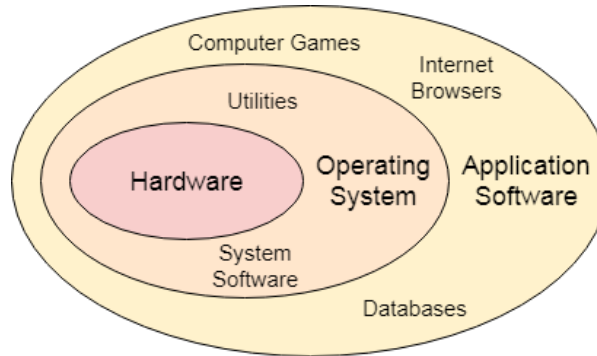


INTRODUCTION

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.

We need a system which can act as an intermediary and manage all the processes and resources present in the system.



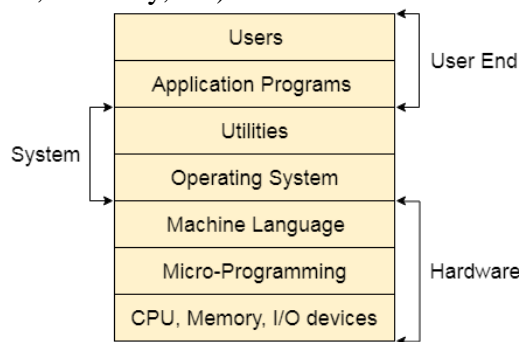
An Operating System can be defined as an interface between user and hardware. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

STRUCTURE OF OPERATING SYSTEM-

A Computer System consists of:

- Users (people who are using the computer)
- Application Programs (Compilers, Databases, Games, Video player, Browsers, etc.)
- System Programs (Shells, Editors, Compilers, etc.)
- Operating System (A special program which acts as an interface between user and hardware)
- Hardware (CPU, Disks, Memory, etc)



FUNCTIONS OF OPERATING SYSTEM-

1. Process Management

2. Process Synchronization
3. Memory Management
4. CPU Scheduling
5. File Management
6. Security

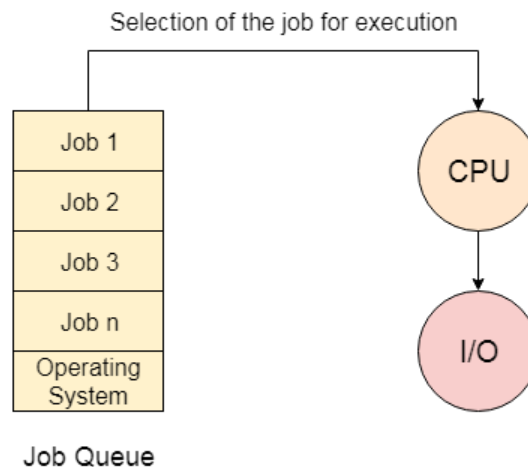
TYPES OF OS

There are many types of operating system exists in the current scenario:

Batch Operating System

In the era of 1970s, the Batch processing was very popular. The Jobs were executed in batches. People were used to have a single computer which was called mainframe. In Batch operating system, access is given to more than one person; they submit their respective jobs to the system for the execution.

The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one. The users collect their respective output when all the jobs get executed.



Disadvantages of Batch OS

1. Starvation

Batch processing suffers from starvation. If there are five jobs J1, J2, J3, J4, J4 and J5 present in the batch. If the execution time of J1 is very high then other four jobs will never be going to get executed or they will have to wait for a very high time. Hence the other processes get starved.

2. Not Interactive

Batch Processing is not suitable for the jobs which are dependent on the user's input. If a job requires the input of two numbers from the console then it will never be going to get it in the batch processing scenario since the user is not present at the time of execution.

Multiprogramming Operating System

Multiprogramming is an extension to the batch processing where the CPU is kept always busy. Each process needs two types of system time: CPU time and IO time.

In multiprogramming environment, for the time a process does its I/O, The CPU can start the execution of other processes. Therefore, multiprogramming improves the efficiency of the system.

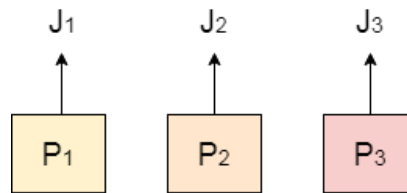
Multitasking Operating System

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking –

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.
- A program that is loaded into memory and is executing is commonly referred to as a **process**.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprocessing Operating System

In Multiprocessing, Parallel computing is achieved. There are more than one processors present in the system which can execute more than one process at the same time. This will increase the throughput of the system.



Multi Processing

Real Time Operating System

In Real Time systems, each job carries a certain deadline within which the Job is supposed to be completed, otherwise the huge loss will be there or even if the result is produced then it will be completely useless.

The Application of a Real Time system exists in the case of military applications, if you want to drop a missile then the missile is supposed to be dropped with certain precision.

The real time operating systems can be of 2 types –

1. **Hard Real Time operating system:**

These operating systems guarantee that critical tasks be completed within a range of time.

For example, a robot is hired to weld a car body, if robot welds too early or too late, the car cannot be sold, so it is a hard real time system that require to complete car welding by robot hardly on the time.

2. **Soft real time operating system:**

This operating systems provides some relaxation in time limit.

DISTRIBUTED ENVIRONMENT

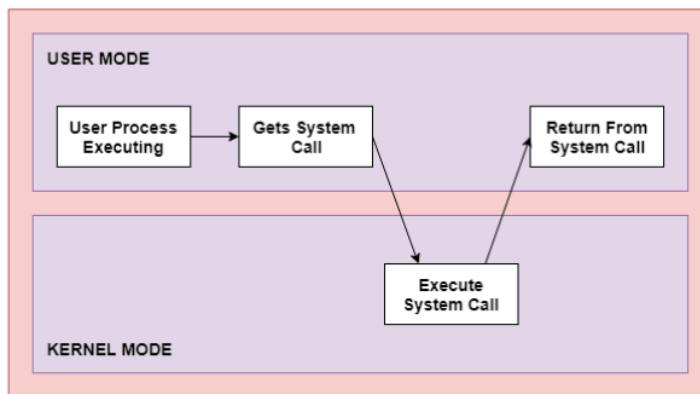
A distributed environment refers to multiple independent CPUs or processors in a computer system. An operating system does the following activities related to distributed environment –

- The OS distributes computation logics among several physical processors.
- The processors do not share memory or a clock. Instead, each processor has its own local memory.
- The OS manages the communications between the processors. They communicate with each other through various communication lines.

SYSTEM CALLS

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers. System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

A figure representing the execution of the system call is given as follows –



As can be seen from this diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

In general, system calls are required in the following situations –

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
- Creation and management of new processes.
- Network connections also require system calls. This includes sending and receiving packets.
- Access to a hardware devices such as a printer, scanner etc. requires a system call.

Types of System Calls

There are mainly five types of system calls. These are explained in detail as follows –

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Some of the examples of all the above types of system calls in Windows and Unix are given as follows –

Types of System Calls	Windows	Linux
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

There are many different system calls as shown above. Details of some of those system calls are as follows –

open()

The open() system call is used to provide access to a file in a file system. This system call allocates resources to the file and provides a handle that the process uses to refer to the file. A file can be opened by multiple processes at the same time or be restricted to one process. It all depends on the file organisation and file system.

read()

The read() system call is used to access data from a file that is stored in the file system. The file to read can be identified by its file descriptor and it should be opened using open() before it can

be read. In general, the read() system calls takes three arguments i.e. the file descriptor, buffer which stores read data and number of bytes to be read from the file.

write()

The write() system calls writes the data from a user buffer into a device such as a file. This system call is one of the ways to output data from a program. In general, the write system calls takes three arguments i.e. file descriptor, pointer to the buffer where data is stored and number of bytes to write from the buffer.

close()

The close() system call is used to terminate access to a file system. Using this system call means that the file is no longer required by the program and so the buffers are flushed, the file metadata is updated and the file resources are de-allocated.

PROCESS MANAGEMENT INTRODUCTION

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources. There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

ATTRIBUTES OF A PROCESS

The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process. Attributes which are stored in the PCB are described below.

1. Process ID

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

2. Program counter

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

3. Process State

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting.

4. Priority

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

5. General Purpose Registers

Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

6. List of open files

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

7. List of open devices

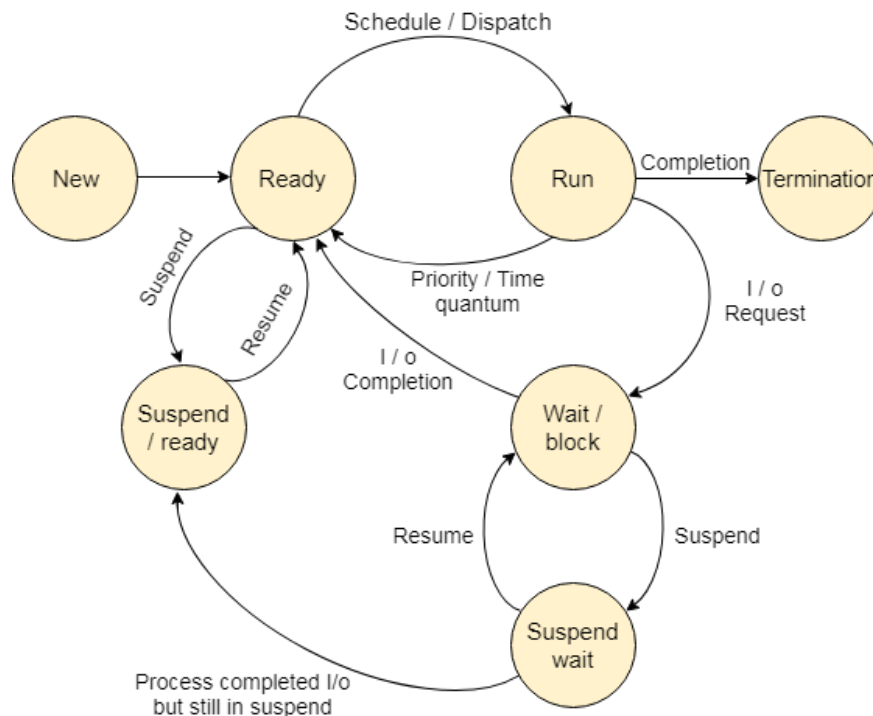
OS also maintain the list of all open devices which are used during the execution of the process.

Process ID
Program Counter
Process State
Priority
General Purpose Registers
List of Open Files
List of Open Devices

Process Attributes

PROCESS STATES

The process, from its creation to completion, passes through various states. The minimum number of states is five. The names of the states are not standardized although the process may be in one of the following states during execution.



1. New

A program which is going to be picked up by the OS into the main memory is called a new process.

2. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

OPERATIONS ON THE PROCESS

1. Creation

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

2. Scheduling

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

3. Execution

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

4. Deletion/killing

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

PROCESS SCHEDULERS

Operating system uses various schedulers for the process scheduling described below.

1. Long term scheduler

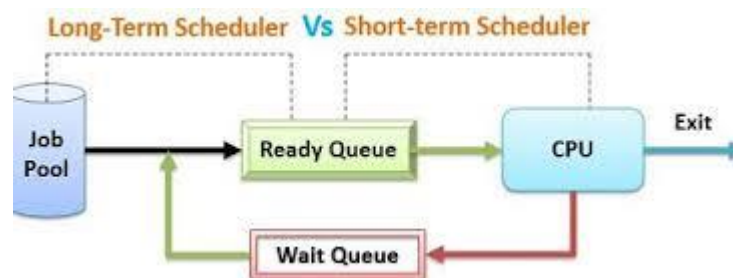
Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory. Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool. If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

2. Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution. A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time. This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

3. Medium term scheduler

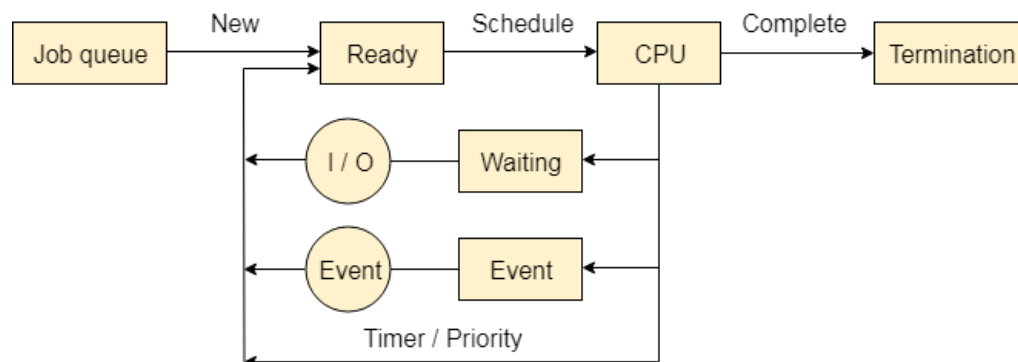
Medium term scheduler takes care of the swapped out processes. If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting. Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes. It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.



Medium Term Scheduler

PROCESS QUEUES

The Operating system manages various types of queues for each of the process states. The PCB related to the process is also stored in the queue of the same state. If the Process is moved from one state to another state then its PCB is also unlinked from the corresponding queue and added to the other state queue in which the transition is made.



There are the following queues maintained by the Operating system.

1. Job Queue

In starting, all the processes get stored in the job queue. It is maintained in the secondary memory. The long term scheduler (Job scheduler) picks some of the jobs and put them in the primary memory.

2. Ready Queue

Ready queue is maintained in primary memory. The short term scheduler picks the job from the ready queue and dispatch to the CPU for the execution.

3. Waiting Queue

When the process needs some IO operation in order to complete its execution, OS changes the state of the process from running to waiting. The context (PCB) associated with the process gets stored on the waiting queue which will be used by the Processor when the process finishes the IO.

CONTEXT SWITCH

1. Switching the CPU to another process requires **saving** the state of the old process and **loading** the saved state for the new process. This task is known as a **Context Switch**.
2. The **context** of a process is represented in the **Process Control Block (PCB)** of a process; it includes the value of the CPU registers, the process state and memory-management information. When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
3. Context switch time is **pure overhead**, because the **system does no useful work while switching**. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.
4. Context Switching has become such a performance **bottleneck** that programmers are using new structures (threads) to avoid it whenever and wherever possible.

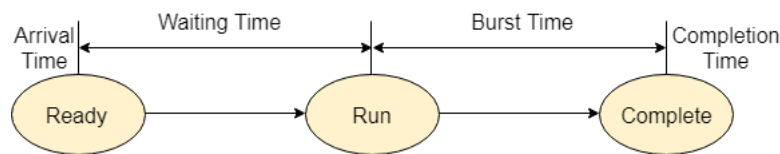
VARIOUS TIMES RELATED TO THE PROCESS

1. Arrival Time

The time at which the process enters into the ready queue is called the arrival time.

2. Burst Time

The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.



$$CT - AT = WT + BT$$

$$TAT = CT - AT$$

$$\text{Waiting Time} = TAT - BT$$

TAT → Turn around time

BT → Burst time

AT → Arrival time

3. Completion Time

The Time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

4. Turnaround time

The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

5. Waiting Time

The Total amount of time for which the process waits for the CPU to be assigned is called waiting time.

6. Response Time

The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

CPU SCHEDULING

In the uniprogramming systems like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idle. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given.

In Multiprogramming systems, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes.

This is a task of the short term scheduler to schedule the CPU for the number of processes present in the Job Pool. Whenever the running process requests some IO operation then the short term scheduler saves the current context of the process (also called PCB) and changes its state from running to waiting. During the time, process is in waiting state; the Short term scheduler picks another process from the ready queue and assigns the CPU to this process. This procedure is called **context switching**.

PROCESS CONTROL BLOCK

The Operating system maintains a process control block during the lifetime of the process. The Process control block is deleted when the process is terminated or killed. There is the following information which is saved in the process control block and is changing with the state of the process.

Process ID
Process State
Pointer
Priority
Program Counter
CPU Registers
I/O Information
Accounting Information
etc.

Why do we need Scheduling?

In Multiprogramming, if the long term scheduler picks more I/O bound processes then most of the time, the CPU remains idle. The task of Operating system is to optimize the utilization of resources.

If most of the running processes change their state from running to waiting then there may always be a possibility of deadlock in the system. Hence to reduce this overhead, the OS needs to schedule the jobs to get the optimal utilization of CPU and to avoid the possibility to deadlock.

SCHEDULING ALGORITHMS

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fair allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

NON-PREEMPTIVE SCHEDULING

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware (for example: a timer) needed for preemptive scheduling.

PREEMPTIVE SCHEDULING

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

SCHEDULING ALGORITHMS

1. ***First Come First Serve(FCFS) Scheduling*** - It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.
 2. ***Shortest-Job-First(SJF) Scheduling-*** The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.
 3. ***Priority Scheduling-*** In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.
 4. ***Round Robin(RR) Scheduling*** - In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum)
-

and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

5. **Shortest Remaining Time First**- It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.
6. **Highest Response Ratio Next**- In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.
7. **Multilevel Queue Scheduling**
8. **Multilevel Feedback Queue Scheduling**

FCFS SCHEDULING

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serve

Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Example

There are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

The Turnaround time and the waiting time are calculated by using the following formula.

Turn Around Time = Completion Time - Arrival Time

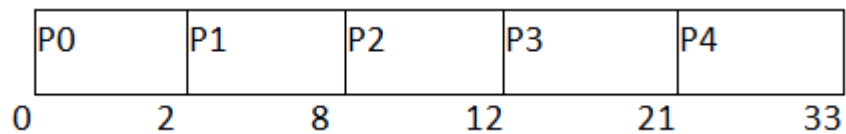
Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
-------------------	---------------------	-------------------	------------------------	-------------------------	---------------------

0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	8	4
3	3	9	21	18	9
4	4	12	33	29	17

Avg Waiting Time=31/5

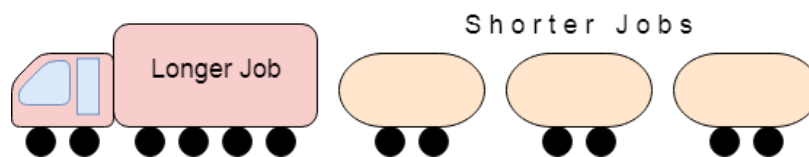


CONVOY EFFECT IN FCFS

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

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

The Convoy Effect, Visualized Starvation



Example-We have 3 processes named as **P1, P2 and P3**. The Burt Time of process P1 is highest.

The Turnaround time and the waiting time in the following table, are calculated by the formula,

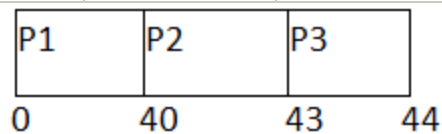
Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turn Around Time - Burst Time

In the First scenario, The Process P1 arrives at the first in the queue although; the burst time of the process is the highest among all. Since, the Scheduling algorithm, we are following is FCFS hence the CPU will execute the Process P1 first.

In this schedule, the average waiting time of the system will be very high. That is because of the convoy effect. The other processes P2, P3 have to wait for their turn for 40 units of time although their burst time is very low. This schedule suffers from starvation.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	40	40	40	0
2	1	3	43	42	39
3	1	1	44	43	42

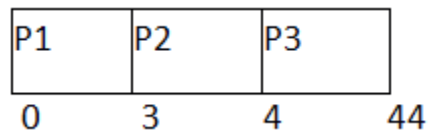


$$\text{Avg waiting Time} = 81/3$$

In the Second scenario, If Process P1 would have arrived at the last of the queue and the other processes P2 and P3 at earlier then the problem of starvation would not be there.

Following example shows the deviation in the waiting times of both the scenarios. Although the length of the schedule is same that is 44 units but the waiting time will be lesser in this schedule.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	40	44	43	3
2	0	3	3	3	0
3	0	1	4	4	3



$$\text{Avg Waiting Time} = 6/3$$

SHORTEST JOB FIRST (SJF) SCHEDULING

SJF scheduling algorithm, schedules the processes according to their burst time. In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

Disadvantages of SJF

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

Example

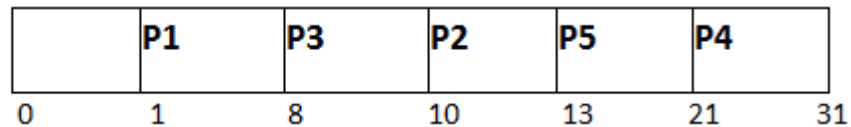
In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives). According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue. Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time. Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

So that's how the procedure will go on in **shortest job first (SJF)** scheduling algorithm.



Avg Waiting Time = $27/5$

SHORTEST REMAINING TIME FIRST (SRTF) SCHEDULING

This Algorithm is the preemptive version of SJF scheduling. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the ready queue, No preemption will be done and the algorithm will work as SJF scheduling. The context of the process is saved in the Process Control Block when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the next execution of this process.

Example

In this Example, there are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
1	0	8	20	20	12	0
2	1	4	10	9	5	1
3	2	2	4	2	0	2
4	3	1	5	2	1	4
5	4	3	13	9	6	10
6	5	2	7	2	0	5

	P1	P2	P3	P3	P4	P6	P2	P5	P1	
0	1	2	3	4	5	7	10	13	20	

Avg Waiting Time = $24/6$

The Gantt chart is prepared according to the arrival and burst time given in the table.

1. Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled.
2. The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a preemptive one, the current execution is stopped and the scheduler checks for the process with the least burst time. Till now, there are two processes available in the ready queue. The OS has executed P1 for one unit of time till now; the remaining burst time of P1 is 7 units. The burst time of Process P2 is 4 units. Hence Process P2 is scheduled on the CPU according to the algorithm.

The next process P3 arrives at time unit 2. At this time, the execution of process P3 is stopped and the process with the least remaining burst time is searched. Since the process P3 has 2 unit of burst time hence it will be given priority over others.

The Next Process P4 arrives at time unit 3. At this arrival, the scheduler will stop the execution of P4 and check which process is having least burst time among the available processes (P1, P2, P3 and P4). P1 and P2 are having the remaining burst time 7 units and 3 units respectively.

P3 and P4 are having the remaining burst time 1 unit each. Since, both are equal hence the scheduling will be done according to their arrival time. P3 arrives earlier than P4 and therefore it will be scheduled again. The Next Process P5 arrives at time unit 4. Till this time, the Process P3 has completed its execution and it is no more in the list. The scheduler will compare the remaining burst time of all the available processes. Since the burst time of process P4 is 1 which is least among all hence this will be scheduled.

The Next Process P6 arrives at time unit 5, till this time, the Process P4 has completed its execution. We have 4 available processes till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst time of P6 is the least among all hence P6 is scheduled. Since, now, all the processes are available hence the algorithm will now work same as SJF. P6 will be executed till its completion and then the process with the least remaining time will be scheduled.

Once all the processes arrive, No preemption is done and the algorithm will work as SJF.

ROUND ROBIN SCHEDULING ALGORITHM

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

Advantages

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fare allocation of CPU.

Disadvantages

1. The higher the time quantum, the higher the response time in the system.
2. The lower the time quantum, the higher the context switching overhead in the system.
3. Deciding a perfect time quantum is really a very difficult task in the system.

Q. Consider the following processes with arrival time and burst time. Calculate average turnaround time, average waiting time and average response time using round robin with time quantum 3?

Process id	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

Queue: P4,P5,P3,P2,P4,P1,P6,P3,P2,P4,P1,P3

CPU IDEAL	P4	P5	P3	P2	P4	P1	P6	P3	P2	P4	P1	P3
-----------	----	----	----	----	----	----	----	----	----	----	----	----

Time: 0 1 4 6 9 12 15 18 21 24 27 30 32 33

Process id	Arrival time	Burst time	Completion time	Turnaround time	Waiting time	Response time
P1	5	5	32	27	22	10
P2	4	6	27	23	17	5
P3	3	7	33	30	23	3
P4	1	9	30	29	20	0
P5	2	2	6	4	2	2
P6	6	3	21	15	12	12

Average turnaround time= $\frac{(27+23+30+29+4+15)}{6}=21.33$

Average waiting time= $\frac{(22+17+23+20+2+12)}{6}=16$

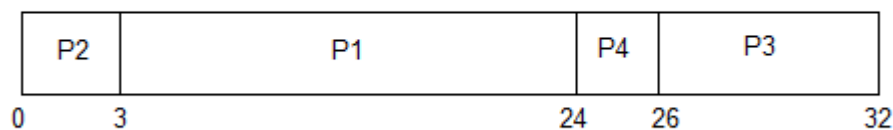
Average response time= $\frac{(10+5+3+0+2+12)}{6}=5.33$

PRIORITY SCHEDULING

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, $(0 + 3 + 24 + 26) / 4 = 13.25$ ms

Shortest Remaining Time First (SRTF) Scheduling Algorithm

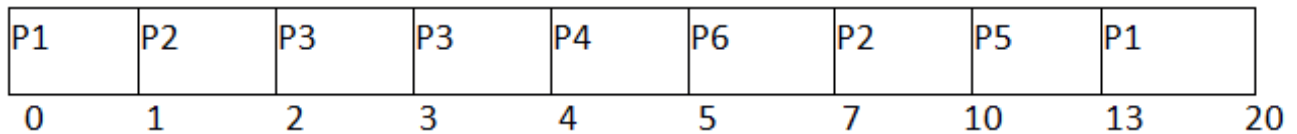
This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

Example: There are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
1	0	8	20	20	12	0

2	1	4	10	9	5	1
3	2	2	4	2	0	2
4	3	1	5	2	1	4
5	4	3	13	9	6	10
6	5	2	7	2	0	5



$$\text{Avg Waiting Time} = 24/6$$

The Gantt chart is prepared according to the arrival and burst time given in the table.

1. Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled.
2. The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a preemptive one, the current execution is stopped and the scheduler checks for the process with the least burst time. Till now, there are two processes available in the ready queue. The OS has executed P1 for one unit of time till now; the remaining burst time of P1 is 7 units. The burst time of Process P2 is 4 units. Hence Process P2 is scheduled on the CPU according to the algorithm.
3. The next process P3 arrives at time unit 2. At this time, the execution of process P3 is stopped and the process with the least remaining burst time is searched. Since the process P3 has 2 unit of burst time hence it will be given priority over others.
4. The Next Process P4 arrives at time unit 3. At this arrival, the scheduler will stop the execution of P4 and check which process is having least burst time among the available processes (P1, P2, P3 and P4). P1 and P2 are having the remaining burst time 7 units and 3 units respectively.

P3 and P4 are having the remaining burst time 1 unit each. Since, both are equal hence the scheduling will be done according to their arrival time. P3 arrives earlier than P4 and therefore it will be scheduled again.

5. The Next Process P5 arrives at time unit 4. Till this time, the Process P3 has completed its execution and it is no more in the list. The scheduler will compare the remaining burst time of all the available processes. Since the burst time of process P4 is 1 which is least among all hence this will be scheduled.
6. The Next Process P6 arrives at time unit 5, till this time, the Process P4 has completed its execution. We have 4 available processes till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst time of P6 is the least among all hence P6 is scheduled. Since, now, all the processes are available hence the algorithm will now work same as SJF. P6 will be executed till its completion and then the process with the least remaining time will be scheduled.

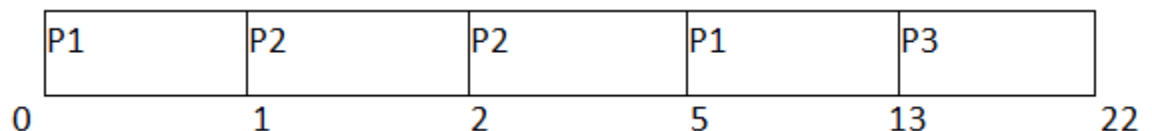
Once all the processes arrive, No preemption is done and the algorithm will work as SJF.

Q. Given the arrival time and burst time of 3 jobs in the table below. Calculate the Average waiting time of the system.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	9	13	13	4
2	1	4	5	4	0
3	2	9	22	20	11

There are three jobs P1, P2 and P3. P1 arrives at time unit 0; it will be scheduled first for the time until the next process arrives. P2 arrives at 1 unit of time. Its burst time is 4 units which is least among the jobs in the queue. Hence it will be scheduled next.

At time 2, P3 will arrive with burst time 9. Since remaining burst time of P2 is 3 units which are least among the available jobs. Hence the processor will continue its execution till its completion. Because all the jobs have been arrived so no preemption will be done now and all the jobs will be executed till the completion according to SJF.



$$\text{Avg Waiting Time} = (4+0+11)/3 = 5 \text{ units}$$

MULTILEVEL QUEUE SCHEDULING

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

For example: A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

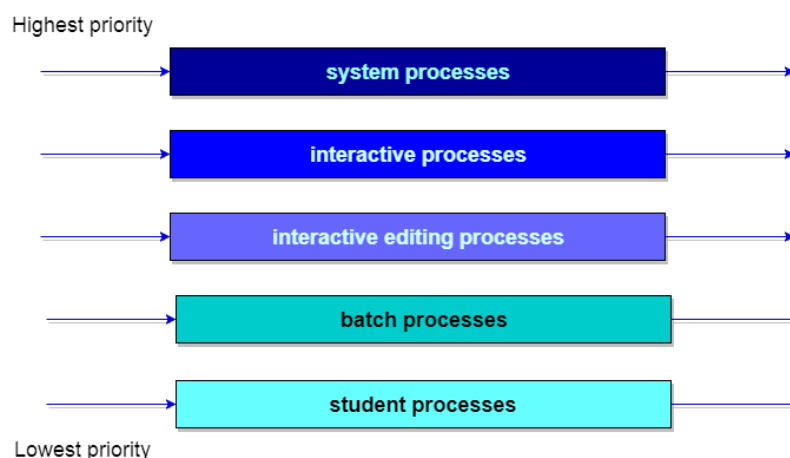
For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. For example: The foreground queue may have absolute priority over the background queue.

An example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

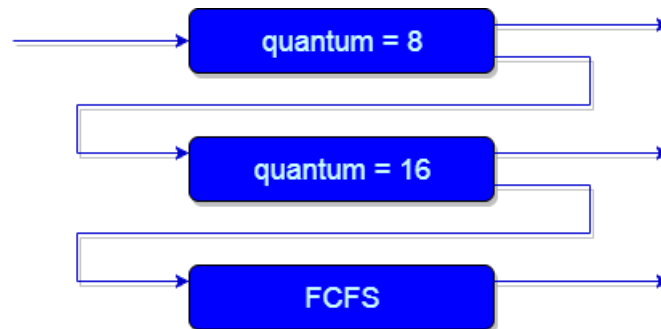
Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.



MULTILEVEL FEEDBACK QUEUE SCHEDULING

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the most general scheme, it is also the most complex.

SYNCHRONIZATION

When two or more process cooperates with each other, their order of execution must be preserved otherwise there can be conflicts in their execution and inappropriate outputs can be produced.

A cooperative process is the one which can affect the execution of other process or can be affected by the execution of other process. Such processes need to be synchronized so that their order of execution can be guaranteed.

The procedure involved in preserving the appropriate order of execution of cooperative processes is known as **Process Synchronization**. There are various synchronization mechanisms that are used to synchronize the processes.

RACE CONDITION

A Race Condition typically occurs when two or more threads try to read, write and possibly make the decisions based on the memory that they are accessing concurrently.

CRITICAL SECTION

The regions of a program that try to access shared resources and may cause race conditions are called critical section. To avoid race condition among the processes, we need to assure that only one process at a time can execute within the critical section.

CRITICAL SECTION PROBLEM

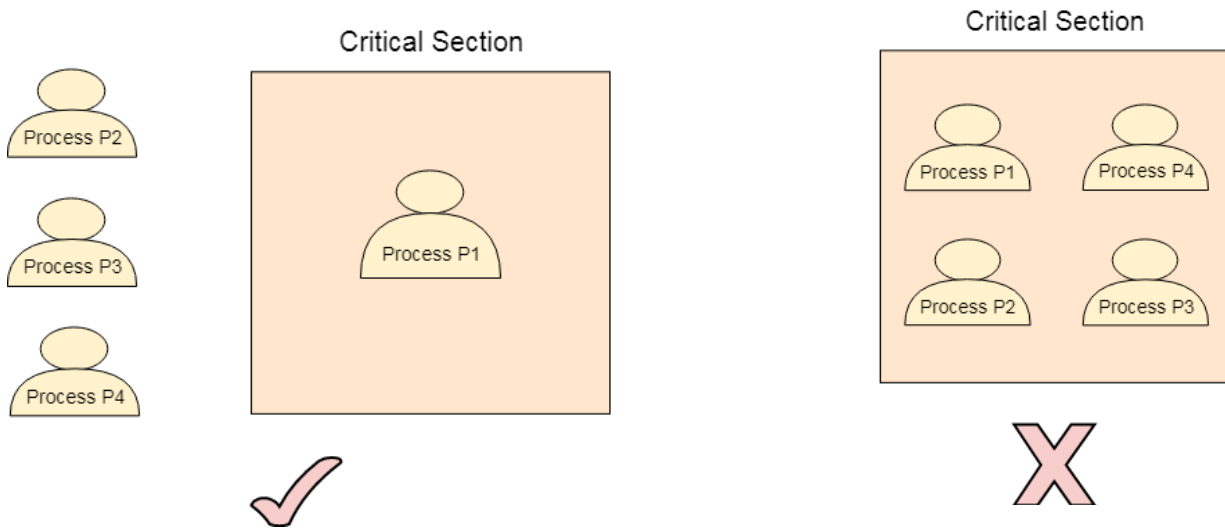
Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device. The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise. In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

Requirements of Synchronization mechanisms

Primary

1. **Mutual Exclusion**-Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.
-



2. Progress

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

Secondary

1. Bounded Waiting

We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

2. Architectural Neutrality

Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

LOCK VARIABLE

This is the simplest synchronization mechanism. This is a Software Mechanism implemented in User mode. This is a busy waiting solution which can be used for more than two processes. In this mechanism, a Lock variable lock is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied.

A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.

The pseudo code of the mechanism looks like following.

1. Entry Section →
2. While (lock! = 0);

3. Lock = 1;
4. //Critical Section
5. Exit Section →
6. Lock =0;

If we look at the Pseudo Code, we find that there are three sections in the code. Entry Section, Critical Section and the exit section.

Initially the value of **lock variable** is **0**. The process which needs to get into the **critical section**, enters into the entry section and checks the condition provided in the while loop.

The process will wait infinitely until the value of **lock** is 1 (that is implied by while loop). Since, at the very first time critical section is vacant hence the process will enter the critical section by setting the lock variable as 1. When the process exits from the critical section, then in the exit section, it reassigns the value of **lock** as 0.

Every Synchronization mechanism is judged on the basis of four conditions.

1. Mutual Exclusion
2. Progress
3. Bounded Waiting
4. Portability

Out of the four parameters, Mutual Exclusion and Progress must be provided by any solution.

MUTUAL EXCLUSION

The lock variable mechanism doesn't provide Mutual Exclusion in some of the cases. This can be better described by looking at the pseudo code by the Operating System point of view I.E. Assembly code of the program. Let's convert the Code into the assembly language.

1. *Load Lock, R0*
2. *CMP R0, #0*
3. *JNZ Step 1*
4. *Store #1, Lock*
5. *Store #0, Lock*

Let us consider that we have two processes P1 and P2. The process P1 wants to execute its critical section. P1 gets into the entry section. Since the value of lock is 0 hence P1 changes its value from 0 to 1 and enters into the critical section.

Meanwhile, P1 is preempted by the CPU and P2 gets scheduled. Now there is no other process in the critical section and the value of lock variable is 0. P2 also wants to execute its critical section. It enters into the critical section by setting the lock variable to 1.

Now, CPU changes P1's state from waiting to running. P1 is yet to finish its critical section. P1 has already checked the value of lock variable and remembers that its value was 0 when it

previously checked it. Hence, it also enters into the critical section without checking the updated value of lock variable. Now, we got two processes in the critical section. According to the condition of mutual exclusion, more than one process in the critical section must not be present at the same time. Hence, the lock variable mechanism doesn't guarantee the mutual exclusion.

The problem with the lock variable mechanism is that, at the same time, more than one process can see the vacant tag and more than one process can enter in the critical section. Hence, the lock variable doesn't provide the mutual exclusion that's why it cannot be used in general.

Since, this method is failed at the basic step; hence, there is no need to talk about the other conditions to be fulfilled.

TURN VARIABLE OR STRICT ALTERNATION APPROACH

Turn Variable or Strict Alternation Approach is the software mechanism implemented at user mode. It is a busy waiting solution which can be implemented only for two processes. In this approach, A turn variable is used which is actually a lock.

This approach can only be used for only two processes. In general, let the two processes be P_i and P_j . They share a variable called turn variable. The pseudo code of the program can be given as following.

For Process P_i

1. Non - CS
2. while (turn \neq i);
3. Critical Section
4. turn = j;
5. Non - CS

For Process P_j

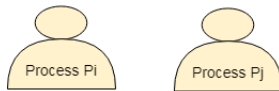
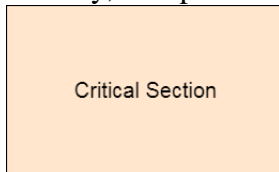
1. Non - CS
2. while (turn \neq j);
3. Critical Section
4. turn = i ;
5. Non - CS

The actual problem of the lock variable approach was the fact that the process was entering in the critical section only when the lock variable is 1. More than one process could see the lock variable as 1 at the same time hence the mutual exclusion was not guaranteed there.

This problem is addressed in the turn variable approach. Now, A process can enter in the critical section only in the case when the value of the turn variable equal to the PID of the process.

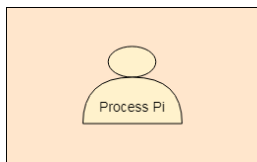
There are only two values possible for turn variable, i or j. if its value is not i then it will definitely be j or vice versa. In the entry section, in general, the process P_i will not enter in the critical section until its value is j or the process P_j will not enter in the critical section until its value is i.

Initially, two processes P_i and P_j are available and want to execute into critical section.

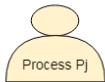


Turn = i

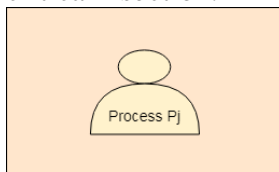
The turn variable is equal to i hence P_i will get the chance to enter into the critical section. The value of P_i remains I until P_i finishes critical section.



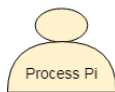
Turn = i



P_i finishes its critical section and assigns j to turn variable. P_j will get the chance to enter into the critical section. The value of turn remains j until P_j finishes its critical section.



Turn = j



ANALYSIS OF STRICT ALTERNATION APPROACH

Mutual Exclusion


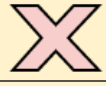


The strict alternation approach provides mutual exclusion in every case. This procedure works only for two processes. The pseudo code is different for both of the processes. The process will only enter when it sees that the turn variable is equal to its Process ID otherwise not Hence No process can enter in the critical section regardless of its turn.

Progress

Progress is not guaranteed in this mechanism. If P_i doesn't want to get enter into the critical section on its turn then P_j got blocked for infinite time. P_j has to wait for so long for its turn since the turn variable will remain 0 until P_i assigns it to j .

Portability

The solution provides portability. It is a pure software mechanism implemented at user mode and doesn't need any special instruction from the Operating System.

Mutual Exclusion	
Progress	
Bounded Waiting	
Portability	

INTERESTED VARIABLE MECHANISM

We have to make sure that the progress must be provided by our synchronization mechanism. In the turn variable mechanism, progress was not provided due to the fact that the process which doesn't want to enter in the critical section does not consider the other interested process as well.

The other process will also have to wait regardless of the fact that there is no one inside the critical section. If the operating system can make use of an extra variable along with the turn variable then this problem can be solved and our problem can provide progress to most of the extent.

Interested variable mechanism makes use of an extra Boolean variable to make sure that the progress is provided.

For Process P_i

1. Non CS
2. $Int[i] = T ;$
3. $while (Int[j] == T) ;$
4. Critical Section
5. $Int[i] = F ;$

For Process P_j

1. Non CS

2. Int [1] = T ;
3. while (Int[i] == T) ;
4. Critical Section
5. Int[j]=F ;

In this mechanism, an extra variable interested is used. This is a Boolean variable used to store the interest of the processes to get enter inside the critical section. A process which wants to enter in the critical section first checks in the entry section whether the other process is interested to get inside. The process will wait for the time until the other process is interested. In exit section, the process makes the value of its interest variable false so that the other process can get into the critical section.

The table shows the possible values of interest variable of both the processes and the process which get the chance in the scenario.

Interest [Pi]	Interest [Pj]	Process which get the chance
True	True	The process which first shows interest.
True	False	Pi
False	True	Pj
False	False	X

Mutual Exclusion

In interested variable mechanism, if one process is interested in getting into the CPU then the other process will wait until it becomes uninterested. Therefore, more than one process can never be present in the critical section at the same time hence the mechanism guarantees mutual exclusion.

Progress

In this mechanism, if a process is not interested in getting into the critical section then it will not stop the other process from getting into the critical section. Therefore the progress will definitely be provided by this method.

Bounded Waiting

To analyze bounded waiting, let us consider two processes Pi and Pj, are the cooperative processes wants to execute in the critical section. The instructions executed by the processes are shown below in relative manner.

Process Pi	Process Pj	Process Pi	Process Pj
------------	------------	------------	------------

1. Int [Pi] = True 2. while (Int [Pj] == True); 3. Critical Section	1. Int [Pj] = True 2. while (Int[Pj]==True);	1. Int [Pi] = False 2. Int [Pi] = True 3. while (Int [Pj] == True); //waiting for Pj	1. While (Int [Pi] == True); //waiting for Pj
---	---	--	---

Initially, the interest variable of both the processes is false. The process Pi shows the interest to get inside the critical section. It sets its Interest Variable to true and check whether the Pj is also interested or not. Since the other process's interest variable is false hence Pi will get enter into the critical section.

Meanwhile, the process Pi is preempted and Pj is scheduled. Pj is a cooperative process and therefore, it also wants to enter in the critical section. It shows its interest by setting the interest variable to true. It also checks whether the other process is also interested or not. We should notice that Pi is preempted but its interested variable is true that means it needs to further execute in the critical section. Therefore Pj will not get the chance and gets stuck in the while loop.



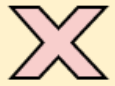

Meanwhile, CPU changes Pi's state from blocked to running. Pi is yet to finish its critical section hence it finishes the critical section and makes an exit by setting the interest variable to False.

Now, a case can be possible when Pi again wants to enter in the critical section and set its interested variable to true and checks whether the interested variable of Pj is true. Here, Pj's interest variable is True hence Pi will get stuck in the while loop and waits for Pj become uninterested. Since, Pj still stuck in the while loop waiting for the Pi' interested variable to become false. Therefore, both the processes are waiting for each other and none of them is getting into the critical section.

This is a condition of deadlock and bounded waiting can never be provided in the case of deadlock. Therefore, we can say that the interested variable mechanism doesn't guarantee deadlock.

Architectural Neutrality

The mechanism is a complete software mechanism executed in the user mode therefore it guarantees portability or architectural neutrality.

Mutual Exclusion	
Progress	
Bounded Waiting	
Portability	

PATERSON SOLUTION

This is a software mechanism implemented at user mode. It is a busy waiting solution can be implemented for only two processes. It uses two variables that are turn variable and interested variable.

The Code of the solution is given below

```

1.     # define N 2
2.     # define TRUE 1
3.     # define FALSE 0
4.     int interested[N] = FALSE;
5.     int turn;
6.     voidEntry_Section (int process)
7.     {
8.         int other;
9.         other = 1-process;
10.        interested[process] = TRUE;
11.        turn = process;
12.        while (interested [other] =True && TURN=process);
13.    }
14.    voidExit_Section (int process)
15.    {
16.        interested [process] = FALSE;
17.    }

```

Till now, each of our solution is affected by one or the other problem. However, the Peterson solution provides you all the necessary requirements such as Mutual Exclusion, Progress, Bounded Waiting and Portability.

Analysis of Peterson Solution

```

1.     voidEntry_Section (int process)
2.     {
3.         1. int other;
4.         2. other = 1-process;

```

```

5.         3. interested[process] = TRUE;
6.         4. turn = process;
7.         5. while (interested [other] =True && TURN=process);
8.     }
9.
10.    Critical Section
11.
12.    voidExit_Section (int process)
13.    {
14.        6. interested [process] = FALSE;
15.    }

```

This is a two process solution. Let us consider two cooperative processes P1 and P2. The entry section and exit section are shown below. Initially, the value of interested variables and turn variable is 0.

Initially process P1 arrives and wants to enter into the critical section. It sets its interested variable to True (instruction line 3) and also sets turn to 1 (line number 4). Since the condition given in line number 5 is completely satisfied by P1 therefore it will enter in the critical section.

P1 → 1 2 3 4 5 CS

Meanwhile, Process P1 got preempted and process P2 got scheduled. P2 also wants to enter in the critical section and executes instructions 1, 2, 3 and 4 of entry section. On instruction 5, it got stuck since it doesn't satisfy the condition (value of other interested variable is still true). Therefore it gets into the busy waiting.

P2 → 1 2 3 4 5

P1 again got scheduled and finish the critical section by executing the instruction no. 6 (setting interested variable to false). Now if P2 checks then it are going to satisfy the condition since other process's interested variable becomes false. P2 will also get enter the critical section.

P1 → 6

P2 → 5 CS

Any of the process may enter in the critical section for multiple numbers of times. Hence the procedure occurs in the cyclic order.

Mutual Exclusion

The method provides mutual exclusion for sure. In entry section, the while condition involves the criteria for two variables therefore a process cannot enter in the critical section until the other process is interested and the process is the last one to update turn variable.

Progress





An uninterested process will never stop the other interested process from entering in the critical section. If the other process is also interested then the process will wait.

Bounded waiting

The interested variable mechanism failed because it was not providing bounded waiting. However, in Peterson solution, A deadlock can never happen because the process which first sets the turn variable will enter in the critical section for sure. Therefore, if a process is preempted after executing line number 4 of the entry section then it will definitely get into the critical section in its next chance.

Portability

This is the complete software solution and therefore it is portable on every hardware.

Mutual Exclusion	
Progress	
Bounded Waiting	
Portability	

SYNCHRONIZATION MECHANISM WITHOUT BUSY WAITING

All the solutions we have seen till now were intended to provide mutual exclusion with busy waiting. However, busy waiting is not the optimal allocation of resources because it keeps CPU busy all the time in checking the while loops condition continuously although the process is waiting for the critical section to become available.

All the synchronization mechanism with busy waiting are also suffering from the priority inversion problem that is there is always a possibility of spin lock whenever there is a process with the higher priority has to wait outside the critical section since the mechanism intends to execute the lower priority process in the critical section.

However these problems need a proper solution without busy waiting and priority inversion.

SLEEP AND WAKE

(Producer Consumer problem)

Assume that we have two system calls as **sleep** and **wake**. The process which calls sleep will get blocked while the process which calls will get waked up. There is a popular example called **producer consumer problem** which is the most popular problem simulating **sleep and wake** mechanism.

The concept of sleep and wake is very simple. If the critical section is not empty then the process will go and sleep. It will be waked up by the other process which is currently executing inside the critical section so that the process can get inside the critical section.

In producer consumer problem, let us say there are two processes, one process writes something while the other process reads that. The process which is writing something is called **producer** while the process which is reading is called **consumer**.

In order to read and write, both of them are using a buffer. The code that simulates the sleep and wake mechanism in terms of providing the solution to producer consumer problem is shown below.

```
#define N 100 //maximum slots in buffer
#define count=0 //items in the buffer
void producer (void)
{
    int item;
    while(True)
    {
        item = produce_item(); //producer produces an item
        if(count == N) //if the buffer is full then the producer will sleep
            Sleep();
        insert_item (item); //the item is inserted into buffer
        countcount=count+1;
        if(count==1) //The producer will wake up the
        //consumer if there is at least 1 item in the buffer
            wake-up(consumer);
    }
}

void consumer (void)
{
    int item;
    while(True)
    {
        {
            if(count == 0) //The consumer will sleep if the buffer is empty.
                sleep();
            item = remove_item();
            countcount = count - 1;
            if(count == N-1) //if there is at least one slot available in the buffer
                //then the consumer will wake up producer
        }
    }
}
```

```

        wake-up(producer);
        consume_item(item); //the item is read by consumer.
    }
}
}

```

The producer produces the item and inserts it into the buffer. The value of the global variable count got increased at each insertion. If the buffer is filled completely and no slot is available then the producer will sleep, otherwise it keep inserting.

On the consumer's end, the value of count got decreased by 1 at each consumption. If the buffer is empty at any point of time then the consumer will sleep otherwise, it keeps consuming the items and decreasing the value of count by 1. The consumer will be waked up by the producer if there is at least 1 item available in the buffer which is to be consumed. The producer will be waked up by the consumer if there is at least one slot available in the buffer so that the producer can write that.

Well, the problem arises in the case when the consumer got preempted just before it was about to sleep. Now the consumer is neither sleeping nor consuming. Since the producer is not aware of the fact that consumer is not actually sleeping therefore it keep waking the consumer while the consumer is not responding since it is not sleeping. This leads to the wastage of system calls. When the consumer get scheduled again, it will sleep because it was about to sleep when it was preempted.

The producer keep writing in the buffer and it got filled after some time. The producer will also sleep at that time keeping in the mind that the consumer will wake him up when there is a slot available in the buffer. The consumer is also sleeping and not aware with the fact that the producer will wake him up.

This is a kind of deadlock where neither producer nor consumer is active and waiting for each other to wake them up. This is a serious problem which needs to be addressed.

Using a flag bit to get rid of this problem

A flag bit can be used in order to get rid of this problem. The producer can set the bit when it calls wake-up on the first time. When the consumer got scheduled, it checks the bit.

The consumer will now get to know that the producer tried to wake him and therefore it will not sleep and get into the ready state to consume whatever produced by the producer.

This solution works for only one pair of producer and consumer, what if there are n producers and n consumers. In that case, there is a need to maintain an integer which can record how many wake-up calls have been made and how many consumers need not sleep. This integer variable is called semaphore.

SEMAPHORE

To get rid of the problem of wasting the wake-up signals, Dijkstra proposed an approach which involves storing all the wake-up calls. Dijkstra states that, instead of giving the wake-up calls directly to the consumer, producer can store the wake-up call in a variable. Any of the consumers can read it whenever it needs to do so.

Semaphore is the variables which stores the entire wake up calls that are being transferred from producer to consumer. It is a variable on which read, modify and update happens automatically in kernel mode.

Semaphore cannot be implemented in the user mode because race condition may always arise when two or more processes try to access the variable simultaneously. It always needs support from the operating system to be implemented.

According to the demand of the situation, Semaphore can be divided into two categories.

1. Counting Semaphore
2. Binary Semaphore or Mutex

COUNTING SEMAPHORE

There are the scenarios in which more than one processes need to execute in critical section simultaneously. However, counting semaphore can be used when we need to have more than one process in the critical section at the same time.

The programming code of semaphore implementation is shown below which includes the structure of semaphore and the logic using which the entry and the exit can be performed in the critical section.

```
struct Semaphore
{
    int value; // processes that can enter in the critical section simultaneously.
    queue type L; // L contains set of processes which get blocked
}
Down (Semaphore S)
{
    SS.value = S.value - 1; //semaphore's value will get decreased when a new
//process enter in the critical section
    if (S.value < 0)
    {
        put_process(PCB) in L; //if the value is negative then
//the process will get into the blocked state.
        Sleep();
    }
    else
```

```

    return;
}
up (Semaphore s)
{
    SS.value = S.value+1; //semaphore value will get increased when
    //it makes an exit from the critical section.
    if(S.value<=0)
    {
        select a process from L; //if the value of semaphore is positive
        //then wake one of the processes in the blocked queue.
        wake-up();
    }
}
}

```

In this mechanism, the entry and exit in the critical section are performed on the basis of the value of counting semaphore. The value of counting semaphore at any point of time indicates the maximum number of processes that can enter in the critical section at the same time.

A process which wants to enter in the critical section first decrease the semaphore value by 1 and then check whether it gets negative or not. If it gets negative then the process is pushed in the list of blocked processes (i.e. q) otherwise it gets enter in the critical section.

When a process exits from the critical section, it increases the counting semaphore by 1 and then checks whether it is negative or zero. If it is negative then that means that at least one process is waiting in the blocked state hence, to ensure bounded waiting, the first process among the list of blocked processes will wake up and gets enter in the critical section.

The processes in the blocked list will get waked in the order in which they slept. If the value of counting semaphore is negative then it states the number of processes in the blocked state while if it is positive then it states the number of slots available in the critical section.

Problem on Counting Semaphore

The questions are being asked on counting semaphore in GATE. Generally the questions are very simple that contains only subtraction and addition.

Wait → Decre → Down → P
 Signal → Inc → Up → V

Ques: A Counting Semaphore was initialized to 12. then 10P (wait) and 4V (Signal) operations were computed on this semaphore. What is the result?

S = 12 (initial)
 10 p (wait) :
 $SS = S - 10 = 12 - 10 = 2$
 then 4 V :
 $SS = S + 4 = 2 + 4 = 6$

BINARY SEMAPHORE OR MUTEX

In counting semaphore, Mutual exclusion was not provided because we have the set of processes which required to execute in the critical section simultaneously.

However, Binary Semaphore strictly provides mutual exclusion. Here, instead of having more than 1 slots available in the critical section, we can only have at most 1 process in the critical section. The semaphore can have only two values, 0 or 1.

```
StructBsemaphore
{
    enum Value(0,1); //value is enumerated data type which can only have two values 0 or 1.
    Queue type L;
}
/* L contains all PCBs corresponding to process
Blocked while processing down operation unsuccessfully.
*/
Down (Bsemaphore S)
{
    if (s.value == 1) // if a slot is available in the
    //critical section then let the process enter in the queue.
    {
        S.value = 0; // initialize the value to 0 so that no other process can read it as 1.
    }
    else
    {
        put the process (PCB) in S.L; //if no slot is available
        //then let the process wait in the blocked queue.
        sleep();
    }
}
Up (Bsemaphore S)
{
    if (S.L is empty) //an empty blocked processes list implies that no process
    //has ever tried to get enter in the critical section.
```

```

{
  S.Value =1;
}
else
{
  Select a process from S.L;
  Wakeup(); // if it is not empty then wake the first process of the blocked queue.
}
}

```

DEADLOCK

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

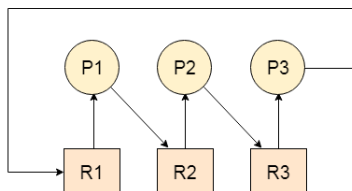
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



Difference between Starvation and Deadlock

Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got	Starvation is a situation where the low

	blocked and no process proceeds	priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

Necessary conditions for Deadlocks

1. Mutual Exclusion

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

2. Hold and Wait

A process waits for some resources while holding another resource at the same time.

3. No preemption

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. Circular Wait

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

STRATEGIES FOR HANDLING DEADLOCK

1. Deadlock Ignorance

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.

There is always a tradeoff between Correctness and performance. The operating systems like Windows and Linux mainly focus upon performance. However, the performance of the system decreases if it uses deadlock handling mechanism all the time if deadlock happens 1 out of 100 times then it is completely unnecessary to use the deadlock handling mechanism all the time.

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 preemption 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.

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.

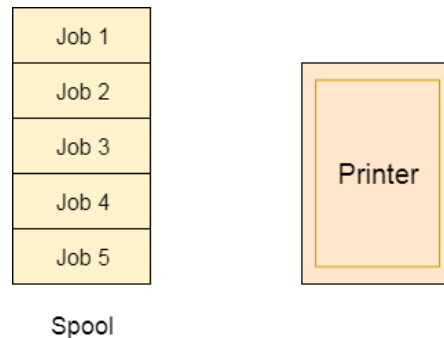
1. Mutual Exclusion

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

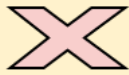
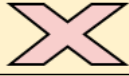


Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

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	

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

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. $E = (7\ 6\ 8\ 4)$
2. $P = (6\ 2\ 8\ 3)$
3. $A = (1\ 4\ 0\ 1)$

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.

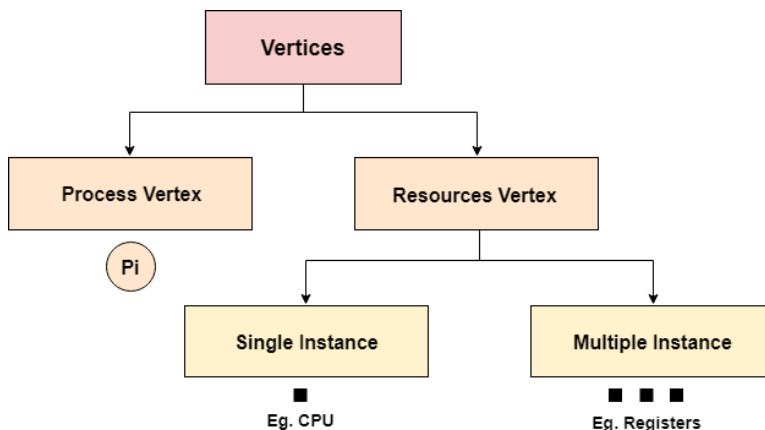
Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use. A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock. If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

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.

RESOURCE ALLOCATION GRAPH

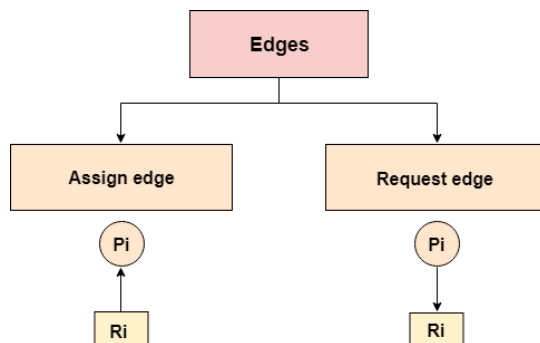
The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

It also contains the information about all the instances of all the resources whether they are available or being used by the processes. In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle.



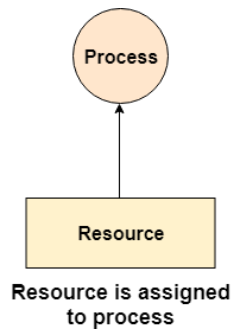
Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them. A resource is shown as assigned to

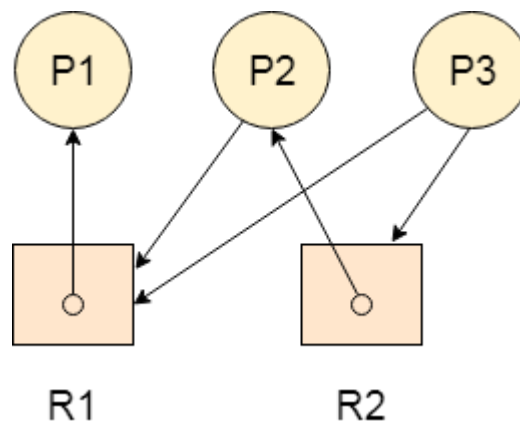
a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process. A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.



Example

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each. According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

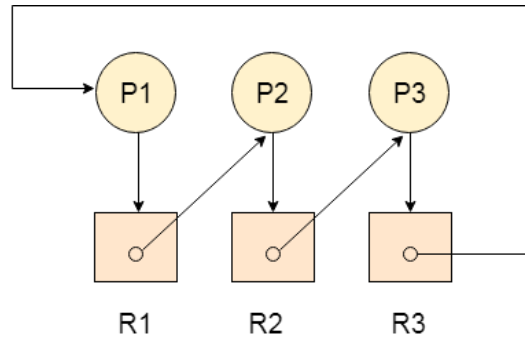
The graph is deadlock free since no cycle is being formed in the graph.



DEADLOCK DETECTION USING RAG

If a cycle is being formed in a Resource allocation graph where all the resources have the single instance then the system is deadlocked. In Case of Resource allocation graph with multi-instanced resource types, Cycle is a necessary condition of deadlock but not the sufficient condition.

The following example contains three processes P1, P2, P3 and three resources R2, R2, R3. All the resources are having single instances each.



If we analyze the graph then we can find out that there is a cycle formed in the graph since the system is satisfying all the four conditions of deadlock.

Allocation Matrix

Allocation matrix can be formed by using the Resource allocation graph of a system. In Allocation matrix, an entry will be made for each of the resource assigned. For Example, in the following matrix, an entry is being made in front of P1 and below R3 since R3 is assigned to P1.

Process	R1	R2	R3
P1	0	0	1
P2	1	0	0
P3	0	1	0

Request Matrix

In request matrix, an entry will be made for each of the resource requested. As in the following example, P1 needs R1 therefore an entry is being made in front of P1 and below R1.

Process	R1	R2	R3
P1	1	0	0
P2	0	1	0
P3	0	0	1

Avial = (0,0,0)

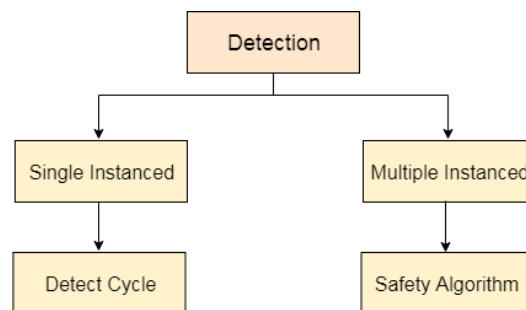
Neither we are having any resource available in the system nor a process going to release. Each of the process needs at least single resource to complete therefore they will continuously be holding each one of them.

We cannot fulfill the demand of at least one process using the available resources therefore the system is deadlocked as determined earlier when we detected a cycle in the graph.

DEADLOCK DETECTION AND RECOVERY

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes.

For Resource

Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

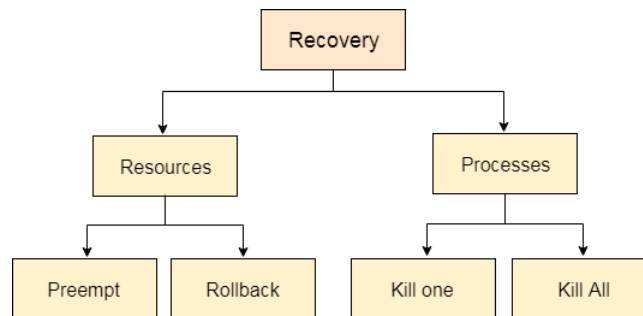
For Process

Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.



MEMORY

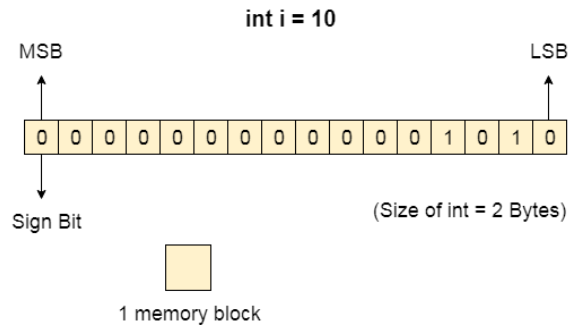
Computer memory can be defined as a collection of some data represented in the binary format. On the basis of various functions, memory can be classified into various categories. A computer device that is capable to store any information or data temporally or permanently, is called storage device.

How Data is being stored in a computer system

In order to understand memory management, we have to make everything clear about how data is being stored in a computer system. Machine understands only binary language that is 0 or 1. Computer converts every data into binary language first and then stores it into the memory.

That means if we have a program line written as **int a = 10** then the computer converts it into the binary language and then store it into the memory blocks.

The representation of **inti = 10** is shown below.



The binary representation of 10 is 1010. Here, we are considering 32 bit system therefore, the size of int is 2 bytes i.e. 16 bit. 1 memory block stores 1 bit. If we are using signed integer then the most significant bit in the memory array is always a signed bit.

Signed bit value 0 represents positive integer while 1 represents negative integer. Here, the range of values that can be stored using the memory array is -32768 to +32767. well, we can enlarge this range by using unsigned int. in that case, the bit which is now storing the sign will also store the bit value and therefore the range will be 0 to 65,535.

Need for Multi programming

However, The CPU can directly access the main memory, Registers and cache of the system. The program always executes in main memory. The size of main memory affects degree of Multi programming to most of the extant. If the size of the main memory is larger than CPU can load more processes in the main memory at the same time and therefore will increase degree of Multi programming as well as CPU utilization.

Let's consider,

Process Size = 4 MB

Main memory size = 4 MB

The process can only reside in the main memory at any time.

If the time for which the process does IO is P,

Then,

CPU utilization = (1-P)

let's say,

P = 70%

CPU utilization = 30 %

Now, increase the memory size, Let's say it is 8 MB.

Process Size = 4 MB

Two processes can reside in the main memory at the same time.

Let's say the time for which, one process does its IO is P,

Then

CPU utilization = (1-P²)

let's say P = 70 %

CPU utilization = (1-0.49) = 0.51 = 51 %

Therefore, we can state that the CPU utilization will be increased if the memory size gets increased.

FIXED PARTITIONING

The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

1. Internal Fragmentation

If the size of the process is lesser than the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

2. External Fragmentation

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

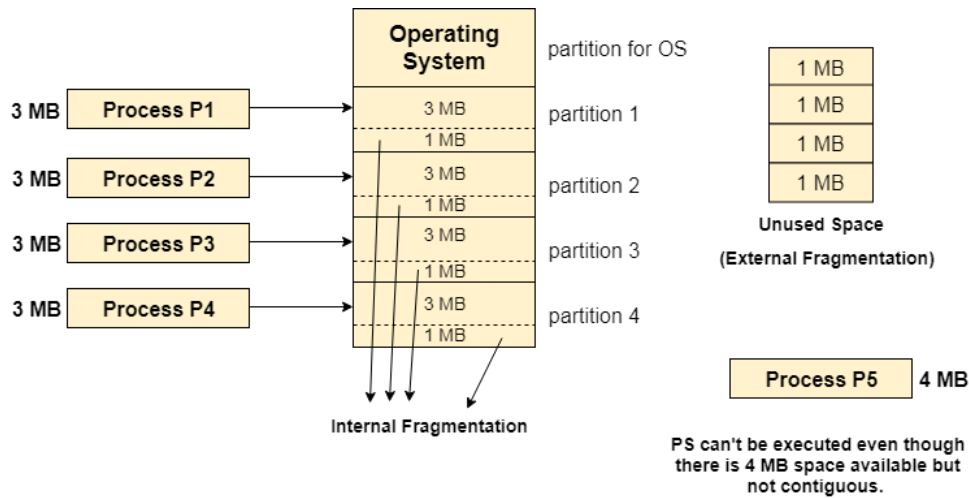
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

3. Limitation on the size of the process

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Degree of multiprogramming is less

By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.



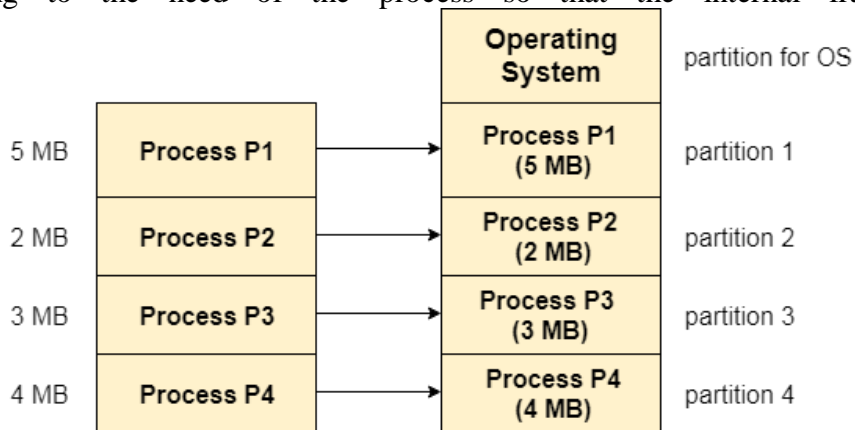
Fixed Partitioning

(Contiguous memory allocation)

DYNAMIC PARTITIONING

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be



Dynamic Partitioning

(Process Size = Partition Size)

avoided.

Advantages of Dynamic Partitioning over fixed partitioning

1. No Internal Fragmentation

Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

2. No Limitation on the size of the process

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

3. Degree of multiprogramming is dynamic

Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

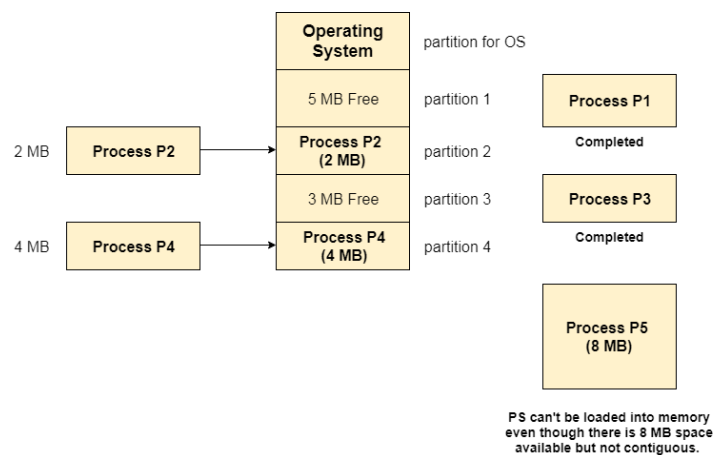
Disadvantages of dynamic partitioning

External Fragmentation

Absence of internal fragmentation doesn't mean that there will not be external fragmentation.

Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory. After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.

The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.



External Fragmentation in Dynamic Partitioning



Complex Memory Allocation

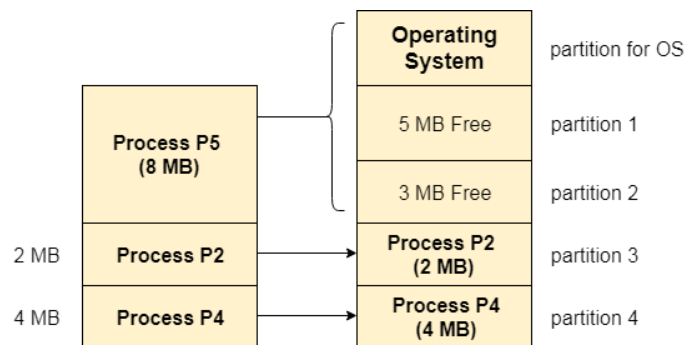
In Fixed partitioning, the list of partitions is made once and will never change but in dynamic partitioning, the allocation and deallocation is very complex since the partition size will be varied every time when it is assigned to a new process. OS has to keep track of all the partitions.

Due to the fact that the allocation and deallocation are done very frequently in dynamic memory allocation and the partition size will be changed at each time, it is going to be very difficult for OS to manage everything.

COMPACTION

The dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems. To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.

We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together. By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.



Now P5 can be loaded into memory because the free space is now made contiguous by compaction

Compaction

As shown in the image above, the process P5, which could not be loaded into the memory due to the lack of contiguous space, can be loaded now in the memory since the free partitions are made contiguous.

Problem with Compaction

The efficiency of the system is decreased in the case of compaction due to the fact that all the free spaces will be transferred from several places to a single place. Huge amount of time is invested for this procedure and the CPU will remain idle for all this time. Despite of the fact that the compaction avoids external fragmentation, it makes system inefficient.

Let us consider that OS needs 6 NS to copy 1 byte from one place to another.

1. 1 B transfer needs 6 NS
2. 256 MB transfer needs $256 \times 2^{20} \times 6 \times 10^{-9}$ secs

hence, it is proved to some extent that the larger size memory transfer needs some huge amount of time that is in seconds.

PARTITIONING ALGORITHMS

There are various algorithms which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.

The explanation about each of the algorithm is given below.

1. First Fit Algorithm

First Fit algorithm scans the linked list and whenever it finds the first big enough hole to store a process, it stops scanning and load the process into that hole. This procedure produces two partitions. Out of them, one partition will be a hole while the other partition will store the process.

First Fit algorithm maintains the linked list according to the increasing order of starting index. This is the simplest to implement among all the algorithms and produces bigger holes as compare to the other algorithms.

2. Next Fit Algorithm

Next Fit algorithm is similar to First Fit algorithm except the fact that, Next fit scans the linked list from the node where it previously allocated a hole.

Next fit doesn't scan the whole list, it starts scanning the list from the next node. The idea behind the next fit is the fact that the list has been scanned once therefore the probability of finding the hole is larger in the remaining part of the list.

Experiments over the algorithm have shown that the next fit is not better then the first fit. So it is not being used these days in most of the cases.

3. Best Fit Algorithm

The Best Fit algorithm tries to find out the smallest hole possible in the list that can accommodate the size requirement of the process.

Using Best Fit has some disadvantages.

1. It is slower because it scans the entire list every time and tries to find out the smallest hole which can satisfy the requirement the process.
2. Due to the fact that the difference between the whole size and the process size is very small, the holes produced will be as small as it cannot be used to load any process and therefore it remains useless.

Despite of the fact that the name of the algorithm is best fit, It is not the best algorithm among all.

3.

4. Worst Fit Algorithm

The worst fit algorithm scans the entire list every time and tries to find out the biggest hole in the list which can fulfill the requirement of the process. Despite of the fact that this algorithm produces the larger holes to load the other processes, this is not the better approach due to the fact that it is slower because it searches the entire list every time again and again.

5. Quick Fit Algorithm

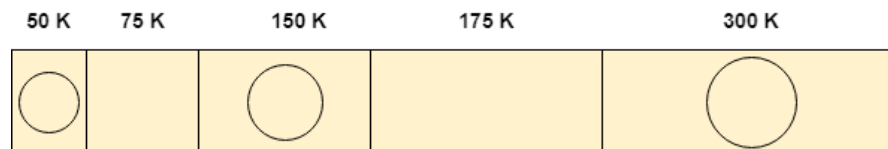
The quick fit algorithm suggests maintaining the different lists of frequently used sizes. Although, it is not practically suggestible because the procedure takes so much time to create the different lists and then expending the holes to load a process.

The first fit algorithm is **the best algorithm** among all because

1. It takes lesser time compare to the other algorithms.
2. It produces bigger holes that can be used to load other processes later on.
3. It is easiest to implement.

Q. Process requests are given as;

25 K , 50 K , 100 K , 75 K



Determine the algorithm which can optimally satisfy this requirement.

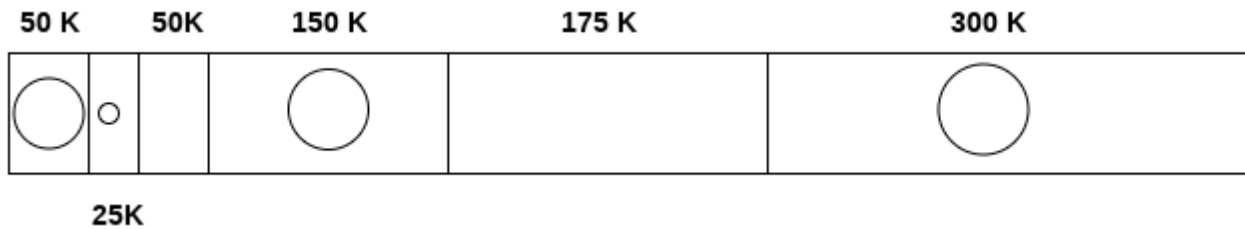
1. First Fit algorithm
2. Best Fit Algorithm
3. Neither of the two
4. Both of them

In the question, there are five partitions in the memory. 3 partitions are having processes inside them and two partitions are holes. Our task is to check the algorithm which can satisfy the request optimally.

Using First Fit algorithm

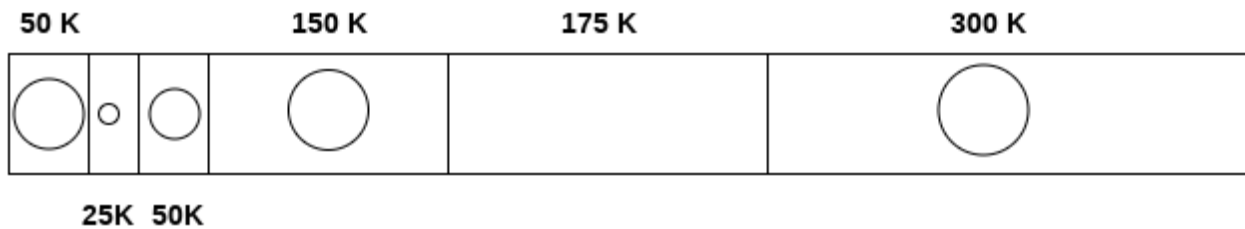
1. 25 K requirement

The algorithm scans the list until it gets first hole which should be big enough to satisfy the request of 25 K. it gets the space in the second partition which is free hence it allocates 25 K out of 75 K to the process and the remaining 50 K is produced as hole.



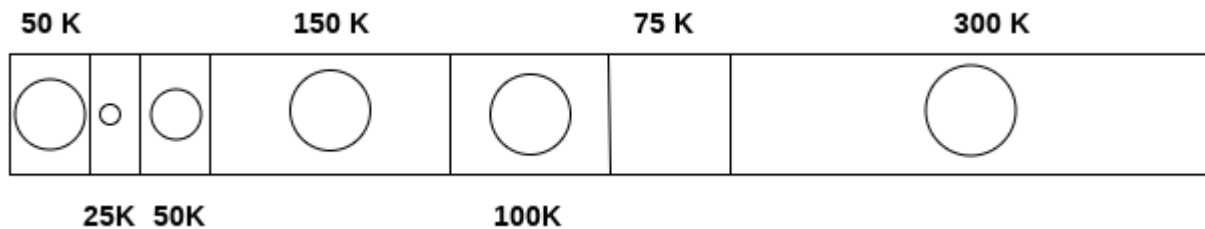
2. 50 K requirement

The 50 K requirement can be fulfilled by allocating the third partition which is 50 K in size to the process. No free space is produced as free space.



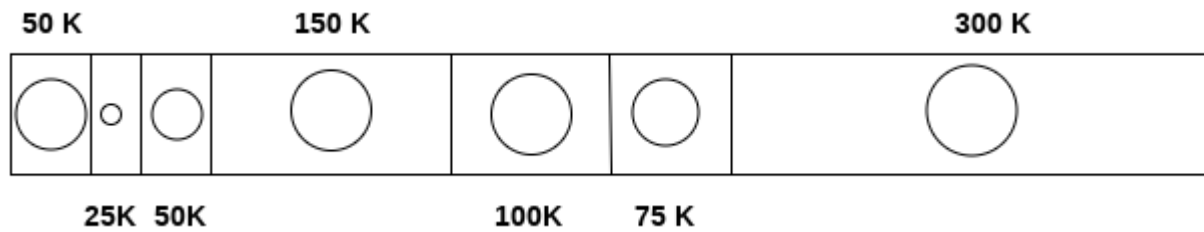
3. 100 K requirement

100 K requirement can be fulfilled by using the fifth partition of 175 K size. Out of 175 K, 100 K will be allocated and remaining 75 K will be there as a hole.



4. 75 K requirement

Since we are having a 75 K free partition hence we can allocate that much space to the process which is demanding just 75 K space.



Using first fit algorithm, we have fulfilled the entire request optimally and no useless space is remaining.

NEED FOR PAGING

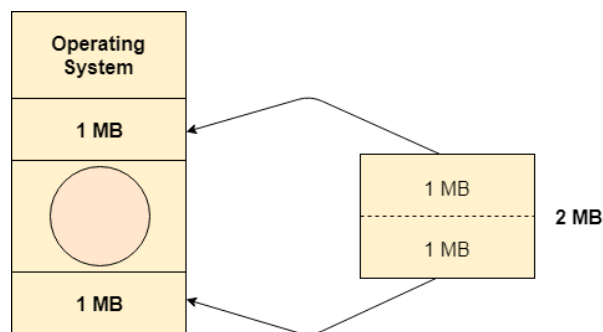
Disadvantage of Dynamic Partitioning

The main disadvantage of Dynamic Partitioning is External fragmentation. Although, this can be removed by Compaction but as we have discussed earlier, the compaction makes the system inefficient. Dynamic and flexible mechanism called paging is a mechanism which can load the processes in the partitions in a more optimal way. Let us discuss a.

Need for Paging

Lets consider a process P1 of size 2 MB and the main memory which is divided into three partitions. Out of the three partitions, two partitions are holes of size 1 MB each. P1 needs 2 MB space in the main memory to be loaded. We have two holes of 1 MB each but they are not contiguous. Although, there is 2 MB space available in the main memory in the form of those holes but that remains useless until it become contiguous. This is a serious problem to address.

We need to have some kind of mechanism which can store one process at different locations of the memory. The Idea behind paging is to divide the process in pages so that, we can store them in the memory at different holes.



The process needs to be divided into two parts to get stored at two different places.

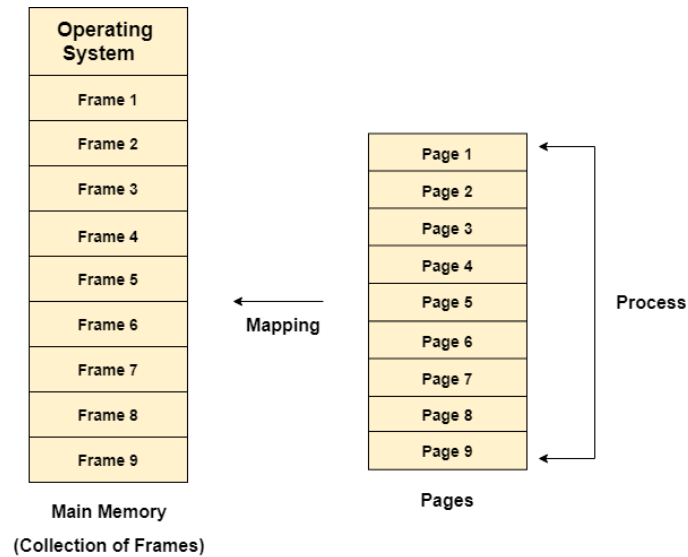
PAGING WITH EXAMPLE

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. The main idea behind the paging

is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes. Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

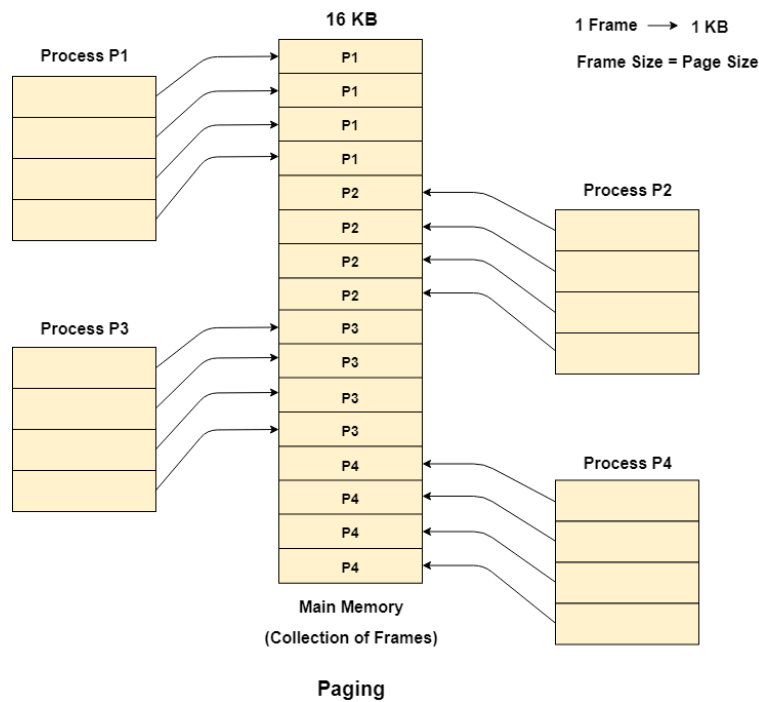
Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.



Example

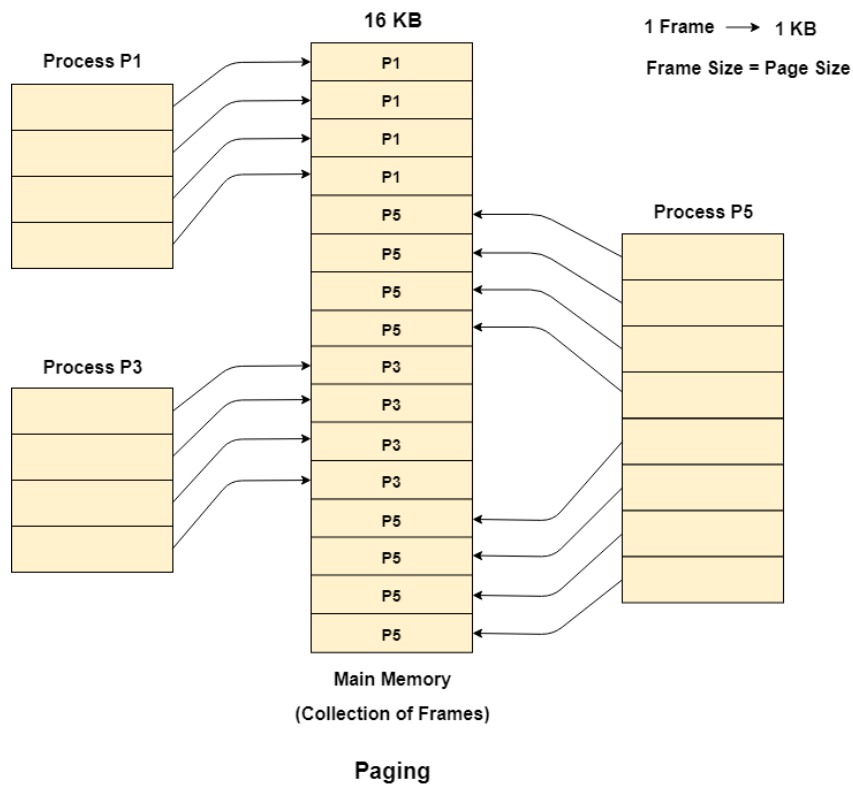
Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each. There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way. Frames, pages and the mapping between the two is shown in the image below.



Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.



MEMORY MANAGEMENT UNIT

The purpose of Memory Management Unit (MMU) is to convert the logical address into the physical address. The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.

When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.

The logical address has two parts.

1. Page Number
2. Offset

Memory management unit of OS needs to convert the page number to the frame number.

Example

Considering the above image, let's say that the CPU demands 10th word of 4th page of process P3. Since the page number 4 of process P1 gets stored at frame number 9 therefore the 10th word of 9th frame will be returned as the physical address.

PHYSICAL AND LOGICAL ADDRESS SPACE

Physical Address Space

Physical address space in a system can be defined as the size of the main memory. It is really important to compare the process size with the physical address space. The process size must be less than the physical address space.

Physical Address Space = Size of the Main Memory

If, physical address space = 64 KB = 2^6 KB = $2^6 \times 2^{10}$ Bytes = 2^{16} bytes

Let us consider,

word size = 8 Bytes = 2^3 Bytes

Hence,

Physical address space (in words) = $(2^{16}) / (2^3) = 2^{13}$ Words

Therefore,

Physical Address = 13 bits

In General,

If, Physical Address Space = N Words

then, Physical Address = $\log_2 N$

Logical Address Space

Logical address space can be defined as the size of the process. The size of the process should be less enough so that it can reside in the main memory.

Let's say,

Logical Address Space = 128 MB = $(2^7 \times 2^{20})$ Bytes = 2^{27} Bytes

Word size = 4 Bytes = 2^2 Bytes

Logical Address Space (in words) = $(2^{27}) / (2^2) = 2^{25}$ Words

Logical Address = 25 Bits

In general,

If, logical address space = L words

Then, Logical Address = $\log_2 L$ bits

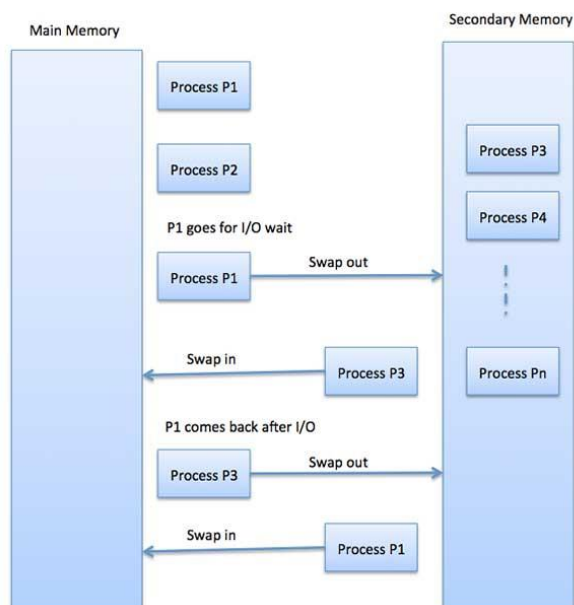
WORD

The Word is the smallest unit of the memory. It is the collection of bytes. Every operating system defines different word sizes after analyzing the n-bit address that is inputted to the decoder and the 2^n memory locations that are produced from the decoder.

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.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction.**



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.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

$$\begin{aligned} & 2048\text{KB} / 1024\text{KB per second} \\ & = 2 \text{ seconds} \\ & = 2000 \text{ milliseconds} \end{aligned}$$

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

MEMORY ALLOCATION

Main memory usually has two partitions –

- **Low Memory** – Operating system resides in this memory.
- **High Memory** – User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	<p>Single-partition allocation</p> <p>In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.</p>
2	<p>Multiple-partition allocation</p> <p>In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.</p>

FRAGMENTATION

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

S.N.	Fragmentation & Description
------	-----------------------------

1	<p>External fragmentation</p> <p>Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.</p>
2	<p>Internal fragmentation</p> <p>Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.</p>

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory –

Fragmented memory before compaction



Memory after compaction



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

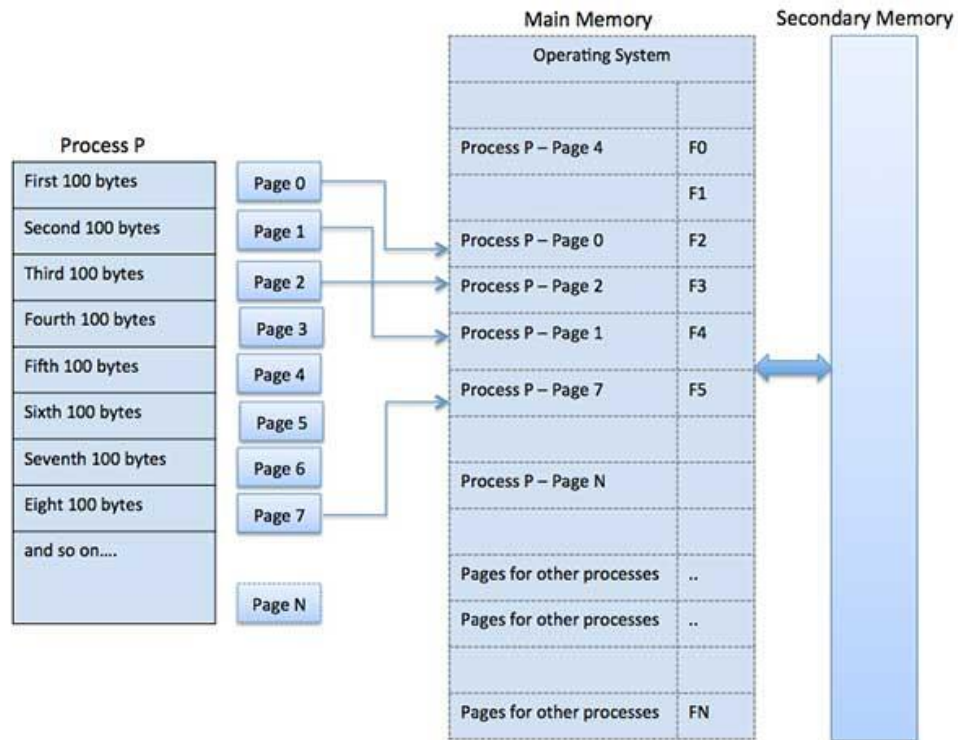
The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

PAGING

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



ADDRESS TRANSLATION

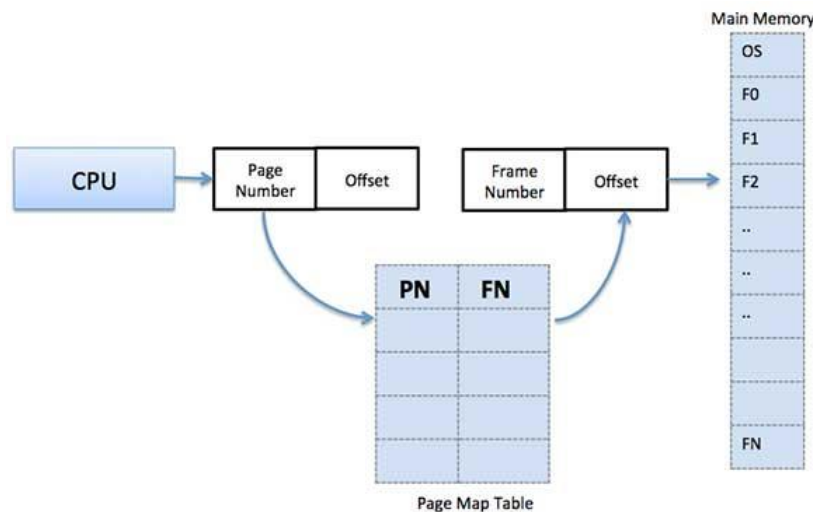
Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging –

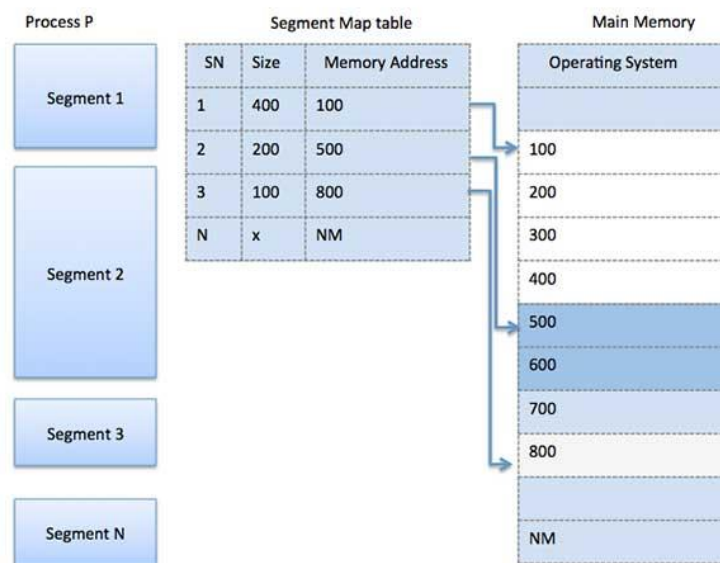
- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

SEGMENTATION

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory. Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



VIRTUAL MEMORY

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory. In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

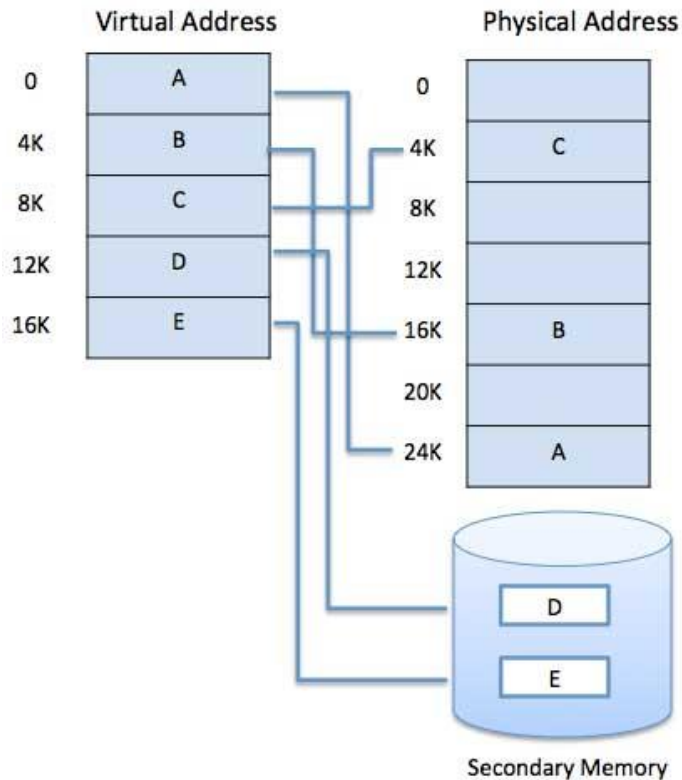
Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory. By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent

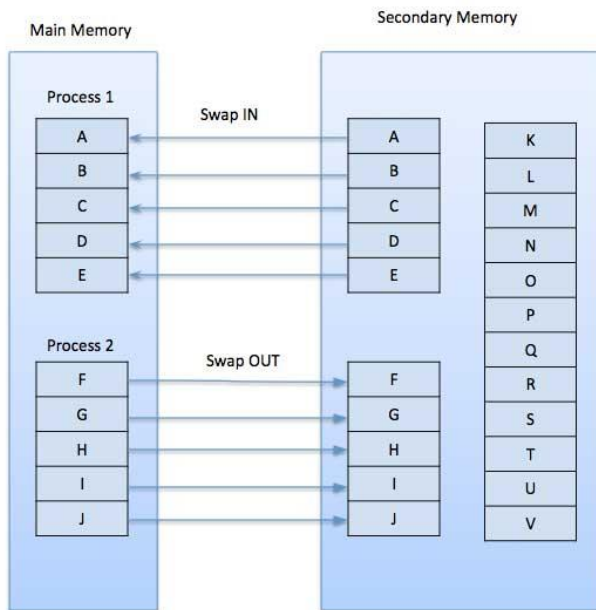
times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.



DEMAND PAGING

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory. A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced. A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory. Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced. While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.



Advantages

Following are the advantages of Demand Paging –

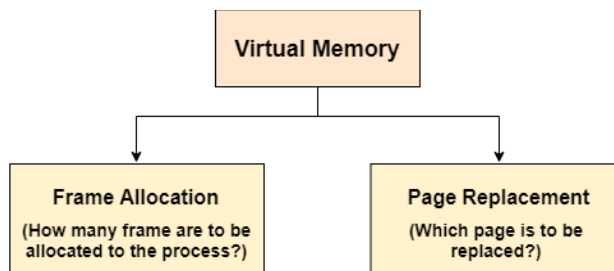
- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

PAGE REPLACEMENT ALGORITHMS

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).



There are two main aspects of virtual memory, Frame allocation and Page Replacement. It is very important to have the optimal frame allocation and page replacement algorithm. Frame allocation is all about how many frames are to be allocated to the process while the page replacement is all about determining the page number which needs to be replaced in order to make space for the requested page.

What If the algorithm is not optimal?

1. if the number of frames which are allocated to a process is not sufficient or accurate then there can be a problem of thrashing. Due to the lack of frames, most of the pages will be residing in the main memory and therefore more page faults will occur.

However, if OS allocates more frames to the process then there can be internal fragmentation.

2. If the page replacement algorithm is not optimal then there will also be the problem of thrashing. If the number of pages that are replaced by the requested pages will be referred in the near future then there will be more number of swap-in and swap-out and therefore the OS has to perform more replacements than usual which cause performance deficiency.

Therefore, the task of an optimal page replacement algorithm is to choose the page which can limit the thrashing.

TYPES OF PAGE REPLACEMENT ALGORITHMS

There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

1. **Optimal Page Replacement algorithm** → this algorithm replaces the page which will not be referred for so long in future. Although it cannot be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.
2. **Least recent used (LRU) page replacement algorithm** → this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. **FIFO** → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rear end of the queue will be replaced on the every page fault.

Q. Consider a main memory with five page frames and the following sequence of page references: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3. which one of the following is true with respect to page replacement policies First-In-First-out (FIFO) and Least Recently Used (LRU)?

- A. Both incur the same number of page faults
- B. FIFO incurs 2 more page faults than LRU
- C. LRU incurs 2 more page faults than FIFO
- D. FIFO incurs 1 more page faults than LRU

Solution: Number of frames = 5

FIFO

According to FIFO, the page which first comes in the memory will first goes out.

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	1	1	1
Frame 4					9	9	9	9	9	9	9	9	2	2	2
Frame 3			2	2	2	2	2	2	8	8	8	8	8	8	8
Frame 2		8	8	8	8	8	8	3	3	3	3	3	3	3	3
Frame 1	3	3	3	3	3	3	6	6	6	6	6	6	6	6	6
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults = 9

Number of hits = 6

LRU

According to LRU, the page which has not been requested for a long time will get replaced with the new one.

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	2	2	2
Frame 4					9	9	9	9	9	9	9	9	9	9	9
Frame 3			2	2	2	2	2	2	8	8	8	8	8	1	1
Frame 2		8	8	8	8	8	6	6	6	6	6	6	6	6	6
Frame 1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Hit	Hit	Miss	Hit	Miss	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Number of Hits = 6

The Number of page faults in both the cases is equal therefore the Answer is **option (A)**.

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm
2. FIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm

OPTIMAL PAGE REPLACEMENT ALGORITHM

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

LRU PAGE REPLACEMENT ALGORITHM

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

FIFO PAGE REPLACEMENT ALGORITHM

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

BELADY'S ANOMALY

In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames. However, Belady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.

This is the strange behavior shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady's Anomaly.

Example :

The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behavior of FIFO algorithm in two cases.

Case 1: Number of frames = 3

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 3			5	5	5	1	1	1	1	1	3	3
Frame 2		1	1	1	0	0	0	0	0	5	5	5
Frame 1	0	0	0	3	3	3	4	4	4	4	4	4
Miss/Hit	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Case 2: Number of frames = 4

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 4				3	3	3	3	3	3	5	5	5
Frame 3			5	5	5	5	5	5	1	1	1	1
Frame 2		1	1	1	1	1	1	0	0	0	0	4
Frame 1	0	0	0	0	0	0	4	4	4	4	3	3
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Number of Page Faults = 10

Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady's Anomaly.

SEGMENTATION

In Operating Systems, 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. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment
2. Limit: It is the length of the segment.

Why Segmentation is required?

Paging is more close to Operating system rather than the User. It divides all the process into the form of pages regardless of the fact that a process can have some relative parts of functions which needs to be loaded in the same page. Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

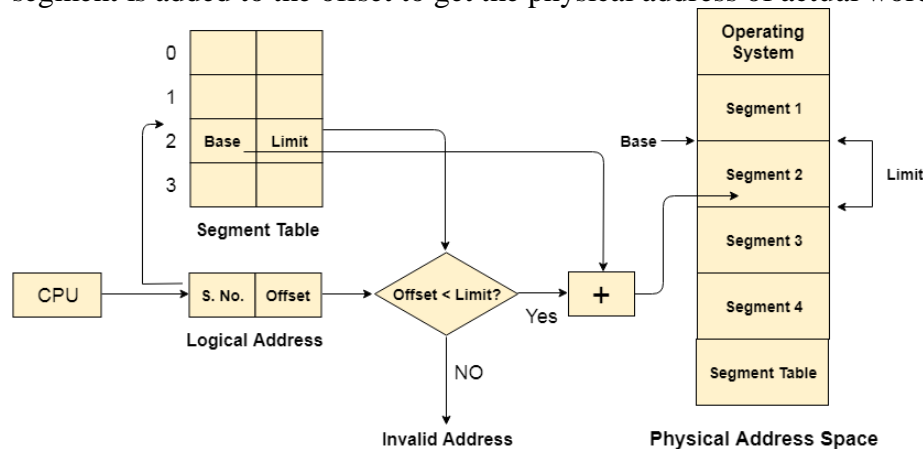
It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

1. Segment Number
2. Offset

The Segment number is mapped to the segment table. 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.

SEGMENTED PAGING

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

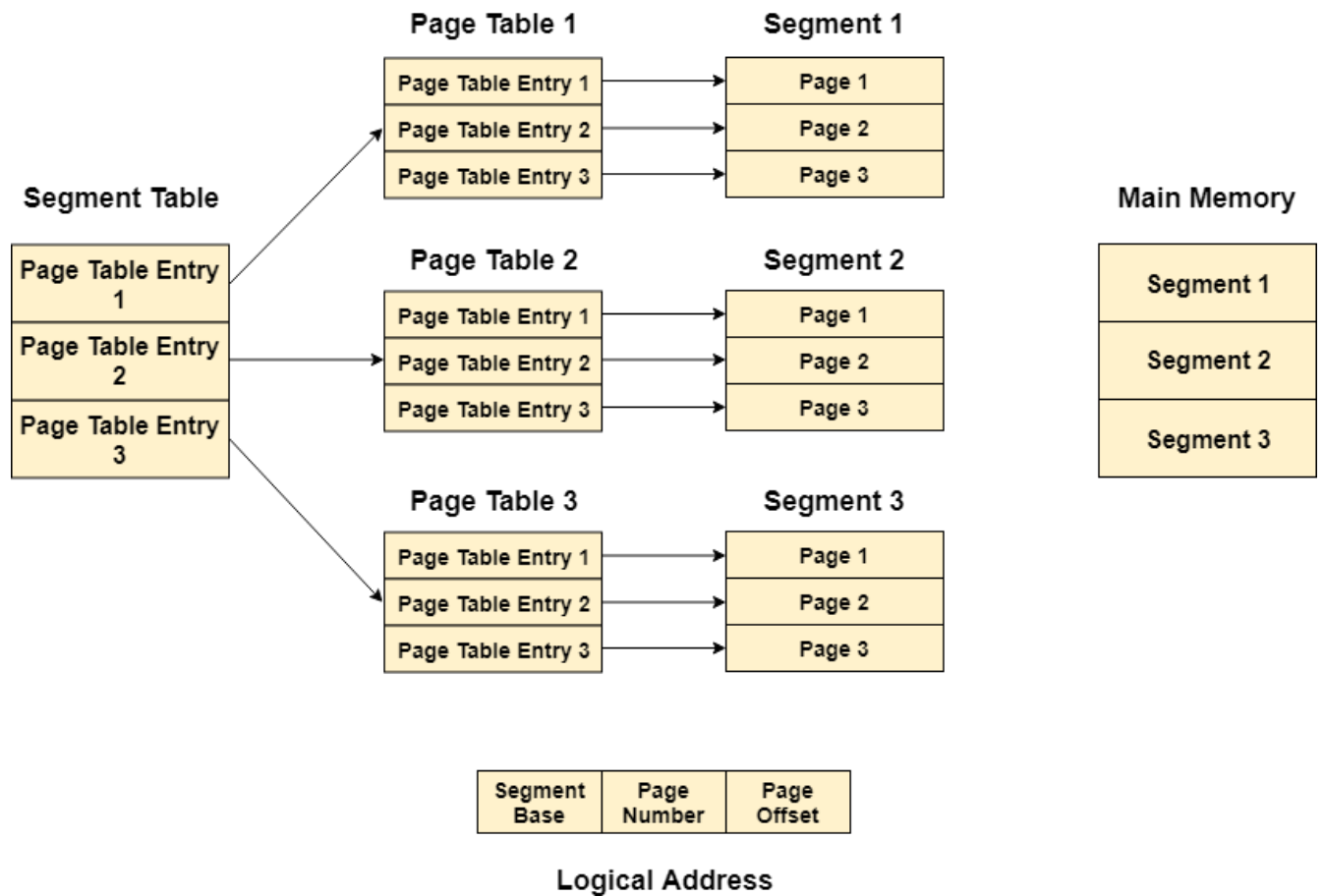
1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

Segment Number → It points to the appropriate Segment Number.

Page Number → It Points to the exact page within the segment

Page Offset → Used as an offset within the page frame

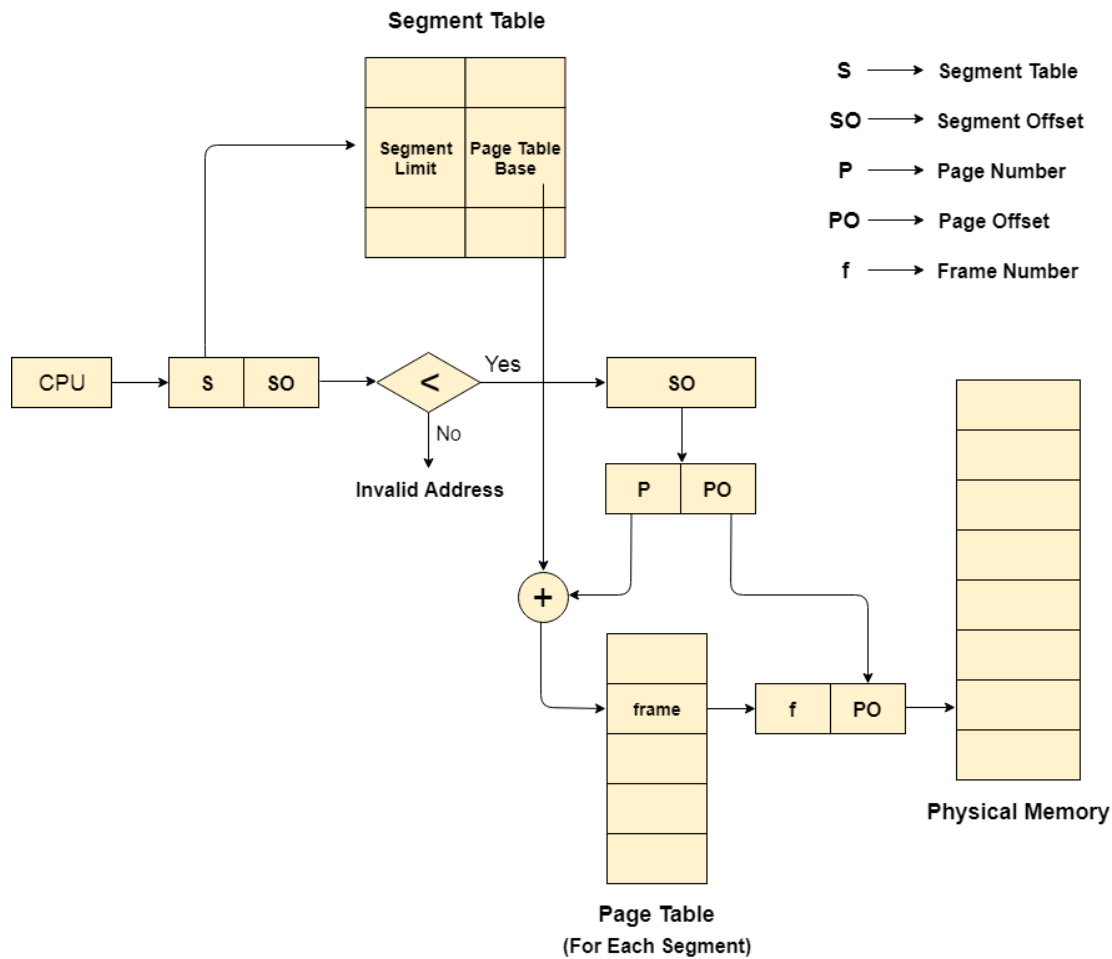
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



Translation of logical address to physical address

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



Advantages of Segmented Paging

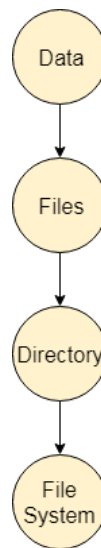
1. It reduces memory usage.
2. Page table size is limited by the segment size.
3. Segment table has only one entry corresponding to one actual segment.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

Disadvantages of Segmented Paging

1. Internal Fragmentation will be there.
2. The complexity level will be much higher as compare to paging.
3. Page Tables need to be contiguously stored in the memory.

FILE

A file can be defined as a data structure which stores the sequence of records. Files are stored in a file system, which may exist on a disk or in the main memory. Files can be simple (plain text) or complex (specially-formatted). The collection of files is known as Directory. The collection of directories at the different levels, is known as File System.



Attributes of the File

1.Name

Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.

2.Identifier

Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt**, A video file can have the extension **.mp4**.

3.Type

In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.

4.Location

In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.

5.Size

The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.

6.Protection

The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.

7. Time and Date

Every file carries a time stamp which contains the time and date on which the file is last modified.

OPERATIONS ON THE FILE

There are various operations which can be implemented on a file. We will see all of them in detail.

1. Create

Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.

2. Write

Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which, the data needs to be written.

3. Read

Every file is opened in three different modes : Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.

4. Re-position

Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as seeking.

5. Delete

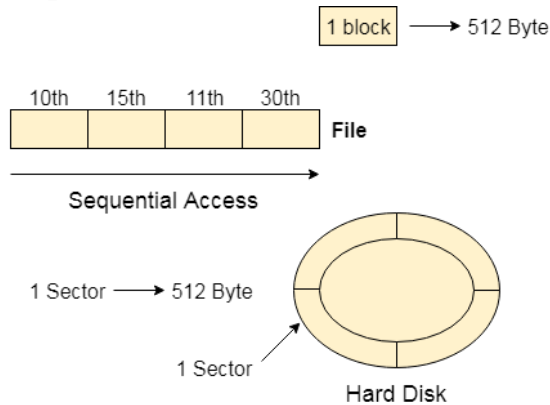
Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.

6. Truncate

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file get replaced.

FILE ACCESS METHODS

Sequential Access



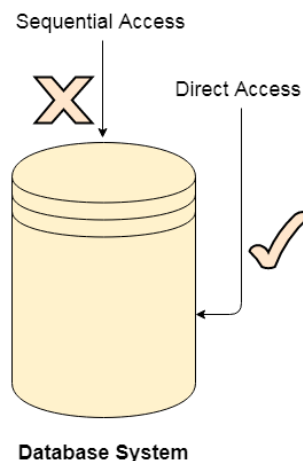
Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.

In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file. Modern word systems do provide the concept of direct access and indexed access but the most used method is sequential access due to the fact that most of the files such as text files, audio files, video files, etc need to be sequentially accessed.

Direct Access

The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.

Suppose every block of the storage stores 4 records and we know that the record we needed is stored in 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record. Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.



Indexed Access

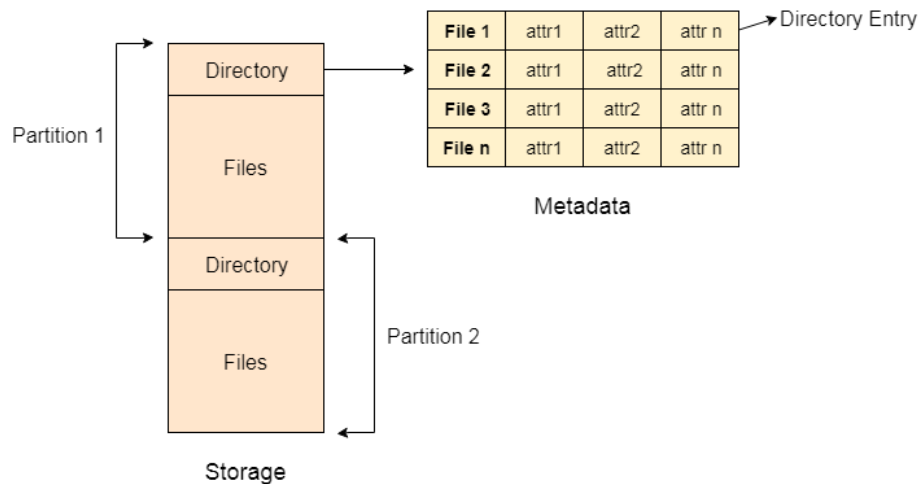
If a file can be sorted on any of the filed then an index can be assigned to a group of certain records. However, A particular record can be accessed by its index. The index is nothing but the address of a record in the file.

In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.

DIRECTORY

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.

To get the benefit of different file systems on the different operating systems, A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks. Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.



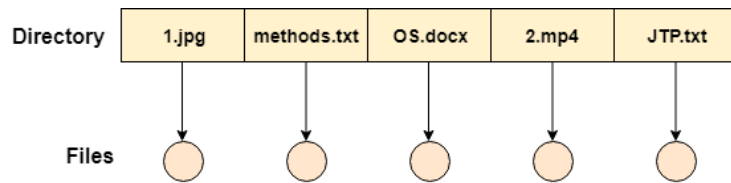
A directory can be viewed as a file which contains the Meta data of the bunch of files.

Every Directory supports a number of common operations on the file:

1. File Creation
2. Search for the file
3. File deletion
4. Renaming the file
5. Traversing Files
6. Listing of files

SINGLE LEVEL DIRECTORY

The simplest method is to have one big list of all the files on the disk. The entire system will contain only one directory which is supposed to mention all the files present in the file system. The directory contains one entry per each file present on the file system.



Single Level Directory

This type of directories can be used for a simple system.

Advantages

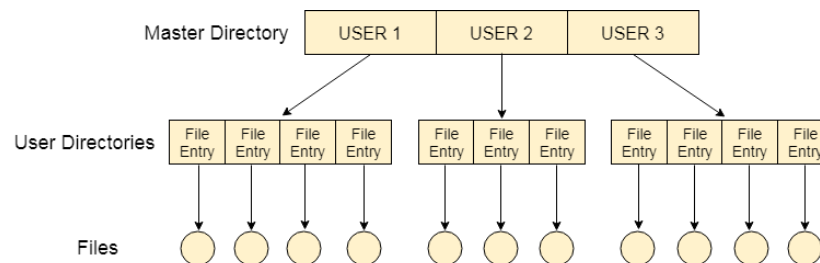
1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

1. We cannot have two files with the same name.
2. The directory may be very big therefore searching for a file may take so much time.
3. Protection cannot be implemented for multiple users.
4. There are no ways to group same kind of files.
5. Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

TWO LEVEL DIRECTORY

In two level directory systems, we can create a separate directory for each user. There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file. The system doesn't let a user to enter in the other user's directory without permission.



Two Level Directory

Characteristics of two level directory system

1. Each files has a path name as */User-name/directory-name/*
2. Different users can have the same file name.
3. Searching becomes more efficient as only one user's list needs to be traversed.
4. The same kind of files cannot be grouped into a single directory for a particular user.

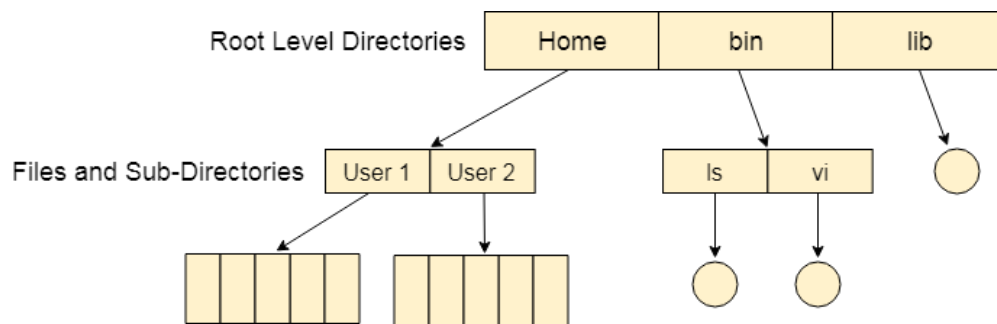
Every Operating System maintains a variable as **PWD** which contains the present directory name (present user name) so that the searching can be done appropriately.

TREE STRUCTURED DIRECTORY

In Tree structured directory system, any directory entry can either be a file or sub directory. Tree structured directory system overcomes the drawbacks of two level directory system. The similar kind of files can now be grouped in one directory.

Each user has its own directory and it cannot enter in the other user's directory. However, the user has the permission to read the root's data but he cannot write or modify this. Only administrator of the system has the complete access of root directory. Searching is more efficient in this directory structure. The concept of current working directory is used. A file can be accessed by two types of path, either relative or absolute.

Absolute path is the path of the file with respect to the root directory of the system while relative path is the path with respect to the current working directory of the system. In tree structured directory systems, the user is given the privilege to create the files as well as directories.



The Structured Directory System

Permissions on the file and directory

A tree structured directory system may consist of various levels therefore there is a set of permissions assigned to each file and directory. The permissions are **R W X** which are regarding reading, writing and the execution of the files or directory. The permissions are assigned to three types of users: owner, group and others.

There is a identification bit which differentiate between directory and file. For a directory, it is **d** and for a file, it is dot (.)

FILE SYSTEMS

File system is the part of the operating system which is responsible for file management. It provides a mechanism to store the data and access to the file contents including data and programs. Some Operating systems treats everything as a file for example Ubuntu.

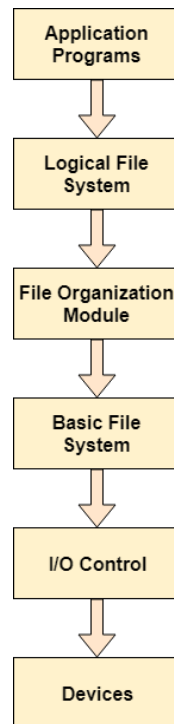
The File system takes care of the following issues

- **File Structure**
We have seen various data structures in which the file can be stored. The task of the file system is to maintain an optimal file structure.
- **Recovering Free space**
Whenever a file gets deleted from the hard disk, there is a free space created in the disk. There can be many such spaces which need to be recovered in order to reallocate them to other files.
- **disk space assignment to the files**
The major concern about the file is deciding where to store the files on the hard disk. There are various disks scheduling algorithm which will be covered later in this tutorial.
- **tracking data location**
A File may or may not be stored within only one block. It can be stored in the non contiguous blocks on the disk. We need to keep track of all the blocks on which the part of the files reside.

FILE SYSTEM STRUCTURE

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file. Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

ALLOCATION METHODS

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

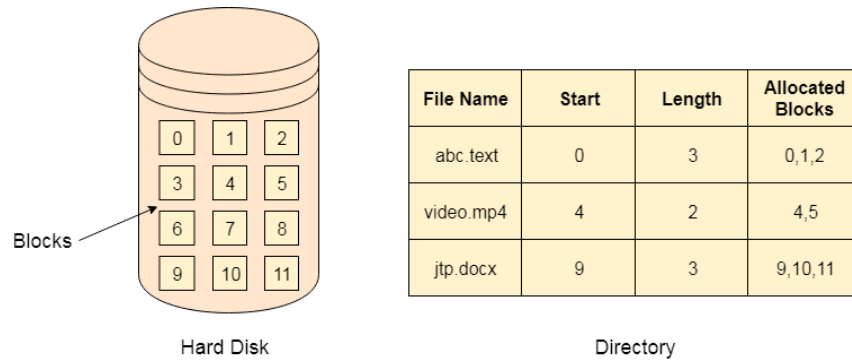
There are following methods which can be used for allocation.

1. Contiguous Allocation.
2. Extents
3. Linked Allocation
4. Clustering
5. FAT
6. Indexed Allocation
7. Linked Indexed Allocation

8. Multilevel Indexed Allocation
9. Inode

CONTIGUOUS ALLOCATION

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation. In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



Contiguous Allocation

Advantages

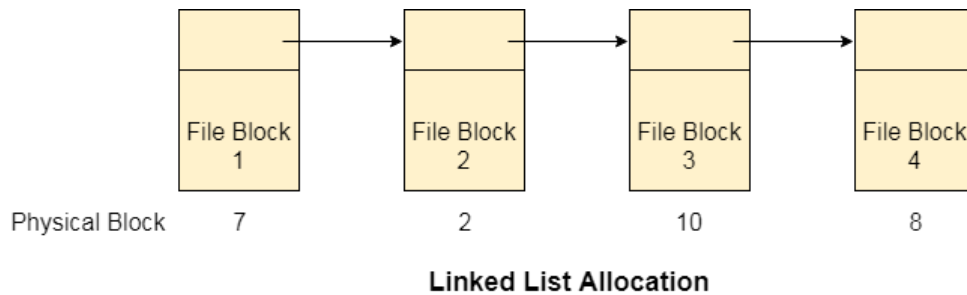
1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

Disadvantages

1. The disk will become fragmented.
2. It may be difficult to have a file grow.

LINKED LIST ALLOCATION

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



Advantages

1. There is no external fragmentation with linked allocation.
2. Any free block can be utilized in order to satisfy the file block requests.
3. File can continue to grow as long as the free blocks are available.
4. Directory entry will only contain the starting block address.

Disadvantages

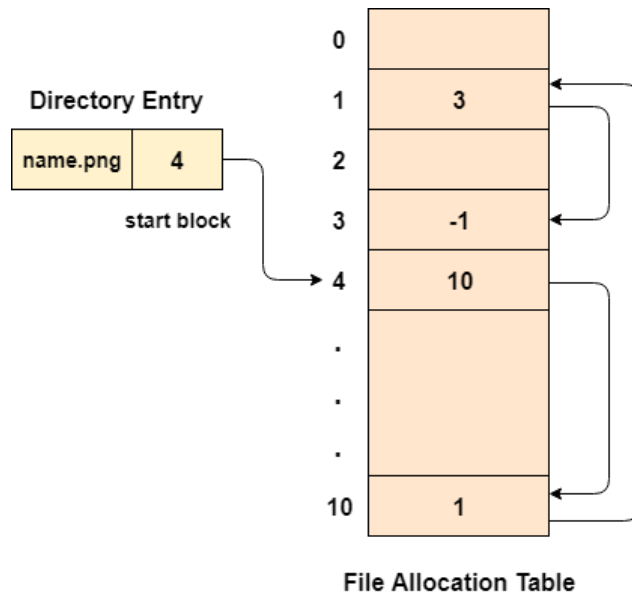
1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block.

FILE ALLOCATION TABLE

The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks.

File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number.

File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block. It simply accesses the file allocation table, read the desired block entry from there and access that block. This is the way by which the random access is accomplished by using FAT. It is used by MS-DOS and pre-NT Windows versions.



Advantages

1. Uses the whole disk block for data.
2. A bad disk block doesn't cause all successive blocks lost.
3. Random access is provided although its not too fast.
4. Only FAT needs to be traversed in each file operation.

Disadvantages

1. Each Disk block needs a FAT entry.
2. FAT size may be very big depending upon the number of FAT entries.
3. Number of FAT entries can be reduced by increasing the block size but it will also increase Internal Fragmentation.

INDEXED ALLOCATION

Limitation of FAT

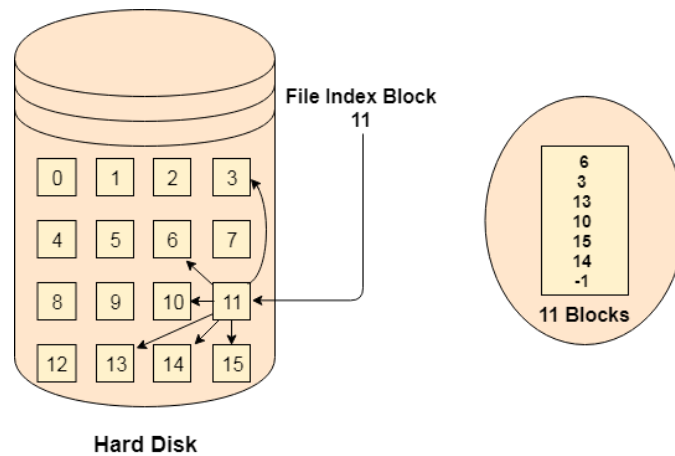
Limitation in the existing technology causes the evolution of a new technology. Till now, we have seen various allocation methods; each of them was carrying several advantages and disadvantages. File allocation table tries to solve as many problems as possible but leads to a drawback. The more the number of blocks, the more will be the size of FAT.

Therefore, we need to allocate more space to a file allocation table. Since, file allocation table needs to be cached therefore it is impossible to have as many space in cache. Here we need a new technology which can solve such problems.

Indexed Allocation Scheme

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't

hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



Advantages

1. Supports direct access
2. A bad data block causes the loss of only that block.

Disadvantages

1. A bad index block could cause the loss of the entire file.
2. Size of a file depends upon the number of pointers, an index block can hold.
3. Having an index block for a small file is totally wasteful.
4. More pointer overhead

DISK SCHEDULING

As we know, a process needs two types of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk. However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

Transfer Time

It is the time taken to transfer the data.

Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughput
- Minimal traveling head time

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

FCFS SCHEDULING ALGORITHM

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Disadvantages

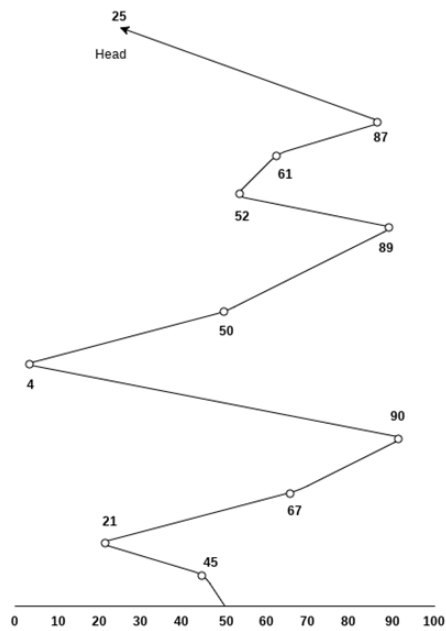
- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Solution



Number of cylinders moved by the head

$$= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)$$
$$= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62$$

$$= 376$$

SSTF SCHEDULING ALGORITHM

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It allows the head to move to the closest track in the service queue.

Disadvantages

- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

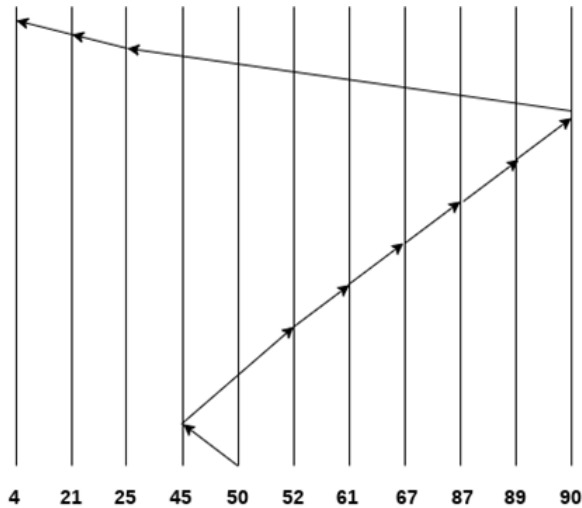
Example

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

Solution:



$$\text{Number of cylinders} = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136$$

Scan Algorithm

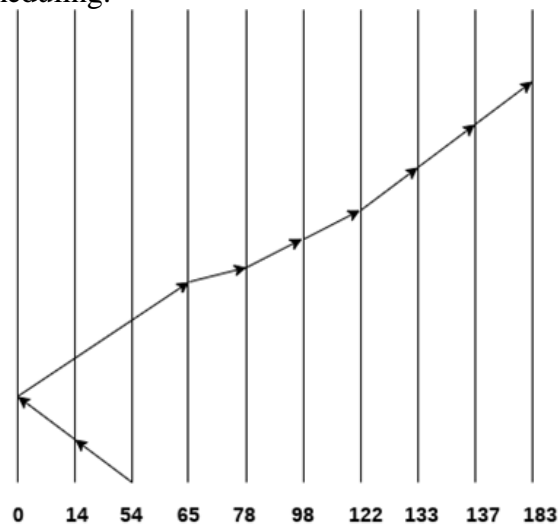
It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path. It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



$$\text{Number of Cylinders} = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237$$

C-SCAN algorithm

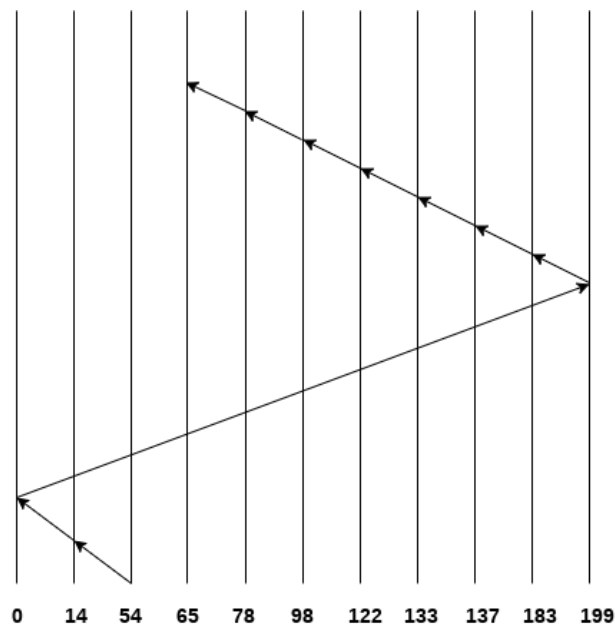
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = $40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387$