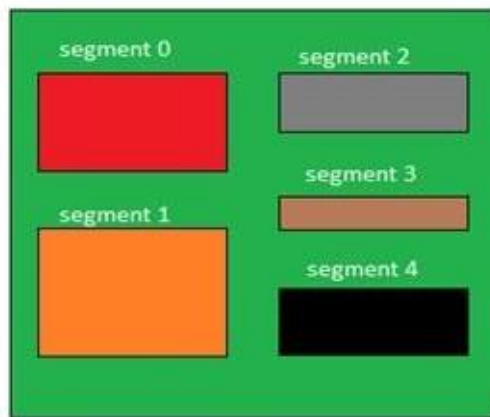
4. It can have external fragmentation.
5. it is difficult to allocate contiguous memory to variable sized partition.
6. Costly memory management algorithms.

iii) Segmentation with paging

In segmentation with paging, we take advantages of both segmentation as well as paging. It is a kind of multilevel paging but in multilevel paging, we divide a page table into equal size partition but here in segmentation with paging, we divide it according to segments. All the properties are the same as that of paging because segments are divided into pages. Each segment in this scheme is divided into pages and each segment maintains a page table. The logical address is divided into 3 parts

- i. Segment no S
- ii. Page number P
- iii. Offset or displacement D

Logical View of Segmentation

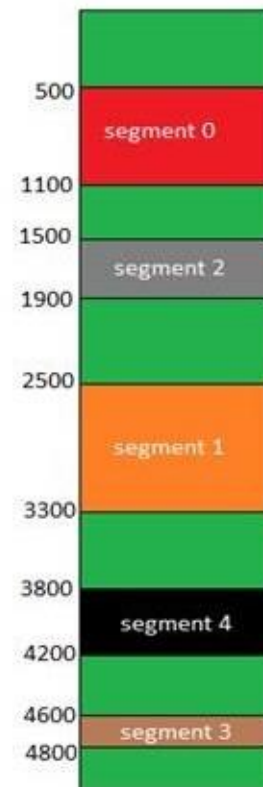


Logical Address Space

Segment Number

	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table



Physical Address Space

3.2 Swapping

Swapping is a mechanism in which a process can be **swapped** temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Swap space is the portion of virtual memory that is on the hard disk, **used when** RAM is full. **Swap** space can be useful to computer in various ways: It can be **used as** a single contiguous memory which reduces i/o operations to read or write a file. Applications which are not **used or** are **used** less can be kept in **swap** file.

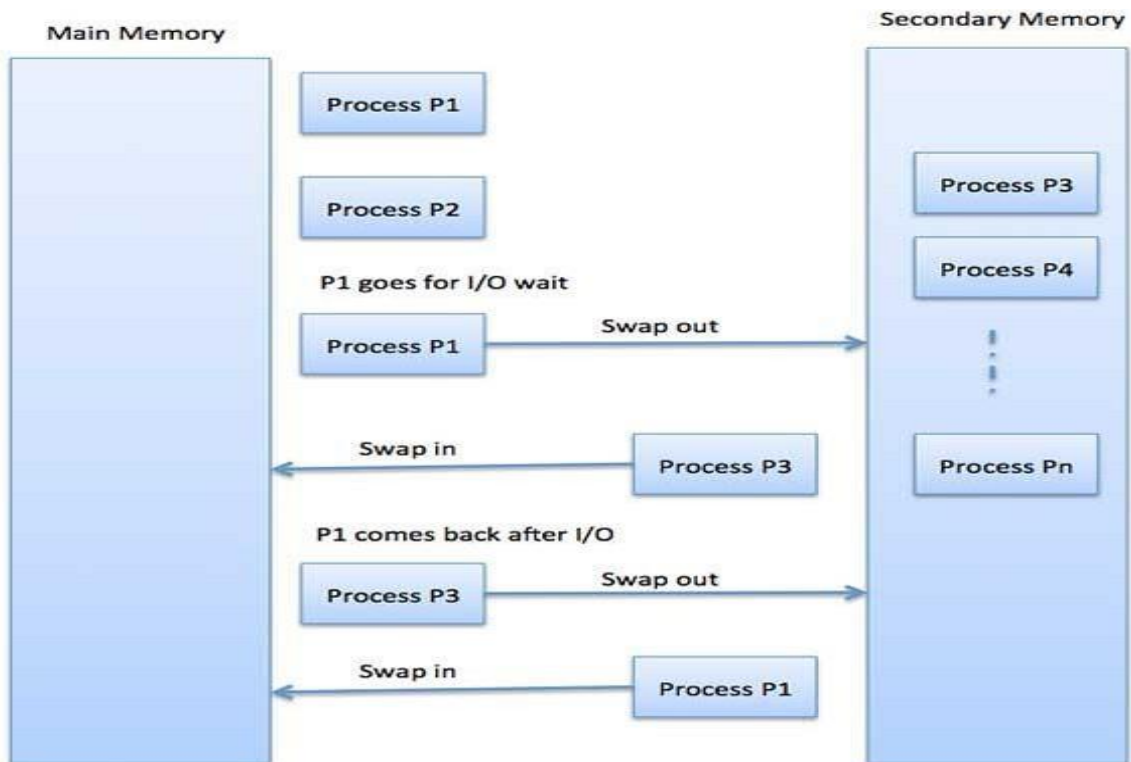
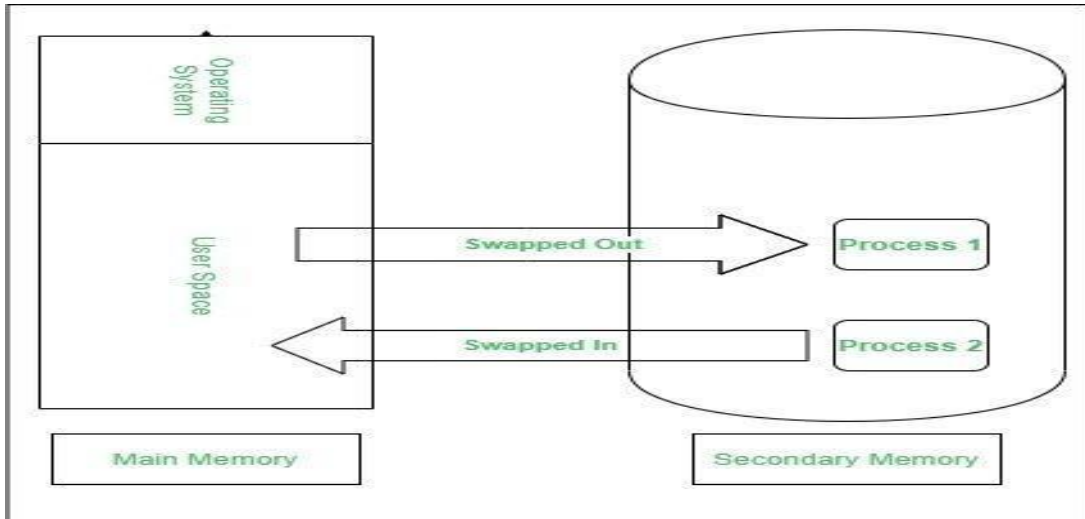
As additional RAM is required, the state of the physical **memory** page is mapped to the **swap** space, enabling a form of virtual (non-physical RAM) **memory** capacity. In other words, the main purpose of **swapping** in **memory management** is to enable more usable **memory** than held by the computer hardware.

The concept of swapping has divided into two more concepts: Swap-in and Swap-out.

- Swap-out is a method of removing a process from RAM and adding it to the hard disk.

- Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.



Advantages of Swapping

5. It helps the CPU to manage multiple processes within a single main memory.
6. It helps to create and use virtual memory.
7. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
8. It improves the main memory utilization.

Dis-Advantages of Swapping

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

3.3 Virtual Memory

Virtual memory is an area of a computer system's secondary memory storage space (such as a [hard disk](#) or [solid state drive](#)) which acts as if it were a part of the system's [RAM](#) or primary memory.

Virtual memory is a feature of an operating system that enables a computer to be able to compensate shortages of physical **memory** by transferring pages of data from random access **memory** to disk storage. This process is done temporarily and is designed to work as a combination of **RAM** and space on the hard disk.

Virtual memory is an area of a computer system's secondary **memory** storage space (such as a hard disk) which acts as if **it** were a part of **the** system's **RAM** or primary **memory**. This frees up space in **RAM**, which can then be used to accommodate data which **the** system needs to access imminently.

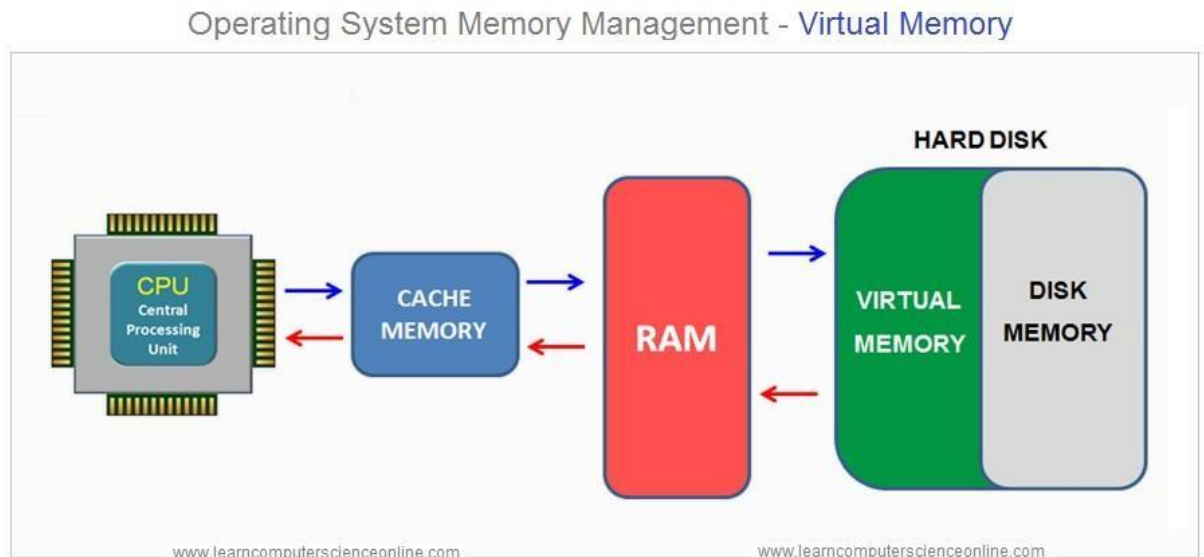
Virtual Memory is a storage allocation scheme in which secondary **memory** can be addressed as though it were part of main **memory**. It maps **memory** addresses used by a program, called **virtual** addresses, into physical addresses in computer **memory**.

Need for Virtual Memory

- Virtual memory was developed when physical RAM was very expensive, and RAM is still more expensive per Gigabyte than storage media such as [hard disks and solid-state drives](#). For that reason, it is much less costly to use a

combination of physical RAM and virtual memory than to equip a computer system with more RAM.

- Since using virtual memory (or increasing virtual memory) has no extra financial cost (because it uses existing storage space) it offers a way for a computer to use more memory than is physically available on the system.



Types of virtual memory: Paging and Segmentation

Virtual memory can be managed in a number of different ways by a system's operating system, and the two most common approaches are paging and segmentation.

3. Virtual Memory Paging

In a system which uses paging, RAM is divided into a number of blocks – usually 4k in size – called pages. Processes are then allocated just enough pages to meet their memory requirements. That means that there will always be a small amount of memory wasted, except in the unusual case where a process requires exactly a whole number of pages.

During the normal course of operations, pages (i.e. memory blocks of 4K in size) are swapped between RAM and a page file, which represents the virtual memory.

4. Virtual Memory Segmentation

Segmentation is an alternative approach to memory management, where instead of pages of a fixed size, processes are allocated segments of differing length to exactly meet their requirements. That means that unlike in a paged system, no memory is wasted in a segment.

Segmentation also allows applications to be split up into logically independent address spaces, which can make them easier to share, and more secure.

Advantages of Virtual Memory

- Allows more applications to be run at the same time.
- Allows larger applications to run in systems that do not have enough physical RAM alone to run them.
- Provides a way to increase memory which is less costly than buying more RAM.
- Provides a way to increase memory in a system which has the maximum amount of RAM that its hardware and operating system can support.

Disadvantages of Virtual Memory

- Does not offer the same performance as RAM.
- Can negatively affect the overall performance of a system.
- Takes up storage space which could otherwise be used for long term data storage.

3.4 Demand paging

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required.

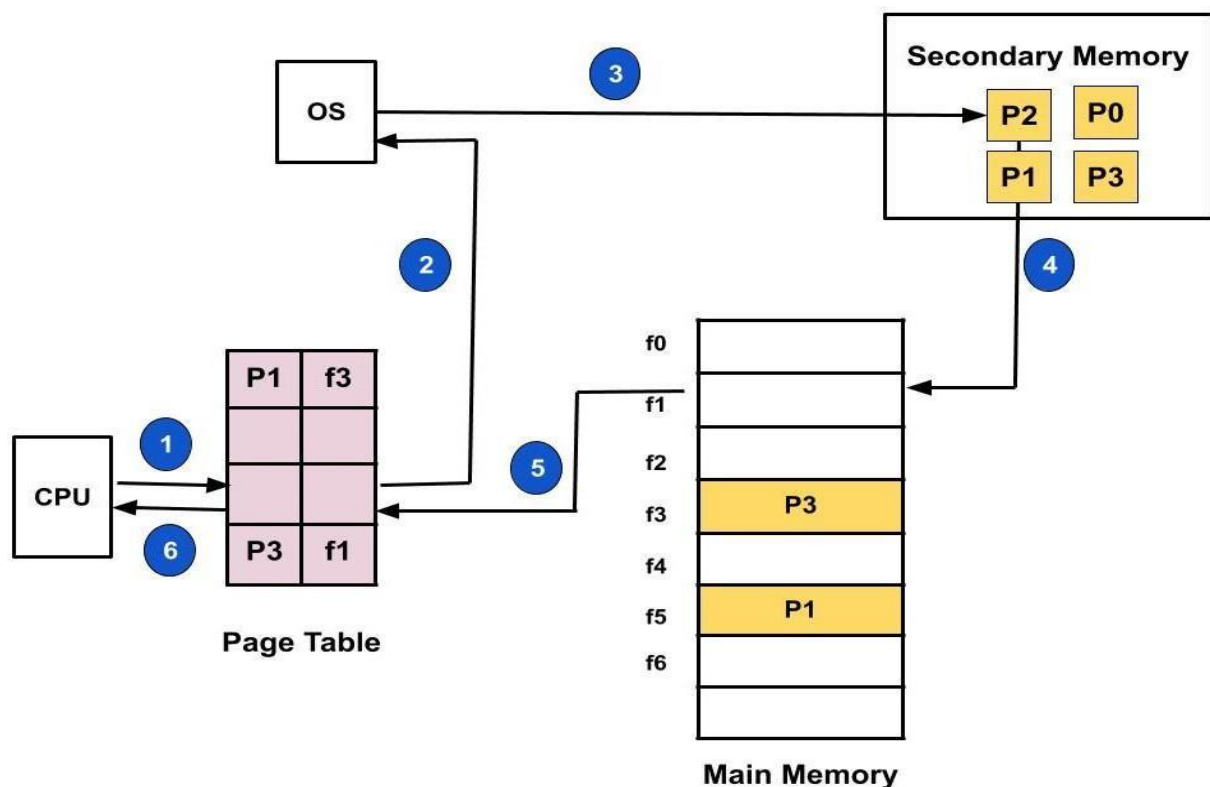
Demand paging is a technique used in virtual memory systems where the pages are brought in the main memory only when required or demanded by the CPU.

A **page fault** occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

Page fault is a type of exception raised by computer hardware when a running program accesses a memory page that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process. Logically, the page may be accessible to the process, but requires a mapping to be added to the process page tables, and may additionally require the actual page contents to be loaded from a backing store such as a disk.

How does demand paging work?

Lets us understand this with the help of an example. Suppose we have to execute a process P having four pages as P0, P1, P2, and P3. Currently, in the page table, we have page P1 and P3.



- Now, if the CPU wants to access page P2 of a process P, first it will search the page in the page table.
- As the page table does not contain this page so it will be a **trap** or **page fault**. As soon as the trap is generated and context switching happens and the control goes to the operating system.

9. The OS system will put the process in a waiting/ blocked state. The OS system will now search that page in the backing store or secondary memory.
10. The OS will then read the page from the backing store and load it to the main memory.
11. Next, the OS system will update the page table entry accordingly.
12. Finally, the control is taken back from the OS and the execution of the process is resumed.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Page Fault Service time

So whenever a page fault occurs all the above steps(2–6) are performed. This time taken to service the page fault is called the **Page fault service time**.

Effective Memory Access time

When the page fault rate is '**p**' while executing any process then the effective memory access time is calculated as follows:

$$\text{Effective Memory Access time} = (p) * (s) + (1-p) * (m)$$

where **p** is the page fault rate.

s is the page fault service time.

m is the main memory access time.

Page Replacement Algorithms

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

There are three Page Replacement Algorithms:

1. First In First Out (FIFO) Algorithm
2. Optimal page replacement Algorithm
3. Least recently used (LRU) Algorithm

1. First In First Out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. Here Replace a page that page is the oldest page of all the pages of main memory. When a page needs to be replaced page in the front of the queue is selected for removal.

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find number of page faults.

Page reference 1, 3, 0, 3, 5, 6, 3

1	3	0	3	5	6	3
		0				
	3	3	3		6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots → 3 Page Faults.
 when 3 comes, it is already in memory so → 0 Page Faults.
 Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. → 1 Page Fault.
 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 → 1 Page Fault.
 Finally when 3 come it is not available so it replaces 0 1-page fault

Page fault rate = No. of page faults / No. of bits in the reference string.
 = 6/7

Example 2:

The page reference string: 0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7 for a memory with 3 frames.

Table FIFO Behaviour

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	3*	3	3	2*	2	2	1*	1	1	4*	4	4	7*
1		1*	1	1	0*	0	0	3*	3	3	2*	2	2	5*	5	5
2			2*	2	2	1*	1	1	0*	0	0	3*	3	3	6*	6
	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

It has 16-page faults. The symbol "*" indicates the new page in the memory. The symbol "m" indicate page fault. Page fault ratio = 16/16=100%

2. Optimal Page replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. Here Replace a page that will not be used for longest period of time

Example-2:

Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

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

Miss Miss Miss Miss Hit Miss Hit Miss Hit Hit Hit Hit Hit Hit Hit

Total Page Fault = 6

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults

0 is already there so → 0 Page fault.

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future. → 1 Page fault.

0 is already there so → 0 Page fault..

4 will takes place of 1 → 1 Page Fault.

Now for the further page reference string → 0 Page fault because they are

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

Page fault rate = No. of page faults / No. of bits in the reference string.
= 6/14

Example-2

The page reference string: 1 2 3 2 5 6 3 4 6 3 7 3 1 5 3 6 3 4 2 4 3 4 5 1 for a memory with 3 frames.

Table FIFO Behaviour

Frame	1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
0	*1	1	1	1	1	1	1	1	1	1	1	1	*1	1	1	1	1	1	*2	2	2	2	2	*1
1		*2	2	*2	2	*6	6	6	*6	6	6	6	6	6	6	*6	6	*4	4	*4	4	*4	4	4
2			*3	3	3	*3	3	3	*3	3	*3	3	3	*3	3	*3	3	3	3	*3	3	3	3	3
3					*5	5	*5	4	4	*7	7	7	*5	5	5	5	5	5	5	5	5	5	*5	5
	m	m	m	h	m	m	h	m	h	h	m	h	h	m	h	h	h	m	m	h	h	h	h	m

Here “m” is for miss and “h” for hit. Further “*” indicates a new page in memory.
 Page fault = 11/24

3. Least Recently Used
 In this algorithm page will be replaced which is least recently used. Replace a page that has not been used for the longest period of time.

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.
 Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots — **> 4**
Page faults
 0 is already their so —> **0** **Page fault.**

when 3 came it will take the place of 7 because it is least recently used → 1

Page fault

0 is already in memory so → 0 **Page fault.**

4 will take place of 1 → 1 **Page Fault**

Now for the further page reference string → 0 **Page fault** because they are already available in the memory.

Page fault rate = No. of page faults / No. of bits in the reference string.
 = 6/14

Example 2:

The page reference string: 0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7 for a memory with 3 frames.

Table Behaviour

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0*	0	0	3*	3	3	2*	2	2	1*	1	1	4*	4	4	7*
1		1*	1	1	0*	0	0	3*	3	3	2*	2	2	5*	5	5
2			2*	2	2	1*	1	1	0*	0	0	3*	3	3	6*	6
	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

Page fault 16/16=100%

Unit-5 Deadlock

In concurrent computing, a **deadlock** is a state in which each member of a group waits for another member, including itself, to take action, such as sending a message or more commonly releasing a lock. Deadlocks are a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to arbitrate shared resources and implement process synchronization.

A process in operating systems uses different resources and uses resources in the following way.

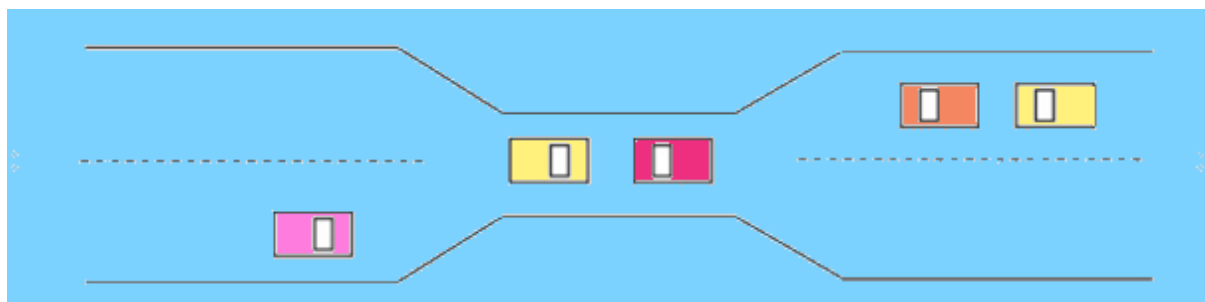
- 1) Requests a resource
- 2) Use the resource
- 3) Releases the resource

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 an **operating system**, a **deadlock** occurs when a process or thread enters a waiting state because a requested **system** resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.

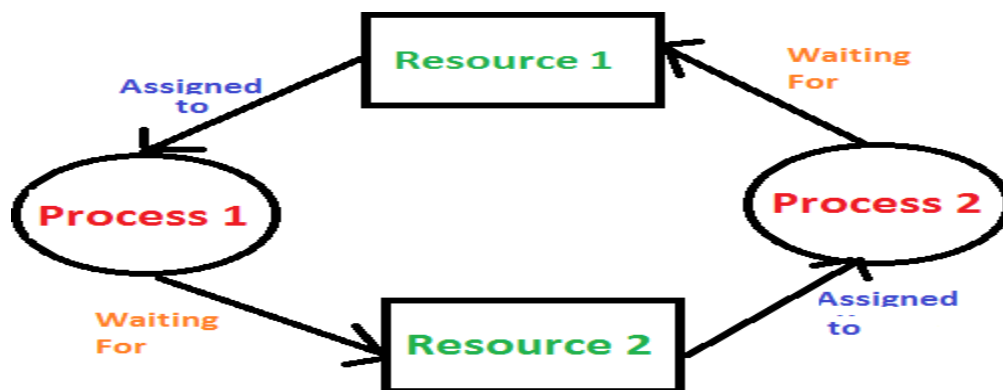
Example of Deadlock

- A real-world example would be traffic, which is going only in one direction.
- Here, a bridge is considered a resource.
- So, when Deadlock happens, it can be easily resolved if one car backs up (Preempt resources and rollback).
- Several cars may have to be backed up if a deadlock situation occurs.
- So starvation is possible.



A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).

For example: in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



5.2 SYSTEM MODEL

Let the resource types be $R_1, R_2, R_3, \dots, R_m$ (like CPU cycles, memory space, I/O devices etc.) Each resource type R_i has W_i instances. Each process utilizes a resource as follows-

1. Request: A process needing a resource will request the OS for assignment of the needed resource. Then the process waits, till OS assigns it an instance of the requested resource.
2. Assignment: The OS will assign to the requesting process an instance of the requested resource, whenever it is available. Then the process comes out from its waiting state.
3. Use: The process will use the assigned resource. In case the resource is non-shareable, the process will have exclusive access to it.
4. Release: After the process finished with the use of the assigned resource, it will return the resource to the system pool. The released resource can now be assigned to another waiting process.

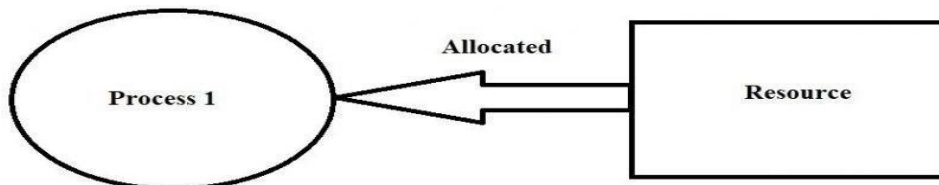
The resources are of two types (a) logical resource (b) physical resource. The Request and Release resources are called system calls, it can be accomplished by (a) **Wait** and (b) **Signal**

Necessary conditions for occurring deadlock

A deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system:

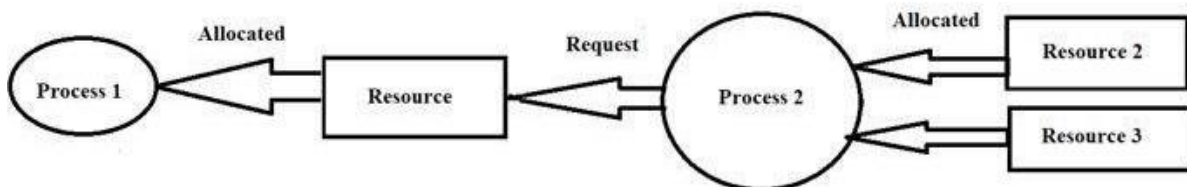
1. **Mutual exclusion:** A resource can be used by only one process at a time. If another process requests for that resource, then the requesting process must be delayed until the resource has been released. At least one resource must be held in a non-shareable mode. Otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time.

It means whatever resource we are using it must be used in a mutually exclusive way. Where only one processes use one resource at a time only. For example, the printing process is going on and all sudden another process tries to interrupt the printing process. So here in mutual exclusion situation, only after the printing task is completed then only the next task is processed. Mutual exclusion can be eliminated by sharing resources simultaneously, which is not possible practically.



2. Hold and wait or resource holding: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.

A process is holding some resources and is waiting for additional resources but those resources are acquired by some other process. From the above example, P1 is holding R1 and waiting for R2, where R2 is acquired by P2, and P2 is holding R2 and waiting for R1, where R1 is acquired by P1 is a hold and wait situation deadlock may occur in the system.



3. No preemption: Resources granted to a process can be released back to the system only as a result of the voluntarily action of that process, after the process has completed its task .

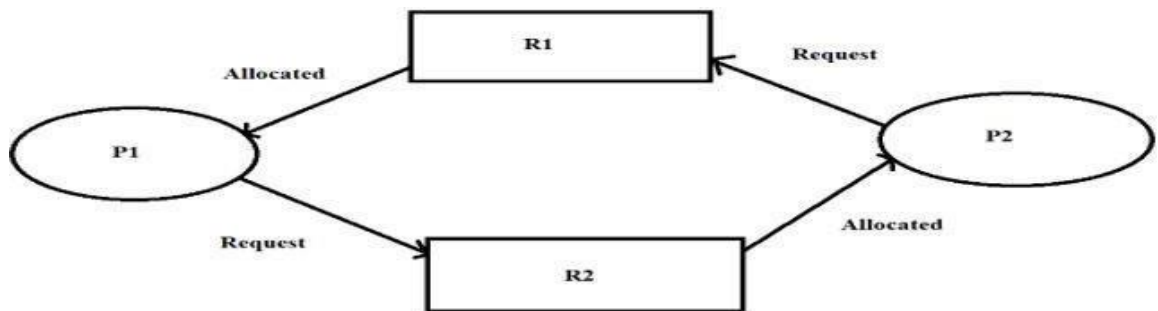
According to pre-emptive based algorithms, if there is a priority task trying to interrupt the current task. The pre-emptive algorithm it holds the current task and firstly executes priority task and get backs to its first task. A situation explained as per the above example where a process holds the resource as long as it gets executed, that is P1 can release R1 only after executing, similarly P2 release R2 only after execution. If there is no pre-emption the deadlock may occur.



no-preemption-example

4. Circular wait: Each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a [set](#) of waiting processes, $P = \{P_1, P_2, \dots, P_N\}$, such that P_1 is waiting for a resource held by P_2 , P_2 is waiting for a resource held by P_3 and so on until P_N is waiting for a resource held by P_1 .

A set of processes are said to be in deadlock if one process is waiting for a resource that is allocated to another process and that process is waiting for a resource, it is similar to the above-explained example where it is in loop form. Where P_1 is waiting for R_2 and R_2 is allocated for P_2 and P_2 is waiting for R_1 and R_1 allocated for P_1 which is a circular wait form if this condition satisfies deadlock occurs.



These four conditions are known as the Coffman conditions

While these conditions are sufficient to produce a deadlock on single-instance resource systems, they only indicate the possibility of deadlock on systems having multiple instances of resources.

Dead-Lock Detection Algorithm

Resource Allocation Graph

The cases where we allocate resources to processes, and operating system rechecks if a deadlock has occurred in the system or no using 2 main deadlock detection algorithms, they are

- Single instance
- Multiple instances of resource type

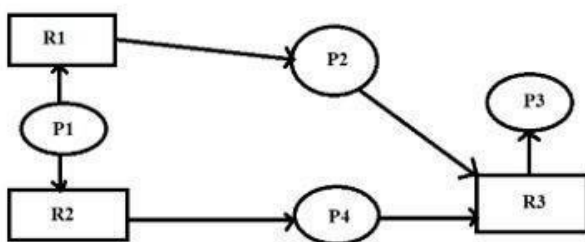
Single Instance

A single instance is a situation where a system is having single instances of all the resources. It is also known as wait for graph algorithm or resource allocation graph.

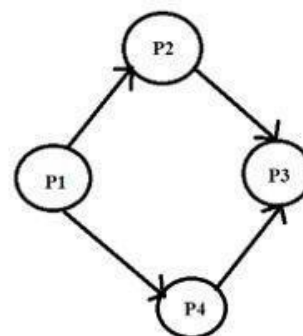
The resource allocation graph is consisting of a set of (i) processes and set of (ii) resources which are represented as two different vertices. The resources in the resource allocation graph are modified and are represented as wait for graph form. Where wait for graph form has only processes which are represented as vertices as shown below wherein,

- Resource allocation graph: Processes P1, P2, P3 and resources R1, R2, R3 are represented in the resource-allocation graph.
 - (a) A directed edge from the process P_i to the resource type R_j and denoted by $P_i \rightarrow R_j$. It signifies that i^{th} process is requesting one unit of the resource type j . (**request edge**)
 - (b) A directed edge from the resource R_i to the process P_j and denoted by $R_i \rightarrow P_j$. It signifies that one unit of i^{th} resource is held by the process j . (**allocation/assignment edge**)
- Wait for Graph: Only Processes P1, P2, P3 are mentioned in wait for the graph.
- If there is a cycle condition, that if there is a continuous flow of a process in one direction it means cycle condition exists and wait for the graph is in a deadlock condition.

Example 1: The below example shows there is no deadlock state because there is no continuous flow observed in wait for the graph.



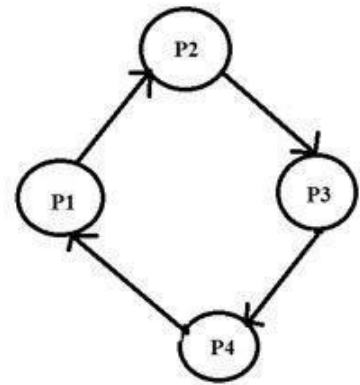
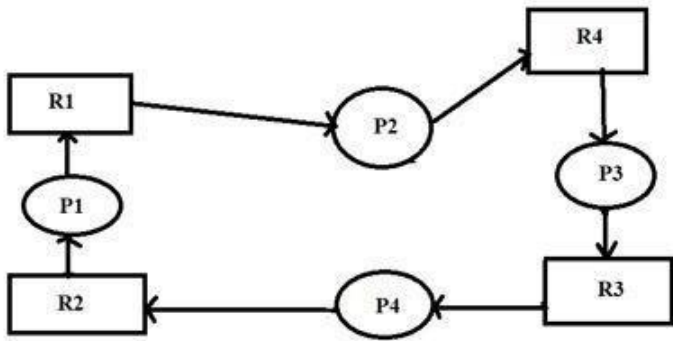
Resource allocation graph



Wait for graph

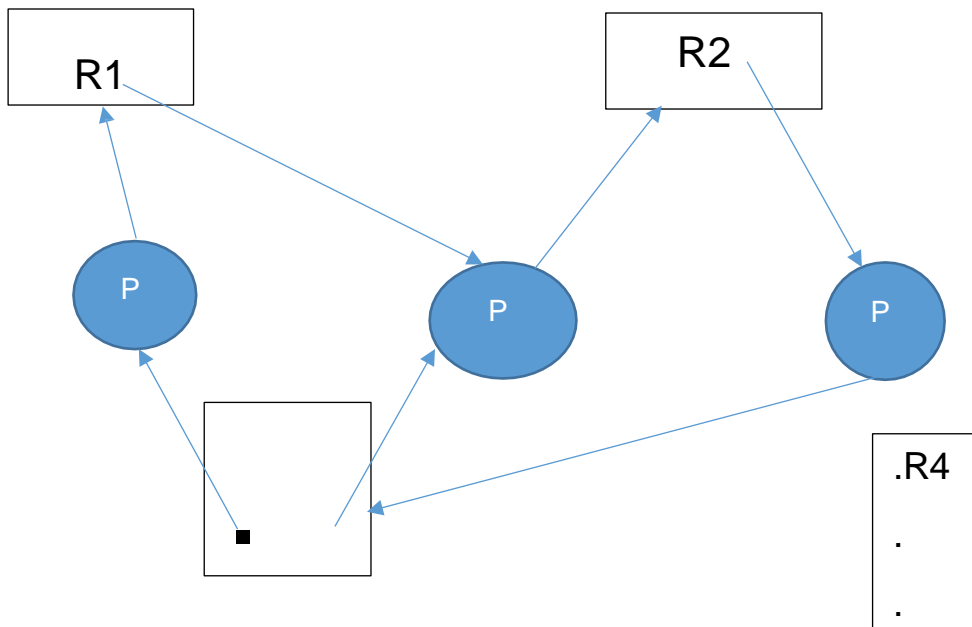
single-instance-example1

Example 2: Deadlock condition has occurred because there is a continuous flow of cycle from P1 to P4.



Resource allocation graph
single-instance-example2

Wait for graph ©F



(RAG with DEADLOCK)

Here 2 dots means that there are two instances of the resource type R3. If a cycle exists in the RAG there may or may not be a deadlock. However it is sure that no cycle exists i.e if the graph is acyclic then deadlock will not exist.

Multiple Instances of Resource Type

Multiple instances of the resource type is a situation where a system is having multiple instances of all resources, it is also known as Bankers algorithm. According to the Bankers algorithm, as soon as the process gets all its required resources, then it releases its resources.

Strategies for handling Deadlock

1. Deadlock Ignorance/Allow deadlock to occur

Deadlock Ignorance is the most widely used approach among all the mechanism. This is being used by many operating systems mainly for end user uses. In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock. This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff. **Let the deadlocks occur in the system and deteriorate the system performance.** When deadlock occurs, system will finally stop functioning. Just reboot the system.

In these types of systems, the user has to simply restart the computer in the case of deadlock. Windows and Linux are mainly using this approach.

2. Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No pre-emption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

The idea behind the approach is very simple that we have to fail one of the four conditions but there can be a big argument on its physical implementation in the system.

3. Deadlock avoidance

In deadlock avoidance, the operating system checks whether the system is in **safe state or in unsafe state** at every step which the operating system performs. The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step.

In simple words, The OS reviews each allocation so that the allocation doesn't cause the deadlock in the system.

4. Deadlock detection and recovery

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.

2. Deadlock Prevention

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

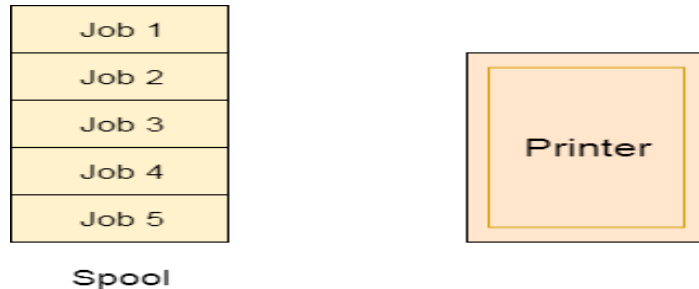
1. Mutual Exclusion

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

2. Hold and Wait

Hold and wait condition lies when **a process holds a resource and waiting for some other resource to complete its task**. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, **we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait**. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

3. No Preemption





Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

4. Circular Wait

To violate circular wait, we can **assign a priority number** to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

3. Deadlock avoidance

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States:

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfil the request of all processes, then the state of the system is called unsafe.

A safe state is not a deadlock state. Conversely, a deadlock state is an unsafe state.

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

Resources Assigned

Process	Type 1	Type 2	Type 3	Type 4
A	3	0	2	2
B	0	0	1	1
C	1	1	1	0
D	2	1	4	0

Resources still needed

Process	Type 1	Type 2	Type 3	Type 4
A	1	1	0	0
B	0	1	1	2
C	1	2	1	0
D	2	1	1	2

1. **E** = (7 6 8 4) Total instances of the resources in system
2. **P** = (6 2 8 3) Resources already allocated
3. **A** = (1 4 0 1) Resources not in use

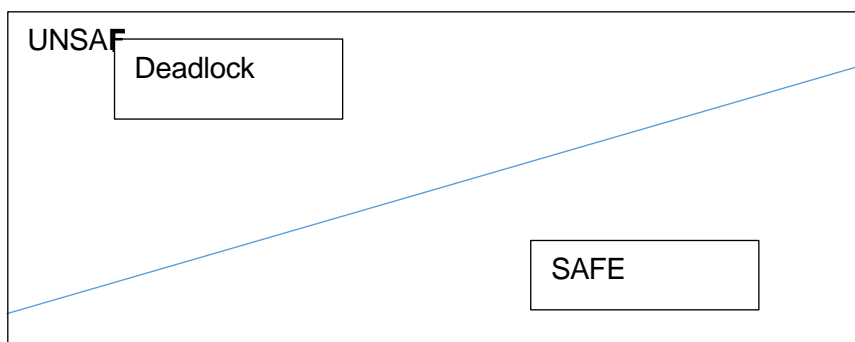
Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state. The followings are the requirements of deadlock avoidance-

1. the maximum resource requirement must be stated in advance.
2. The process under consideration must be independent. i.e no synchronization
3. There must be a fixed no. of resources to allocate.
4. No process may exist till it releases the resources it is holding.



Example: Consider that there are total 10 magnetic tapes. There are 04 processes in the system, in which process P1 may need maximum 04 tapes, P2 may need maximum 03 tapes, P3 may need maximum 05 tapes and P4 may need maximum 07 tape drives. The matrix is as follows:

Process	Max. Need	Allocated	Current Need
P1	4	2	2
P2	3	2	1
P3	5	3	2
P4	7	1	6

Is the system SAFE or UNSAFE?

Solution: Tape drives allocated = $2+2+3+1=8$

Left over drives = $10-8=2$ Available

The safe sequence $\langle P1, P2, P3, P4 \rangle$

Banker's Algorithm

This algorithm is used with resource allocation system with multiple instances of each resource type.

the process must wait until some ...
Data structures used:

Let 'n' be the number of processes in the system and 'm' be the number of resource types. We need the following data structures to implement Banker's algorithm.

1. **Available:** A vector of length 'm' indicates the number of available resources of each type. If $Available [j] = k$, there are k instances of resource type R_j available in the system.
2. **Max:** An $(n * m)$ matrix defines the maximum demand of each process. If $Max [i, j] = k$, then process P_i may request at most k instances of resource type, R_j .
3. **Allocation:** An $(n * m)$ matrix defines the number of resources of each type currently allocated to each process. If $Allocation [i, j] = k$, then process P_i is currently allocated k instances of resource type, R_j .
4. **Need:** An $(n * m)$ matrix indicates the remaining resource need of each process. If $Need [i, j] = k$ then process P_i may need k more instances of resource type R_j to complete its task. **Please note that—**

$$Need [i, j] = Max[i, j] - Allocation [i, j]$$

Each row in the matrices— Allocation and Need is treated as a **vector**. So, we can refer to them as $Allocation_i$ and $Need_i$ respectively. Herein, **the vector, Allocation**, specifies the resources currently allocated to process, P_i . Also the vector, $Need_i$ specifies the additional resources that process, P_i may still request to complete its task.

Firstly, we need to write an algorithm that finds out whether or not a system is in a safe state. This algorithm is named as a **safety algorithm** and is given below—

Safety algorithm

- Step-1:** Let Work and Finish be the two vectors of length m and n, respectively. Initialize : $Work = Available$ and $Finish [i] = false$
for $i = 1, 2, 3, \dots n$.
- Step-2:** Find i such that both:
 - (a) $Finish [i] = false$
 - (b) $Need_i \leq Work$If no such i exists, go to step-4.
- Step-3:** $Work = Work + Allocation_i$
 $Finish [i] = true$
go to step-2
- Step-4:** If $Finish [i] = true$ for all i, then the system is in a safe state else it is in unsafe state.
This algorithm takes $O(m \times n^2)$ operations to decide whether a state is safe.



Next, we develop a **Resource-Request Algorithm**. 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:-

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

$$\begin{aligned} Available &= Available - Request_i; \\ Allocation_i &= Allocation_i + Request_i; \\ Need_i &= Need_i - Request_i; \end{aligned}$$

If the resulting resource-allocation is safe, then 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.

This algorithm is known as a Banker's algorithm and has good results. **We are in a position to solve some problems now.**

Example 1: Let there be five processes (P_1 to P_5) and three resource types A, B, C.
 Resource type A has 10 instances.
 Resource type B has 5 instances.
 Resource type C has 7 instances.

and

Suppose that at time, t_0 , the following snapshot of the system has been taken:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2
P_2	2	0	0	3	2	2			
P_3	3	0	2	9	0	2			
P_4	2	1	1	2	2	2			
P_5	0	0	2	4	3	3			

Find the safe sequence? Is this system safe at time, t_0 . If process, P_2 sends one additional request, can it be granted access? Say, P_2 sends the request of (1, 0, 2). What is the new content of matrix-Need? Is this state, a safe system state? Explain.

Solution. Firstly, we need to find vector, need as follows—

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

So, we get

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2	7	4	3
P_2	2	0	0	3	2	2				1	2	2
P_3	3	0	2	9	0	2				6	0	0
P_4	2	1	1	2	2	2				0	1	1
P_5	0	0	2	4	3	3				4	3	1



We can find a safe sequence as $\langle P_2, P_4, P_5, P_3, P_1 \rangle$. So, we can say that the system is in a safe state at time, t_0 .

It is given that P_2 makes another request now—

A B C

Request₂ = (1, 0, 2)

This request is sent to OS. To decide whether this request can be immediately granted or not, we first check that Request₁ ≤ Available, which is true. We then pretend that this request has been fulfilled. We arrive at a new state as given below—

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P ₁	0	1	0	7	5	3	2	3	0	7	4	3
P ₂	3	0	2	3	2	2				0	2	0
P ₃	3	0	2	9	0	2				6	0	0
P ₄	2	1	1	2	2	2				0	1	1
P ₅	0	0	2	4	3	3				4	3	1

Now, we check whether this new system is safe or not. For that, we execute our safety algorithm and find the sequence $\langle P_2, P_4, P_5, P_1, P_3 \rangle$ satisfies our safety requirement. Hence, we can immediately grant the request of process, P_2 .

Please note that when the system is in this state, a request for (3, 3, 0) by process, P_2 cannot be granted since the resources are not available. Also note that a request for (0, 2, 0) by process, P_1 cannot be granted, even though the resources are available because after allocating the required resources by the process, P_1 , there is no safe sequence available.

Example 2. Consider a system with five (5) process— P_0 to P_4 and three (3) resources A, B, C. Given that—

Resource type A has 10 instances.

Resource type B as 5 instances.

Resource type C has 7 instances.

Suppose that at time, t_0 , the following snapshot of the system has been taken—

Process	Allocation			Max		
	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃
P ₀	0	1	0	7	5	3
P ₁	2	0	0	3	2	2
P ₂	3	0	2	9	0	2
P ₃	2	1	1	2	2	2
P ₄	0	0	2	4	3	3

Calculate the available matrix of resources. Find the need matrix. Find the safe sequence?

Solution. The available matrix is calculated as follows—

$$\therefore \text{Available} = \text{Total} - \text{Allocation}$$

$$\therefore R_1 = 10 - (2 + 3 + 2) = 3$$

$$R_2 = 5 - (1 + 1) = 3$$

$$R_3 = 7 - (2 + 1 + 2) = 2$$

∴ The available matrix is (3, 3, 2). Now, what is the Need matrix?

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

So, we get

Process	R ₁	R ₂	R ₃
P ₀	(7-0)	(5-1)	(3-0)
P ₁	(3-2)	(2-0)	(2-0)
P ₂	(9-3)	(0-0)	(2-2)
P ₃	(2-2)	(2-1)	(2-1)
P ₄	(4-0)	(3-0)	(3-2)

which gives us the Need matrix as follows—

Process	Need		
	R ₁	R ₂	R ₃
P ₀	7	4	3
P ₁	1	2	2
P ₂	6	0	0
P ₃	0	1	1
P ₄	4	3	1

Applying Banker's algorithm, we now search the Need-matrix from P₀ to P₄ which is less than the available-matrix i.e., (3, 3, 2). We find that (1, 2, 2) < (3, 3, 2). So, allocate the requesting resource to P₁. Now, the process, P₁'s allocation is—

$$(2, 0, 0) + (1, 2, 2) = (3, 2, 2)$$

After the completion of job P₁, it releases the total resources. Then the available resource is—

$$(3, 3, 2) + (2, 0, 0) = (5, 3, 2)$$

We search the need matrix again, which is less than the available. Now, we find that—

$$(0, 1, 1) < (5, 3, 2)$$

So, allocate resources to P₃. After P₃ completes, it releases all its resources. Then,

$$\text{Available} = \text{Available} + \text{Allocation}$$

$$\text{i.e., } (5, 3, 2) + (2, 1, 1) = (7, 4, 3)$$

This process is continued till we find the safe state. So, the safe sequence in this example is — <

Safe sequence is < P₁, P₃, P₄, P₂, P₀ >.

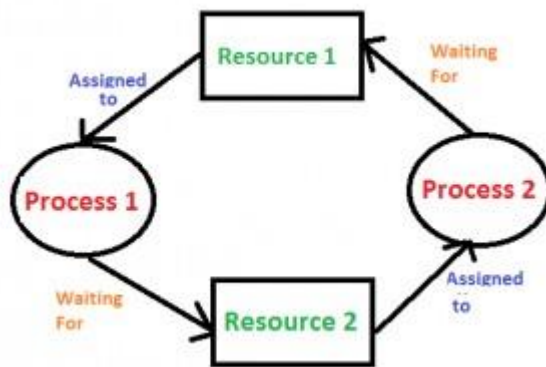
4. Deadlock Detection & Recovery

(A) Deadlock Detection

1. If resources have single instance:

In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the

sufficient condition for deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle $R1 \rightarrow P1 \rightarrow R2 \rightarrow P2$. So, deadlock is Confirmed.

2. If there are multiple instances of resources: Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

(B)Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real-time operating systems use Deadlock recovery. Deadlock recovery performs when a [deadlock](#) is detected.

When [deadlock detected](#), then our system stops working, and after the recovery of the deadlock, our system starts working again.

Therefore, after the detection of deadlock, a method/way must require to recover that deadlock to run the system again. The method/way is called as deadlock recovery.

- Deadlock recovery through preemption
- Deadlock recovery through rollback
- Deadlock recovery through killing processes

Let's discuss about all the above three ways of deadlock recovery one by one.

(A)Deadlock Recovery through Preemption

The ability to take a resource away from a process, have another process use it, and then give it back without the process noticing. It is highly dependent on the nature of the resource.

Deadlock recovery through preemption is too difficult or sometime impossible.

(B) Deadlock Recovery through Rollback

In this case of deadlock recovery through rollback, whenever a deadlock is detected, it is easy to see which resources are needed.

To do the recovery of deadlock, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource just by starting one of its earlier checkpoints.

© Deadlock Recovery through Killing Processes

This method of deadlock recovery through killing processes is the simplest way of deadlock recovery.

Sometime it is best to kill a process that can be return from the beginning with no ill effects.