UNIT-III

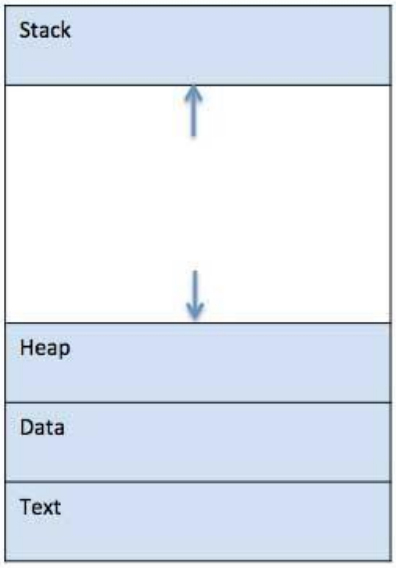
Process Concepts- The Process

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

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

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

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –



Component & Description

Stack -The process Stack contains the temporary data such as method/function parameters, return address and local variables.

Heap- This is dynamically allocated memory to a processduring its run time.

Text- This includes the current activity represented by thevalue of Program Counter and the Contents of the processor'sregisters.

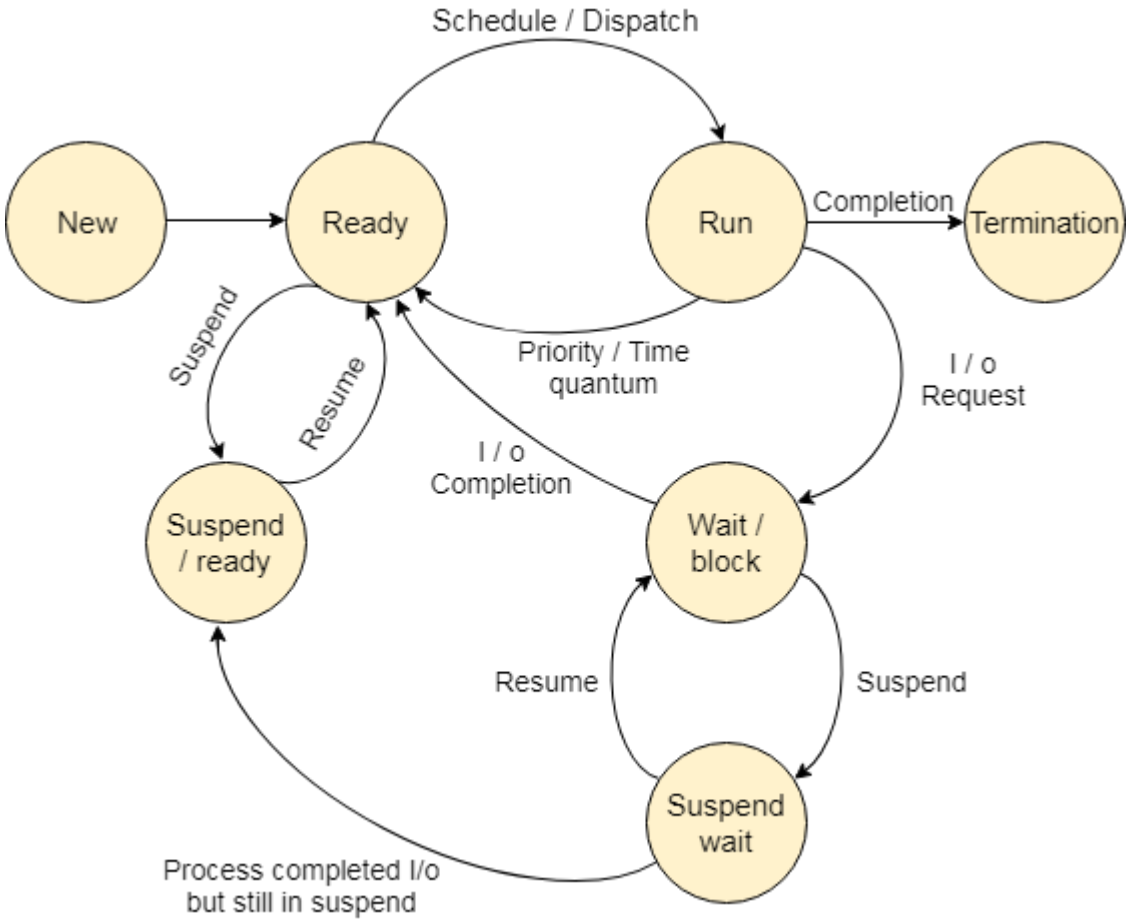
Data-This section contains the global and static variables.

PROCESS STATE

Process Life Cycle:

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

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



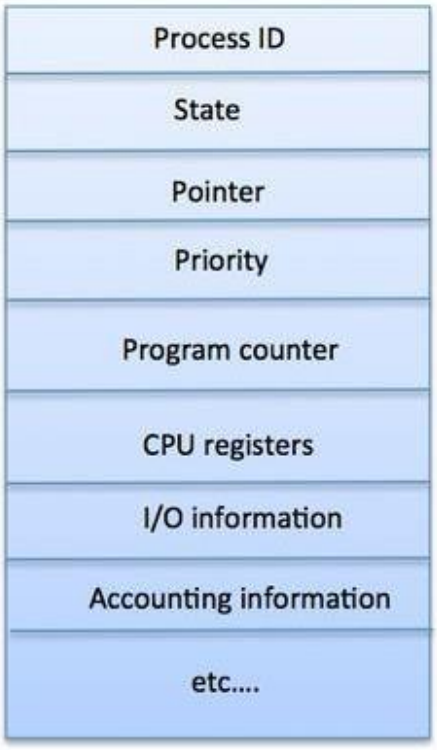
S.N.O	State & Description
1	**Start:-** This is the initial state when a process is first started/created.
2	**Ready:** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running:** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

4	**Waiting:** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit:** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table −

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

OPERATIONS ON PROCESS

There are many operations that can be performed on processes. Some of these are process creation, process preemption, process blocking, and process termination.

These are given in detail as follows −

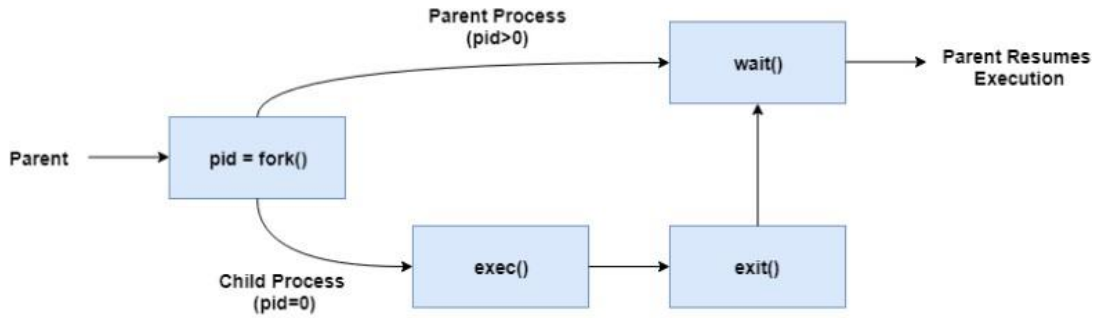
Process Creation

Processes need to be created in the system for different operations. This can be done by the following events −

- User request for process creation
- System initialization
- Execution of a process creation system call by a running process
- Batch job initialization

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files, and environment strings. However, they have distinct address spaces.

A diagram that demonstrates process creation using fork() is as follows –

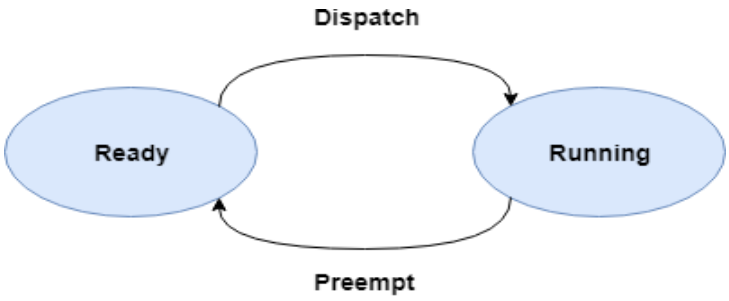


Process Creation Using fork()

Process Preemption

An interrupt mechanism is used in preemption that suspends the process executing currently and the next process to execute is determined by the short-term scheduler. Preemption makes sure that all processes get some CPU time for execution.

A diagram that demonstrates process preemption is as follows –

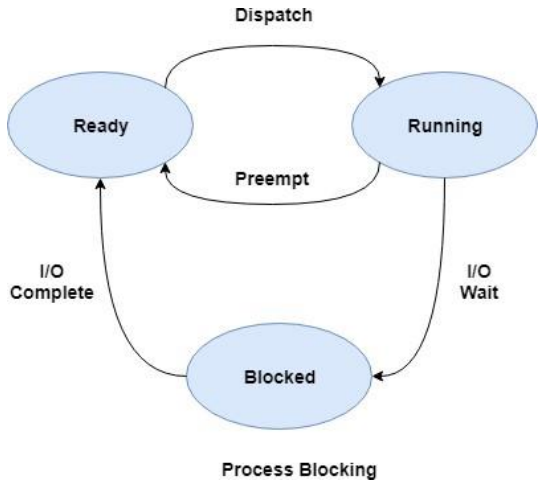


Process Preemption

Process Blocking

The process is blocked if it is waiting for some event to occur. This event may be I/O as the I/O events are executed in the main memory and don't require the processor. After the event is complete, the process again goes to the ready state.

A diagram that demonstrates process blocking is as follows –



Process Blocking

Process Termination

After the process has completed the execution of its last instruction, it is terminated. The resources held by a process are released after it is terminated.

A child process can be terminated by its parent process if its task is no longer relevant. The child process sends its status information to the parent process before it terminates. Also, when a parent process is terminated, its child processes are terminated as well as the child processes cannot run if the parent processes are terminated.

THREADS

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.

4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Advantages:

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages:

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. Kernel threads are supported directly by the operating system. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

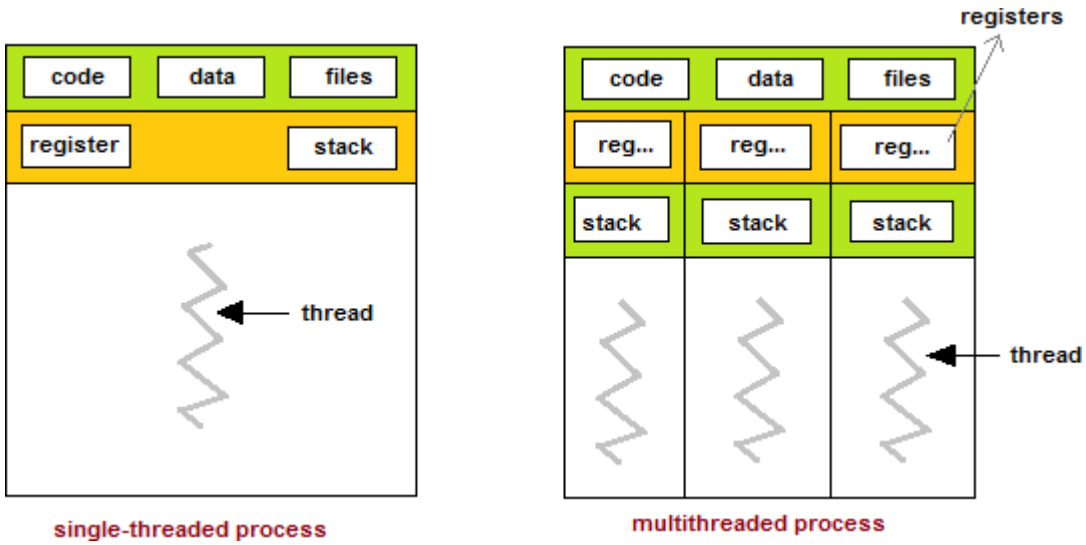
Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

As each thread has its own independent resource for process execution, multpile processes can be executed parallely by increasing number of threads.



Multithreading

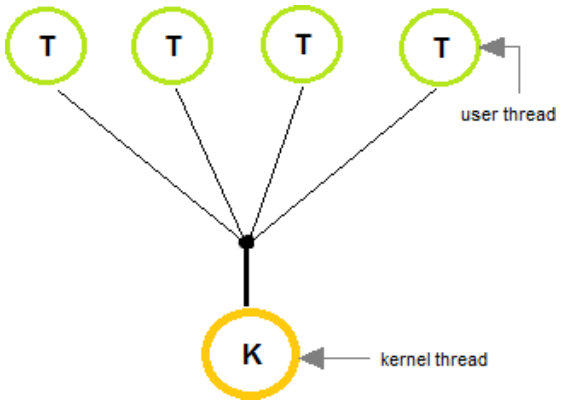
Multithreading Models

The user threads must be mapped to kernel threads, by one of the following strategies.

- Many-To-One Model
- One-To-One Model
- Many-To-Many Model

Many-To-One Model

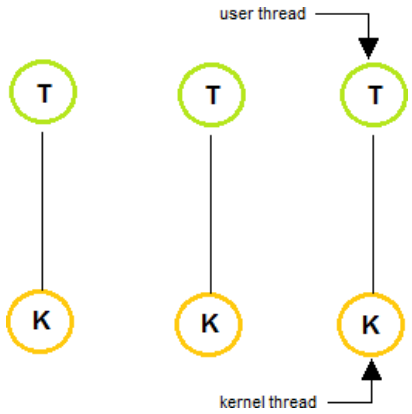
- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.



One-To-One Model

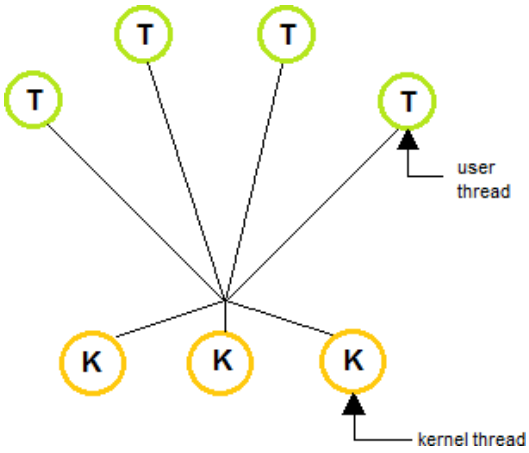
- The one-to-one model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.

- Linux and Windows from 95 to XP implement the one-to-one model for threads.



Many-To-Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.



Benefits of Multithreading

1. Responsiveness
2. Resource sharing, hence allowing better utilization of resources.
3. Economy. Creating and managing threads becomes easier.
4. Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
5. Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

Multithreading Issues

1. **Thread Cancellation**.

   Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is **Asynchronous cancellation**, which terminates the target thread immediately. The other is **Deferred cancellation** allows the target thread to periodically check if it should be cancelled.

2. **Signal Handling**.

   Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

3. **fork() System Call**.

   fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

4. **Security Issues** because of extensive sharing of resources between multiple threads.

There are many other issues that you might face in a multithreaded process, but there are appropriate solutions available for them. Pointing out some issues here was just to study both sides of the coin.

INTER PROCESS COMMUNICATION

A process can be of two types:

Independent process.

Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them.

Processes can communicate with each other through both:

Shared Memory

Message passing

The Figure 1 below shows a basic structure of communication between processes via the shared memory method and via the message passing method.

An operating system can implement both method of communication. First, we will discuss the shared memory methods of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from another process. Process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Let’s discuss an example of communication between processes using shared memory method.

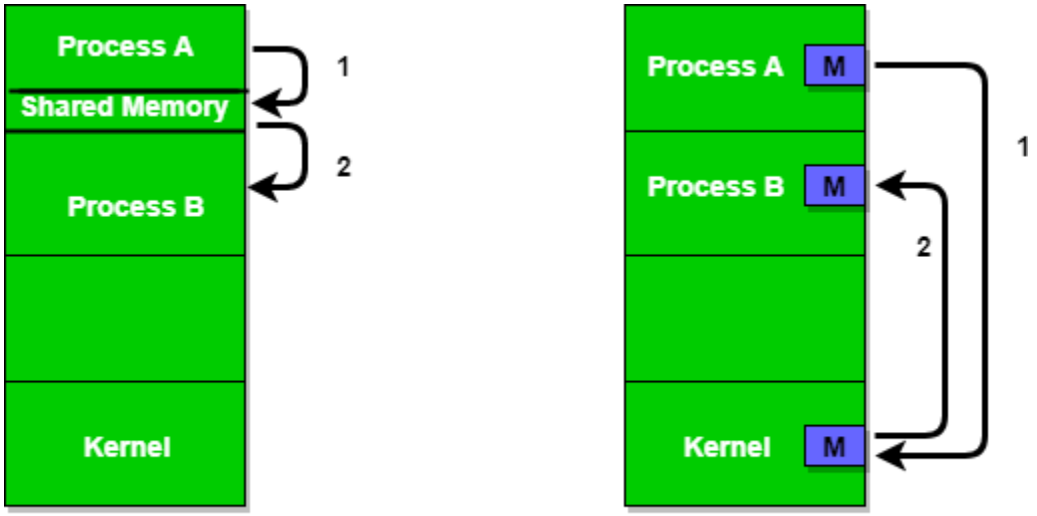


Figure 1 - Shared Memory and Message Passing

ii) Messaging Passing Method

Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

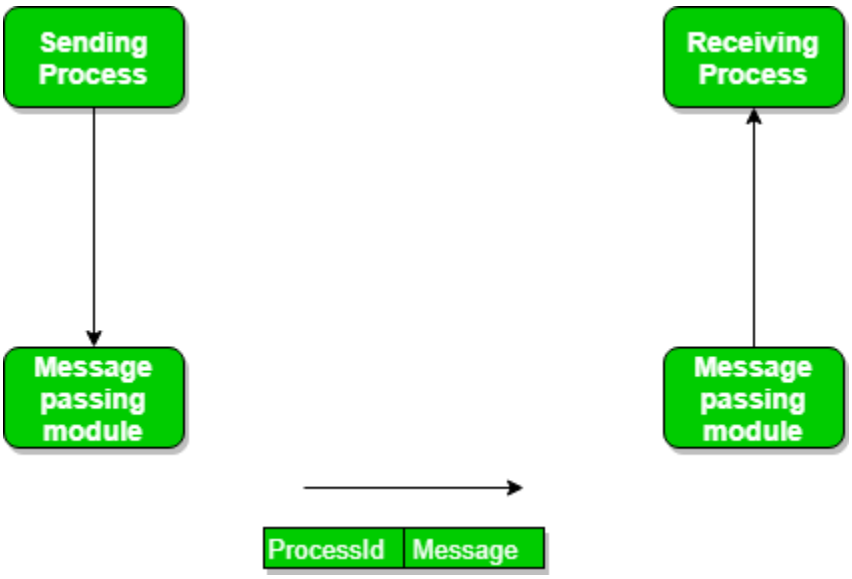
Establish a communication link (if a link already exists, no need to establish it again.)

Start exchanging messages using basic primitives.

We need at least two primitives:

– send(message, destinaion) or send(message)

– receive(message, host) or receive(message)Process Scheduling



CPU SCHEDULING

Definition

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

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

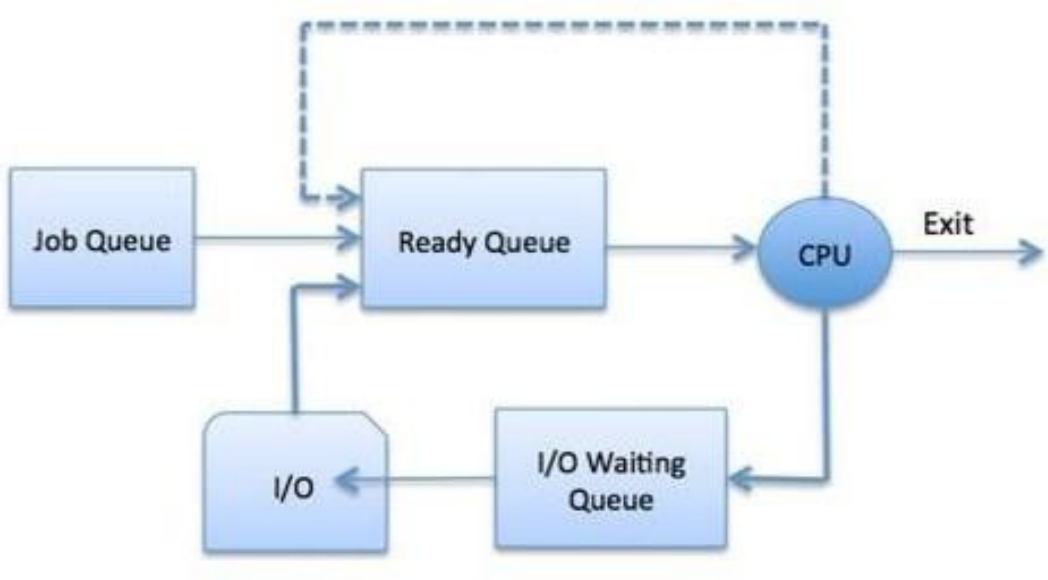
The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

• **Job queue** − This queue keeps all the processes in the system.

• **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

• **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

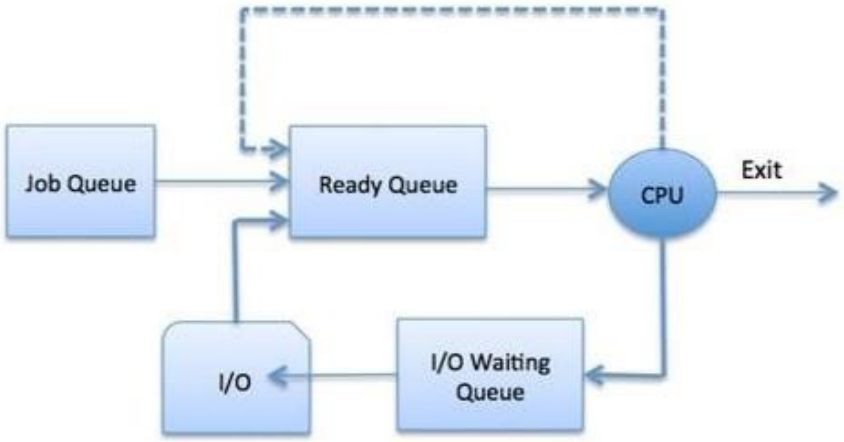
- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.



Short Term Scheduler

It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

SCHEDULING CRITERIA

There are many different criterias to check when considering the "best" scheduling algorithm :

- **CPU utilization**

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput**

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

- **Turnaround time**

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

OR

Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

- **Waiting time**

The sum of the periods spent waiting in the ready queue

OR

amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Waiting Time = Turn Around Time – Burst Time

- **Response time**

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

- **Arrival Time**: Time at which the process arrives in the ready queue.
- **Completion Time**: Time at which process completes its execution.
- **Burst Time**: Time required by a process for CPU execution.

SCHEDULING ALGORITHMS

Scheduling Algorithms

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling

First Come First Serve(FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

CPU Scheduling Algorithms

1. First come first Serve (FCFS) Schedule
2. Shortes Job first (SJF) scheduling Algorithm.
3. Priority Scheduling algorithm
4. Round Robin (RR) Scheduling algorithm
5. Multi level Queue scheduling
6. Multilevel Feedback Queue scheduling

① First come First Serve (FCFS)

→ Assigns cpu to the process which comes first.

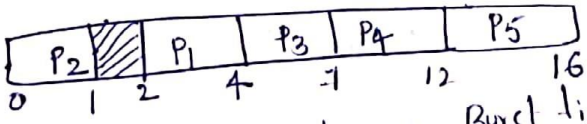
→ Non-preemptive :- In this once the cpu has been allocated to one process thon that process can not be removed until unless it terminates.

→ CPU scheduling :- which takes the process from ready queue and allocates to cpu.

Ex ②

Process	Arrival time (AT) (or) cpu time arrives in ready queue	Burst Time (BT) (or) unit of time to cpu	CT Completion Time	TAT Turn around Time	WT Waiting Time
P1	2	2	4	2	0
P2	0	1	1	1	0
P3	2	3	7	5	2
P4	3	5	12	9	4
P5	4	4	16	12	8

Gantt chart :-



Turnaround Time : waiting time + Burst time

: completion time - arriaval time

waiting time : Turaround time - Burst time

Response time : first time cpu allocates to process - arriaval time

	response time
P1	0
P2	0
P3	2
P4	4
P5	8

avg waiting time = $\frac{0+0+2+4+8}{5} = \frac{14}{5} = 2.8$

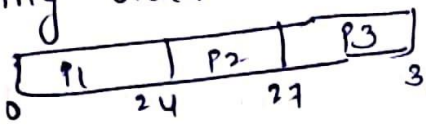
avg turnaround time = $\frac{2+1+5+9+12}{5} = \frac{29}{5} = 5.8$

Example 1 → The time cpu taking to execute a process.

Process	Burst time
P_1	24
P_2	3
P_3	3

Process arriving order is P_1, P_2, P_3

Grantt chart :-



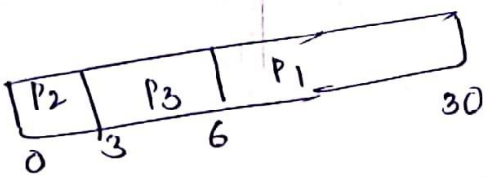
waiting time for $P_1 = 0$, $P_2 = 24$, $P_3 = 27$

avg waiting time : $\frac{0+24+27}{3}$ =

Example :- If the process arraival order is

Process	Burst time
P_2	3
P_3	3
P_1	24

Grantt chart



waiting time for $P_1 = 6$, $P_2 = 0$, $P_3 = 3$

avg waiting time : $\frac{6+0+3}{3} = 3$

Problems with FCFS are :-

1. It is nonpreemptive algorithm
2. Improper process scheduling :- which gives different avg waiting time for different orders of process.
3. Resource utilization in parallel is not possible, which leads to convoy effect & hence poor resource utilization In which whole os slows down due to few slow process.

② Shortest Job First (SJF) Scheduling Algorithm

→ From all available (waiting) processes, it selects the process with the smallest burst time to execute next.

- → Pre-emptive (Shortest Remaining time first (SRTF))
- Non-preemptive (SJF)

→ In non preemptive once a one cpu has been allocated to a process that can not be forcefully removed from the CPU till the termination of that process.

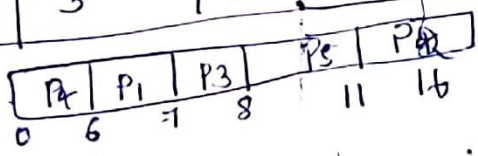
→ while in preemptive you can remove that process from cpu based on some time quantum or priority basis. you can forcefully remove process from cpu.

→ If 2 processes are having same burst time then you will apply FCFS.

	Arrival time	Burst time	Completion time CT	Turnaround time TT = CT − AT	WT = TT − BT	RT = ST − AT
P1	2	1	7	5	4	6−2 = 4
P2	1	5	16	15	10	11−1 = 10
P3	4	1	8	4	3	7−4 = 3
P4	0	6	6	6	0	0−0 = 0
P5	2	3	11	9	6	8−2 = 6

* In non preemption algorithms response time and waiting time both are same

Gronlt chat



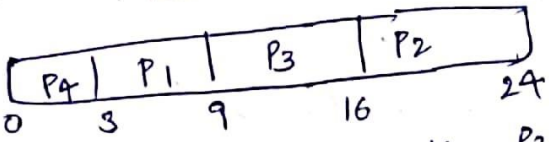
at time 0 only one process is available. so cpu allocates to P4. after completion of P4 now the time is 6 so all the processes arrived time is less than 6 that means now ready queue contains P1, P2, P3, P4. Now we have to check the burst time for allocating to cpu. P1 and P3 having same Burst time now we have to apply FCFS. P1 for arrive first so CPU allocated P1 now. then P3. then P5 (BT = 3) and then P2 (BT = 5).

→ SJF is the best approach to minimize waiting time.

Ex

process	Burst time
P1	6
P2	8
P3	7
P4	3

Gantt Chart



Waiting time for $P_1 = 3$, $P_2 = 16$, $P_3 = 9$, $P_4 = 0$,

avg waiting time : $\frac{3+16+9+0}{4} = 7$.

Disadvantage :-

The total execution time of job must be known before execution while it is not possible to perfectly predict execution.

③ Priority Scheduling :-

→ A Priority number (integer) is associated with each Process

→ The cpu is allocated to the process with highest Priority

→ It is "preemptive" :. If any proces comes with highst Priority than which stops the execution of current proces and assigns cpu to highst priority proces. which leads to "starvation problem".

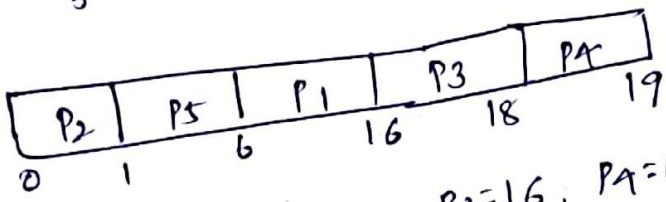
Starvation → low priority processes may never execute.

Solution "Aging" :- as time progresses increases the priority of Process.

Q1

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Grantt chart



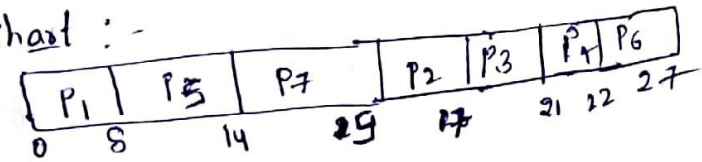
avg waiting time $P_1 = 6$, $P_2 = 0$, $P_3 = 16$, $P_4 = 18$, $P_5 = 1$

$$= \frac{6+0+16+18+1}{5} = \frac{41}{5} = 8.2$$

Q: Non Preemptive

	Priority	AT	BT	CT	TT	WT	RT
P1	3	0	8	8	8	0	0
P2	4	1	2	17	16	14	14
P3	4	3	4	21	18	14	14
P4	5	4	1	22	18	17	17
P5	2	5	6	14	9	3	3
P6	6	6	5	27	21	16	16
P7	1	10	1	15	5	4	4
		total	27				

Grantt chart :-

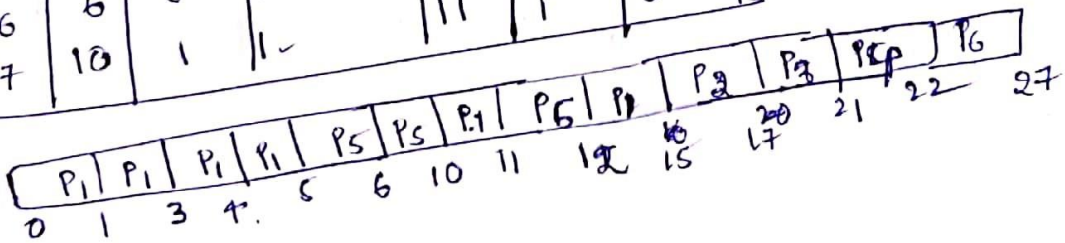


at 8 ns ready queue having P_2, P_3, P_4, P_5, P_6 in which P_5 having highest priority

Q Preemptive

	AT	Priority	BT	C.T	TT	WT	RT
P1	0	3	8(3)	15	15	7	0
P2	1	4	2	17	16	14	14
P3	3	4	4	21	18	14	14
P4	4	5	1	22	18	17	17
P5	5	2	6(5-1)	12	7	1	0
P6	6	6	5	27	21	16	15
P7	10	1	1	11	1	0	0

Grantt chart



① Round Robin (RR) Algorithm (Preemptive)

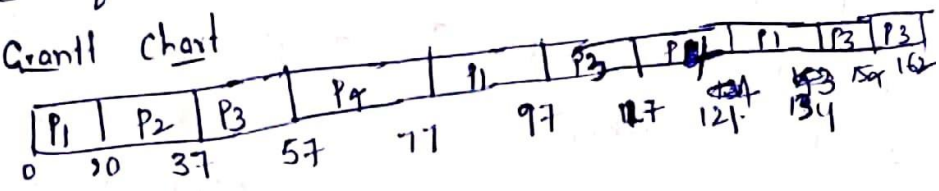
→ The processes are dispatched in FIFO manner but are given limited amount called as quantum time or time slice of CPU.

→ If a process does not complete before its CPU time expires, CPU is preempted and given to next process waiting in queue.

Ex:- RR with time quantum = 20 ms

Process	Burst time
P_1	53 → 33 → 13
P_2	17 ✓
P_3	68 → 48 → 28
P_4	24 → 4 ✓

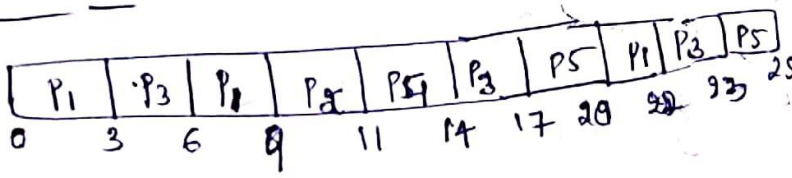
Gantt chart



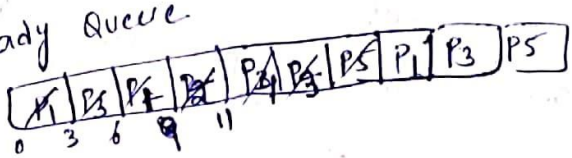
Ex2:- TQ = 3

	AT	BT	CT	TT	WT	RT
P_1	0	5 → 2	22	22	14	0
P_2	5	2 ✓	11	6	4	4
P_3	1	7 → 4 → 1	23	22	15	2
P_4	6	3 ✓	14	8	5	5
P_5	8	5 → 2 ✓	25	17	12	9
				75/5 = 15	50/5 = 10	20/5 = 4

Gantt chart



Ready Queue



Disadvantage

If the time quantum is more then its same like FCFS.

→ If the time quantum is less then its take more context switchings. ie more

→ So that the time quantum should not be more or less

→ the best time quantum is b/n 10 to 100 ms.

scheduler will then proceed to the next process in the ready queue. If the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

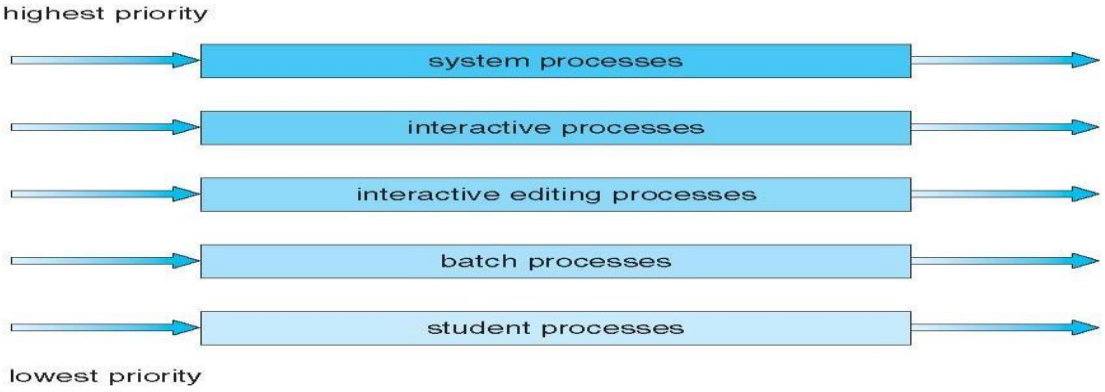
- The average waiting time under the RR policy is often long.
- The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, if the time quantum is extremely large, the RR policy is the same as the FCFS policy. In contrast, if the time quantum is extremely small (say, 1 millisecond), the RR approach can result in a large number of context switches. It creates a processor sharing and creates an appearance that each of n processes has its own processor running at 1/n the speed of the real processor.

Example:

Multilevel Queue Scheduling

- Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.
- A **multilevel queue** scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm. For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR 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.
- Consider the example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:
  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 would be preempted.
- Another possibility is to time-slice among the queues. Here, each queue gets a certain portion of the CPU time, which it can then schedule among its various processes. For instance, in the foreground–background queue example, the foreground queue can be

given 80 percent of the CPU time for RR scheduling among its processes, while the background queue receives 20 percent of the CPU to give to its processes on an FCFSbasis.



Multilevel Feedback Queue Scheduling

- Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, for example, processes do not move from one queue to the other, since processes do not change their foreground or background nature.
- This setup has the advantage of low scheduling overhead, but it is inflexible.
- The **multilevel feedback queue** scheduling algorithm, in contrast, allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues. In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.
- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.

- 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.

SYSTEMCALL INTERFACE for PROCESS MANAGEMENT

There are many different system calls used for the process management.Details of some ofthose system calls are as follows:

• **wait()**

In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent processoccurs with a wait() system call. When the child process completes execution, the control isreturned back to the parent process.

• **exec()**

This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the processare replaced by the new process.

• **fork()**

Processes use the fork() system call to create processes that are a copy of themselves. Thisis one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.

• **exit()**

The exit() system call is used by a program to terminate its execution. In a multithreadedenvironment, this means that the thread execution is complete. The operating system reclaims resources that were used by the process after the exit() system call.

• **kill()**

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit.However, kill system call does not necessary mean killing the process and can have various meanings.

Memory Management

Memory is central to the operation of a modern computer system. Memory consists of a large array of bytes, each with its own address.

A typical instruction-execution cycle, for example, first fetches an instruction from memory. The instruction is then decoded and may cause operands to be fetched from memory. After the instruction has been executed on the operands, results may be stored back in memory.

1. Basic Hardware

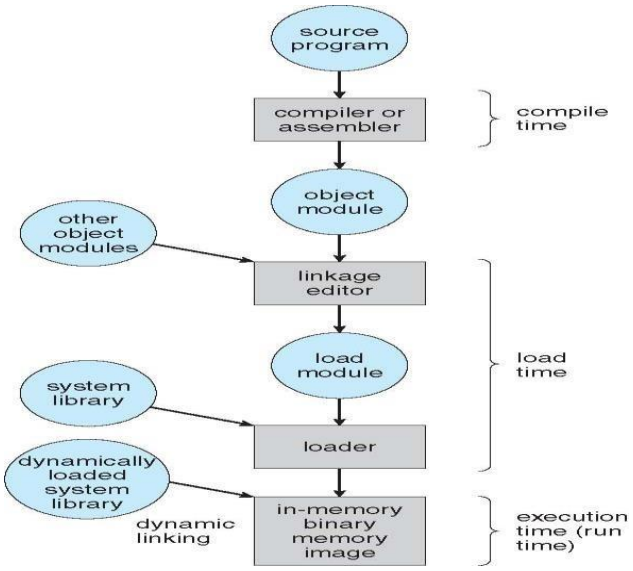
Main memory and the registers built into the processor itself are the only general-purpose storage that the CPU can access directly. Therefore, any instructions in execution, and any data being used by the instructions, must be in one of these direct-access storage devices. If the data are not in memory, they must be moved there before the CPU can operate on them.

2. Address Binding

Usually, a program resides on a disk as a binary executable file. To be executed, the program must be brought into memory and placed within a process. Depending on the memory management in use, the process may be moved between disk and memory during its execution. The processes on the disk that are waiting to be brought into memory for execution form the **input queue**.

Classically, the binding of instructions and data to memory addresses can be done at any step along the way:

- **Compile time**. If you know at compile time where the process will reside in memory, then **absolute code** can be generated. For example, if you know that a user process will reside starting at location *R*, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-format programs are bound at compile time.
- **Load time**. If it is not known at compile time where the process will reside in memory, then the compiler must generate **relocatable code**. In this case, final binding is delayed until load time. If the starting addresses changes, we need only reload the user code to incorporate this changed value.
- **Execution time**. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Most general-purpose operating systems use this method.

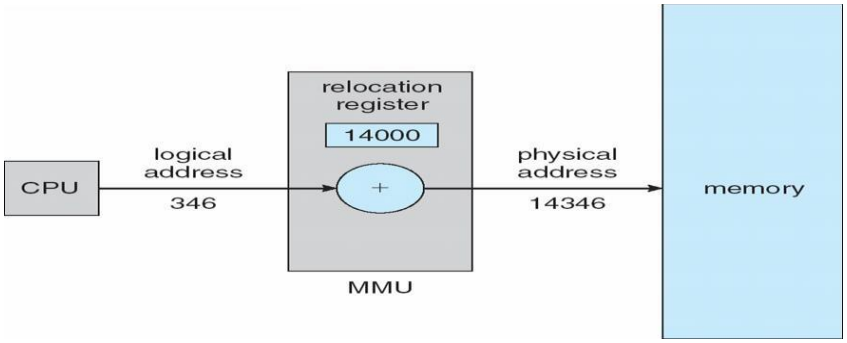


3. Logical Versus Physical Address Space

- An address generated by the CPU is commonly referred to as a **logical address or virtual address**.
- An address seen by the memory unit—that is, the one loaded into the **memory-address register** of the memory—is commonly referred to as a **physical address**.
- The set of all logical addresses generated by a program is a **logical address space**.
- The set of all physical addresses corresponding to these logical addresses is a **physical address space.**

Memory-Management Unit (MMU)

- The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU)**.
- The base register is now called a **relocation register**. The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory.
- For example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location14000; an access to location346 is mapped to location 14346.



Swapping

What is Swapping?

A process must be in memory to be executed. A process, however, can be **swapped** temporarily out of memory to a **backing store** and then brought back into memory for continued execution. Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system, thus increasing the degree of multiprogramming in a system.

1. Standard Swapping

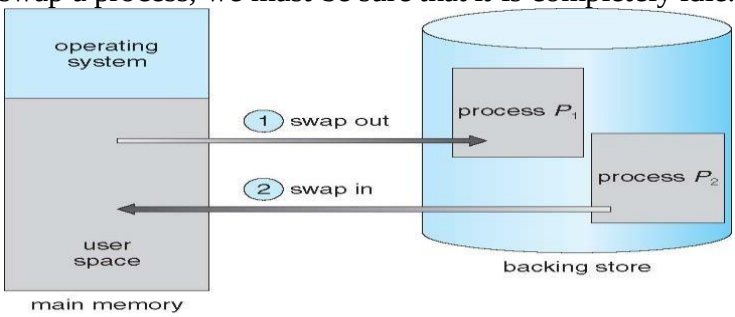
Standard swapping involves moving processes between main memory and a backing store. The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.

Ready Queue: The system maintains a **ready queue** consisting of all processes whose memory images are on the backing store or in memory and are ready to run.

Dispatcher: Whenever the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.

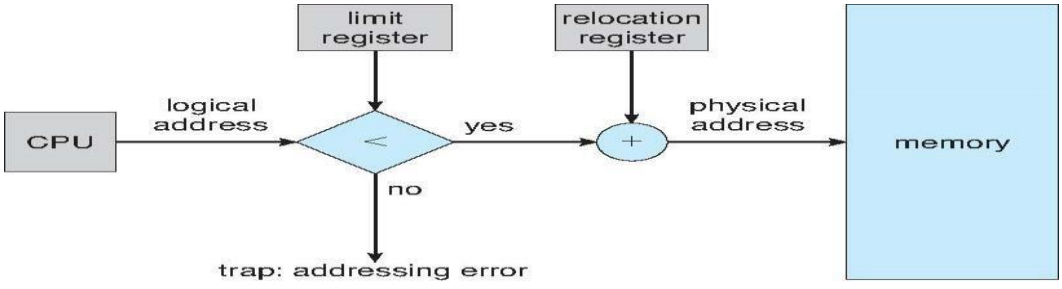
Factors

- The context-switch time in such a swapping system is fairly high.
- The total transfer time is directly proportional to the amount of memory swapped.
- If we want to swap a process, we must be sure that it is completely idle.



Contiguous Memory Allocation

We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In **contiguous memory allocation**, each process is contained in a single section of memory that is contiguous to the section containing the next process.



1. Memory Allocation

Methods for Memory Allocation

a. Fixed-Sized Partitions

- One of the simplest methods for allocating memory is to divide memory into several fixed-sized **partitions**.
- Each partition may contain exactly one process.
- In this **multiple partition method**, 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.
- This method was originally used by the IBM OS/360 operating system (called MFT) but is no longer in use.

b. Variable Sized -Partition

- In the **variable-partition** scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied.
- Initially, all memory is available for user processes and is considered one large block of available memory, a **hole**.
- When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process.
- If the hole is too large, it is split into two parts. One part is allocated to the arriving process; the other is returned to the set of holes.
- When a process terminates, it releases its block of memory, which is then placed back in the set of holes.
- If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole.
- At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes.

Dynamic Storage Allocation Problem (Memory Allocation Techniques)

This concerns how to satisfy a request of size *n* from a list of free holes. There are many solutions to this problem. The **first-fit**, **best-fit**, and **worst-fit** strategies are the ones most commonly used to select a free hole from the set of available holes.

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

Comparison:

- First fit and Best fit are better than Worst fit in terms of decreasing time and storage utilization.
- Neither first fit nor Best fit is clearly better than the other in terms of storage utilization,

but First fit is generally faster.

2. Fragmentation

Memory fragmentation can be internal as well as external.

a. Internal Fragmentation

- The overhead to keep track of this hole will be substantially larger than the hole itself. The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size.
- With this approach, the memory allocated to a process may be slightly larger than the requested memory.
- The difference between these two numbers is **internal fragmentation**—unused memory that is internal to a partition.

b. External Fragmentation

- Both the first-fit and best-fit strategies for memory allocation suffer from **external fragmentation**. As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes.

50-percent rule: Depending on the total amount of memory storage and the average process size, external fragmentation may be a minor or a major problem. Statistical analysis of first fit, for instance, reveals that, even with some optimization, given *N* allocated blocks, another 0.5 *N* blocks will be lost to fragmentation. That is, one-third of memory may be unusable! This property is known as the **50-percent rule**.

Solution to External Fragmentation

a. Compaction

- The goal is to shuffle the memory contents so as to place all free memory together in one large block.
- Compaction is not always possible, however. If relocation is static and is done at assembly or load time, compaction cannot be done. It is possible only if relocation is dynamic and is done at execution time.
- The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive.

b. Noncontiguous logical address space

- This permits the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever such memory is available.
- Two complementary techniques achieve this solution: segmentation and paging.

Virtual memory: Virtual memory is a memory management technique used by modern operating systems to provide an abstraction of more memory to applications than is physically available in the system's physical RAM (Random Access Memory). It allows applications to access and

use more memory than what's physically installed in the computer, by using a combination of RAM and disk storage. This concept provides several benefits, including efficient memory utilization, isolation between processes, and simplification of memory management for programmers. how virtual memory works:

1. Virtual Address Space: Each process running on the system has its own virtual address space. This address space is the range of memory addresses that the process can use. The size of the virtual address space is typically much larger than the amount of physical RAM available.

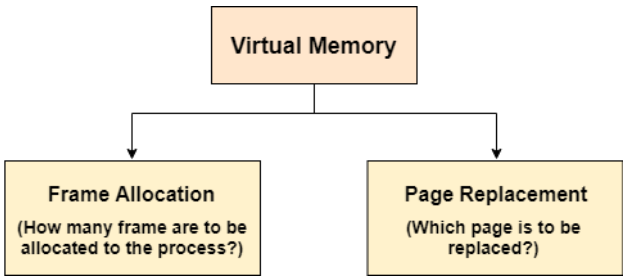
2. Physical Memory (RAM): The actual physical memory available in the system is divided into fixed-size blocks called "pages." Similarly, the virtual address space is divided into fixed-size blocks called "virtual pages."

3. Page Table: The operating system maintains a data structure called the "page table." This table keeps track of the mapping between virtual pages and physical pages. When a process accesses a virtual address, the operating system translates it into a physical address using the page table.

4. Page Faults: When a process accesses a virtual page that is not currently in physical memory (RAM), a page fault occurs. The operating system then brings the required page into RAM from the disk. If there's no free space in RAM, the operating system has to choose a page to evict (remove) using a page replacement algorithm.

5. Swapping: If physical memory becomes congested, the operating system can temporarily move entire pages (also called page frames) to disk storage. This process is known as "swapping." Swapping helps free up space in physical memory for pages that are actively used.

6. Demand Paging: Not all pages are loaded into RAM at once. Instead, only the pages that are actually accessed are brought into memory. This technique is called "demand paging." It helps reduce the initial memory requirements for a process and improves overall system responsiveness.



The important jobs of virtual memory in Operating Systems are two. They are:

- Frame Allocation
- Page Replacement.

Benefits of Virtual Memory:

1. Memory Isolation: Each process believes it has the entire virtual address space to itself, providing memory isolation and security.

2. Effective Use of Resources: Virtual memory enables running more processes concurrently, as not all processes need to fit entirely in RAM. It optimizes the utilization of available resources.

3. Simplifies Programming: Developers can write programs without worrying about the physical memory constraints of the system. They can work with a larger virtual address space.

4. Flexible Memory Allocation: Virtual memory systems allow dynamic allocation of memory to processes, adapting to their changing memory requirements.
However, there are some potential drawbacks to virtual memory, such as increased overhead due to page table management, potential performance degradation when excessive swapping occurs, and increased complexity in memory management algorithms.

Overall, virtual memory is a crucial component of modern operating systems, enabling efficient utilization of system resources and providing a seamless and isolated memory environment for applications.

Paging:

Paging is another memory-management scheme that offers physical address space of a process to be non-contiguous. Paging also avoids external fragmentation and the need for compaction, whereas segmentation does not. Because of its advantages, paging in its various forms is used in most operating systems, from mainframes to smart phones.

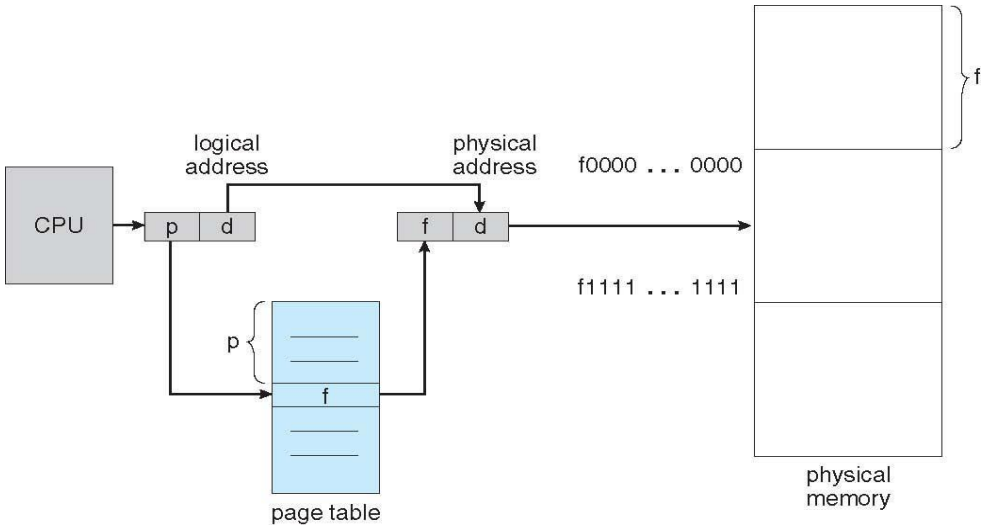
1. **Basic Method**
   - **Frames**: Paging involves breaking physical memory into fixed-sized blocks called **frames.**
   - **Pages:** Breaking logical memory into blocks of the same size called **pages**.

When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store).

Hardware Support for Paging

Every address generated by the CPU is divided into two parts: a **page number (p)** and a **page offset (d)**.

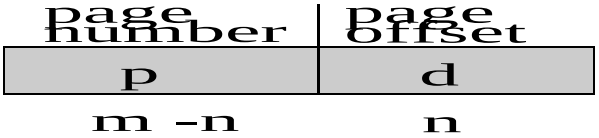
- **Page Table:** The page number is used as an index into a **page table**. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.
- **Frame Table:** Since the operating system is managing physical memory, it must be aware of the allocation details of physical memory—which frames are allocated, which frames are available, how many total frames there are, and so on? This information is generally kept in a data structure called a **frame table**. The frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process or processes.



Defining of Page Size

The page size (like the frame size) is defined by the hardware. The size of a page is a power of 2, varying between 512 bytes and 1 GB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.

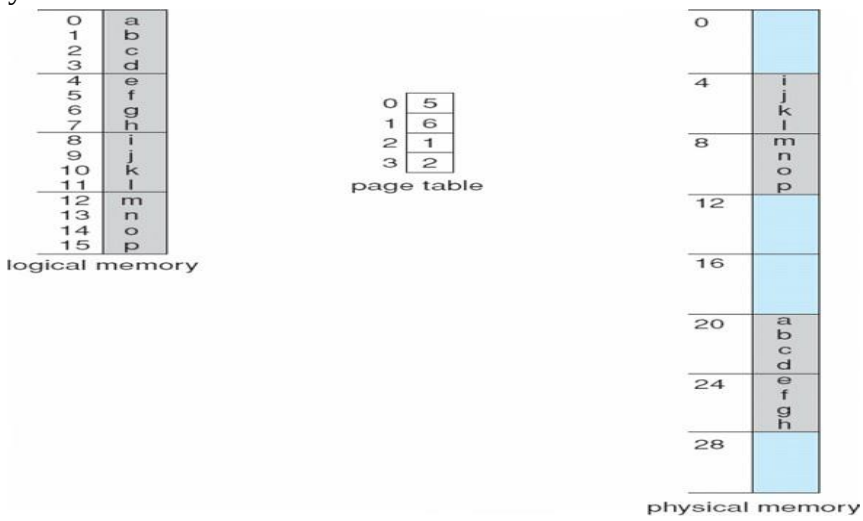
If the size of the logical address space is 2^m, and a page size is 2^n bytes, then the high-order $m- n$ bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:



where p is an index into the page table and d is the displacement within the page.

Example

Here, in the logical address, n= 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages).Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 [= (5 × 4) + 0]. Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 × 4) + 3]. Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 [= (6 × 4) + 0]. Logical address 13 maps to physical address 9.

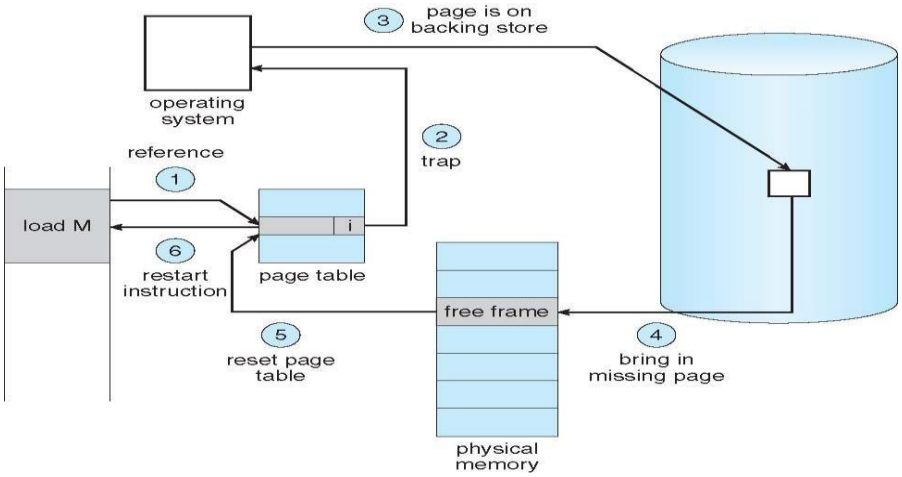


Page Fault

Access to a page marked invalid causes a **page fault**. The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory.

The procedure for handling this page fault is straightforward

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.
3. We find a free frame (by taking one from the free-frame list, for example).
4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.



Page Replacement

In Demand Paging, pages are only brought into memory only when needed. This has two benefits,

1. Saves I/O necessary to load unused pages.
2. Increases the degree of multiprogramming.

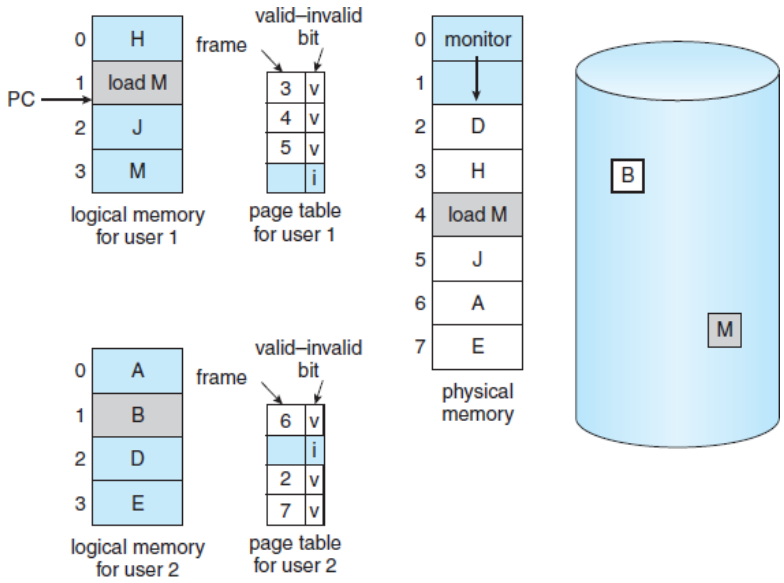
But increasing degree of multi programming may arise new problem called "Over allocating of memory".

Over-Allocating Memory

For example, there are 10 processes and each has 10 pages out of which only 5 may be used. If there are 50 frames then we can allocate only 5 processes if all the 10 pages are loaded. But by using demand paging (we load only used or demanded pages) we can accommodate 10 processes as only 5 pages are in demand. Problem arises when suddenly a process needs all 10 pages but no frames are free.

Over-allocation of memory manifests itself as follows. While a user process is executing, a page fault occurs. The operating system determines where the desired page is residing on the disk but then finds that there are ***no*** free frames on the free-frame list; all memory is in use. The operating system has several options at this point. It could terminate

the user process. This option is not the best choice. The operating system could instead swap out a process, freeing all its frames and reducing the level of multiprogramming. This option is a good one but requires page replacement.

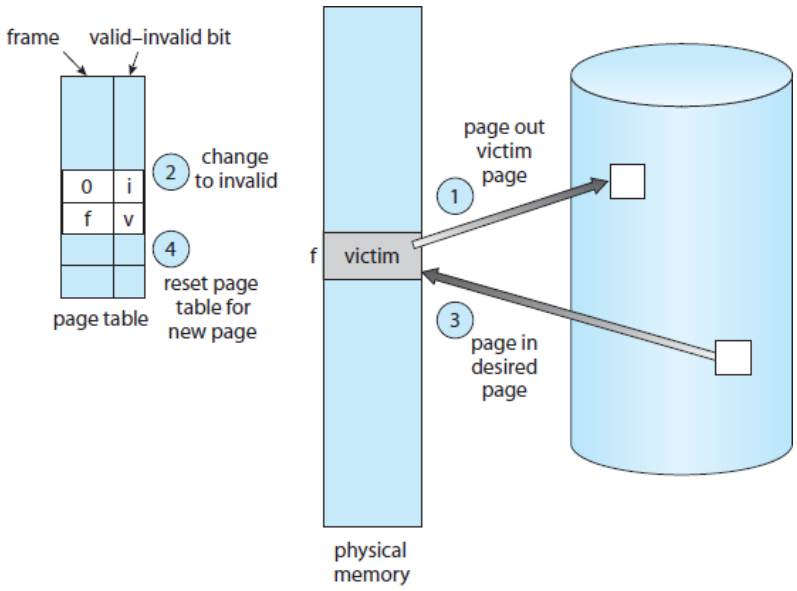


Need for page replacement.

1. Basic Page Replacement

Page replacement takes the following approach,

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



Page replacement.

That is, if we have multiple processes in memory, we must decide how many frames to allocate to each process; and when page replacement is required, we must select the frames that are to be replaced.

Reference String

There are many different page-replacement algorithms. We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a **reference string**.

We can generate reference strings

- Artificially (by using a random-number generator, for example).
- We can trace a given system and record the address of each memory reference. But this produces large amount of data.

To reduce this, we use two facts

a. First, for a given page size (and the page size is generally fixed by the hardware or system), we need to consider only the page number, rather than the entire address.

b. Second, if we have a reference to a page *p*, then any references to page *p* that ***immediately*** follow will never cause a page fault.

Example

If we trace a particular process, we might record the following address sequence:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,

0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

At 100 bytes per page, this sequence is reduced to the following reference string:

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

Page Replacement Algorithms-

Page replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.

Various page replacement algorithms are-

FIFO Page Replacement Algorithm

Optimal Page Replacement Algorithm

LRU Page Replacement Algorithm

MRU Page Replacement Algorithm

FIFO Page Replacement Algorithm-

- As the name suggests, this algorithm works on the principle of "**First in First out**".
- It replaces the oldest page that has been present in the main memory for the longest time.
- It is implemented by keeping track of all the pages in a queue.

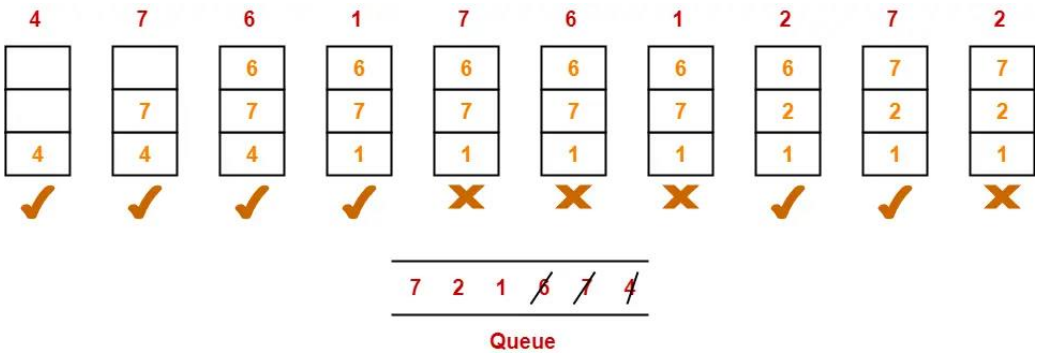
A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2

Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10



Total number of page faults occurred = 6

Calculating Hit ratio-

Total number of page hits = Total number of references – Total number of page misses or page faults

= 10 – 6

= 4

Thus, Hit ratio

= Total number of page hits / Total number of references

= 4 / 10

= 0.4 or 40%

Calculating Miss ratio-

Total number of page misses or page faults = 6

Thus, Miss ratio

= Total number of page misses / Total number of references

= 6 / 10

= 0.6 or 60%

Alternatively,

Miss ratio

= 1 – Hit ratio

= 1 – 0.4

= 0.6 or 60%

Optimal Page Replacement Algorithm-

This algorithm replaces the page that will not be referred by the CPU in future for the longest time.

- It is practically impossible to implement this algorithm.
- This is because the pages that will not be used in future for the longest time can not be predicted.
- However, it is the best known algorithm and gives the least number of page faults.
- Hence, it is used as a performance measure criterion for other algorithms.

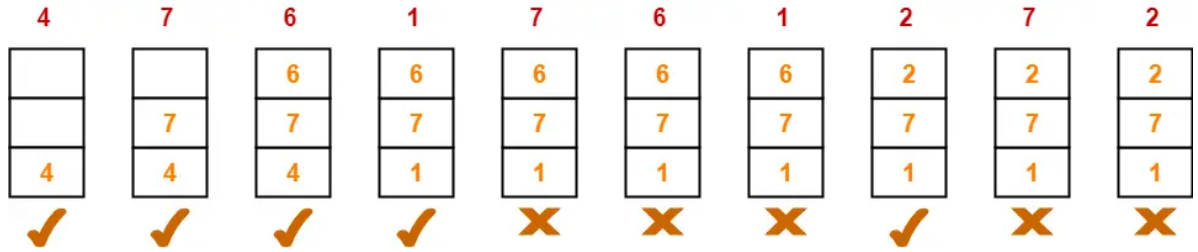
A system uses 3 page frames for storing process pages in main memory. It uses the Optimal page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2

Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10



Total number of page faults occurred = 5

In the similar manner as above-

- Hit ratio = 0.5 or 50%
- Miss ratio = 0.5 or 50%
-

Least Recently Used (LRU) Replacement Algorithm

As the name suggests, this algorithm works on the principle of "**Least Recently Used**".

- It replaces the page that has not been referred by the CPU for the longest time.

This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging.
After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames.
If the page to be searched is found among the frames then, this process is known as Page Hit.
If the page to be searched is not found among the frames then, this process is known as Page Fault.
When Page Fault occurs this problem arises, then the Least Recently Used (LRU) Page Replacement Algorithm comes into picture.
The Least Recently Used (LRU) Page Replacement Algorithms works on a certain principle. The principle is:
Replace the page with the page which is less dimension of time recently used page in the past.
Example:

Suppose the Reference String is:
6, 1, 1, 2, 0, 3, 4, 6, 0
The pages with page numbers 6, 1, 2 are in the frames occupying the frames.
Now, we need to allot a space for the page numbered 0.
Now, we need to travel back into the past to check which page can be replaced.
6 is the oldest page which is available in the Frame.
So, replace 6 with the page numbered 0.
Let us understand this Least Recently Used (LRU) Page Replacement Algorithm working with the help of an example.

Example:
Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with three frames and calculate number of page faults by using Least Recently Used (LRU) Page replacement algorithms.

Points to Remember
Page Not Found - - - > Page Fault
Page Found - - - > Page Hit
Reference String:

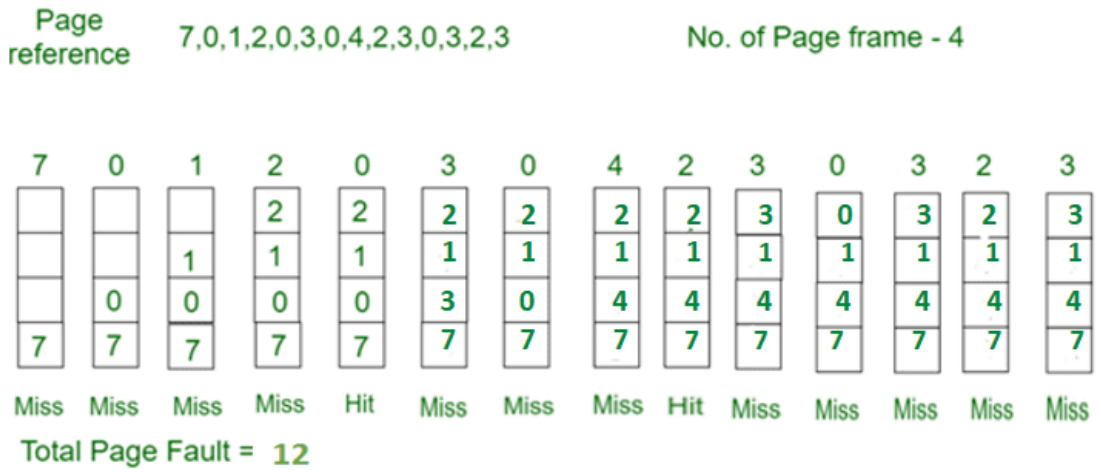
6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

S. no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F3				2	2	2	4	4	4	2	2	2	2	2	2	2	2	2	2	2
F2		1	1	1	1	3	3	3	0	0	0	0	0	0	0	0	0	1	1	1
F1	6	6	6	6	0	0	0	6	6	6	1	1	1	1	1	3	3	3	3	0
Hit (H)/ Fault (F)	F	F	H	F	F	F	F	F	F	F	F	H	H	H	H	F	H	F	H	F

Number of Page Hits = 7

Number of Page Faults = 13
The Ratio of Page Hit to the Page Fault = 7 : 12 - - - > 0.5833 : 1
The Page Hit Percentage = 7 * 100 / 20 = 35%
The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 35 = 65%

4. Most Recently Used (MRU): In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so–> **0 page fault**
when 3 comes it will take place of 0 because it is most recently used —>**1 Page fault**
when 0 comes it will take place of 3 —>**1 Page fault**
when 4 comes it will take place of 0 —>**1 Page fault**

2 is already in memory so —> **0 Page fault**
when 3 comes it will take place of 2 —>**1 Page fault**
when 0 comes it will take place of 3 —>**1 Page fault**
when 3 comes it will take place of 0 —>**1 Page fault**
when 2 comes it will take place of 3 —>**1 Page fault**
when 3 comes it will take place of 2 —>**1 Page fault**

Secondary Storage

1.1 Disk Structure:

A. Magnetic disks provide bulk of secondary storage of modern computers

- Drives rotate at 60 to 200 times per second
- Transfer rate is rate at which data flow between drive and computer
- Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency).
- Head crash results from disk head making contact with the disk surface

B. Disks can be removable

C. Drive attached to computer via I/O bus

1.2 Disk Scheduling:

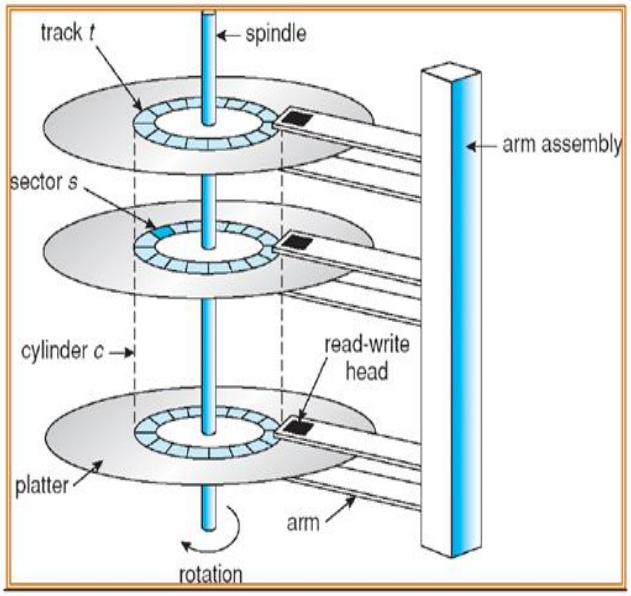
A. The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.

B. Access time has two major components Seek time is the time for the disk are to move the heads to the cylinder containing the desired sector. Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

C. Minimize seek time

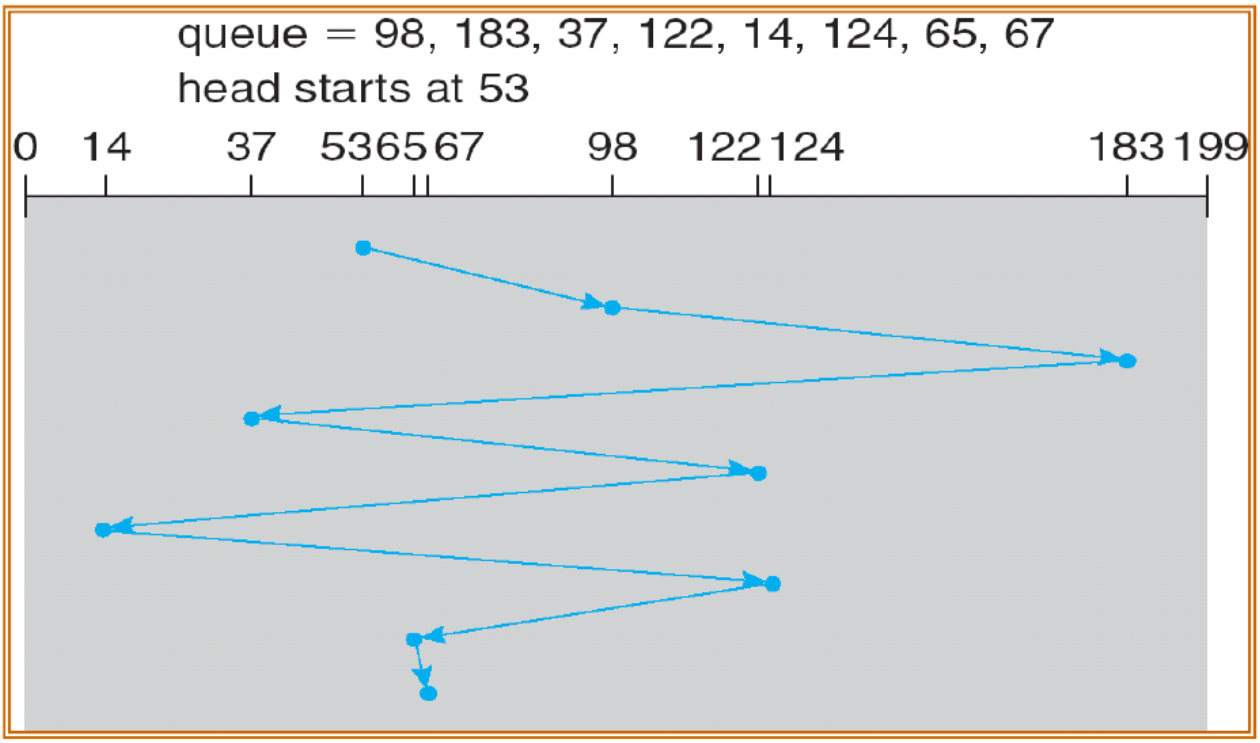
D. Seek time ≈ seek distance

E. Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.



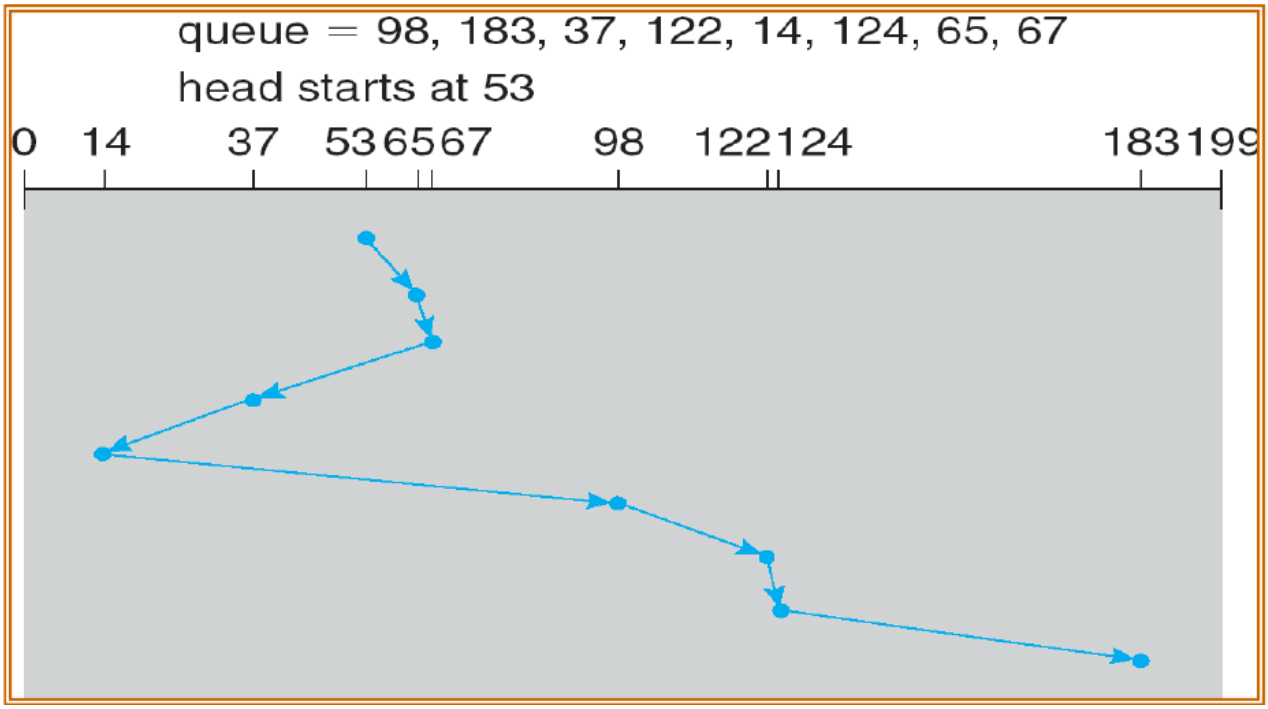
Several algorithms exist to schedule the servicing of disk I/O requests.

1.2.1 FCFS Scheduling (First Come First Served):

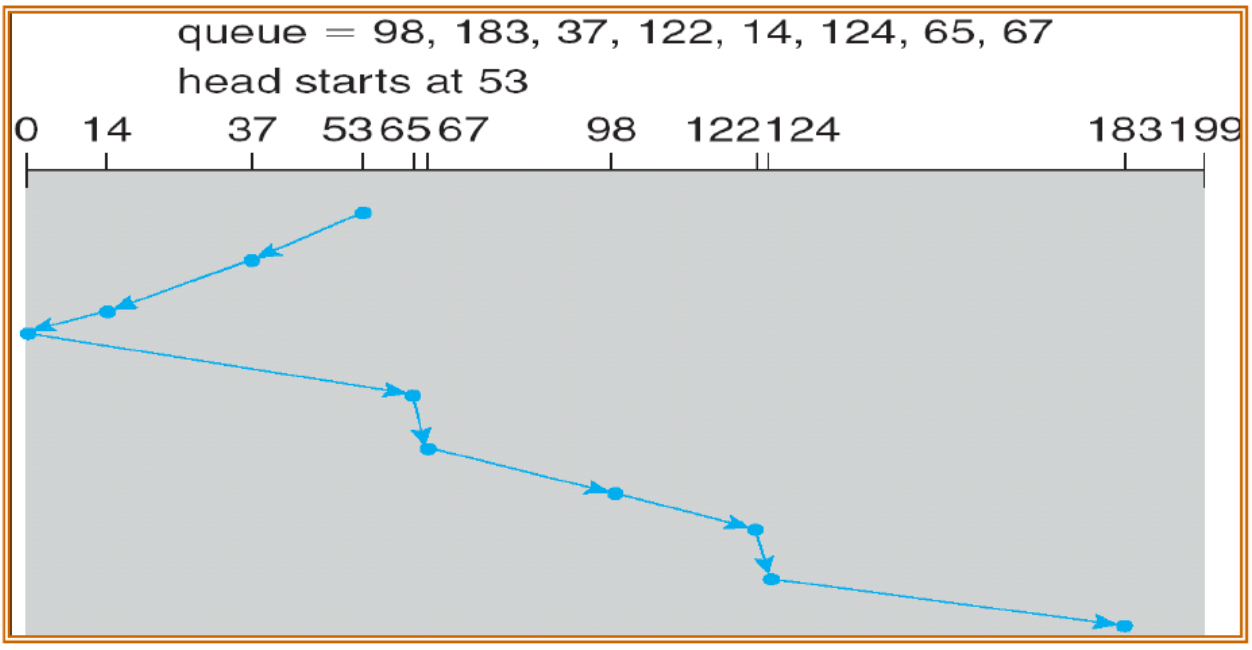


1.2.2 SSTF Scheduling (Shortest-Seek-Time-First)

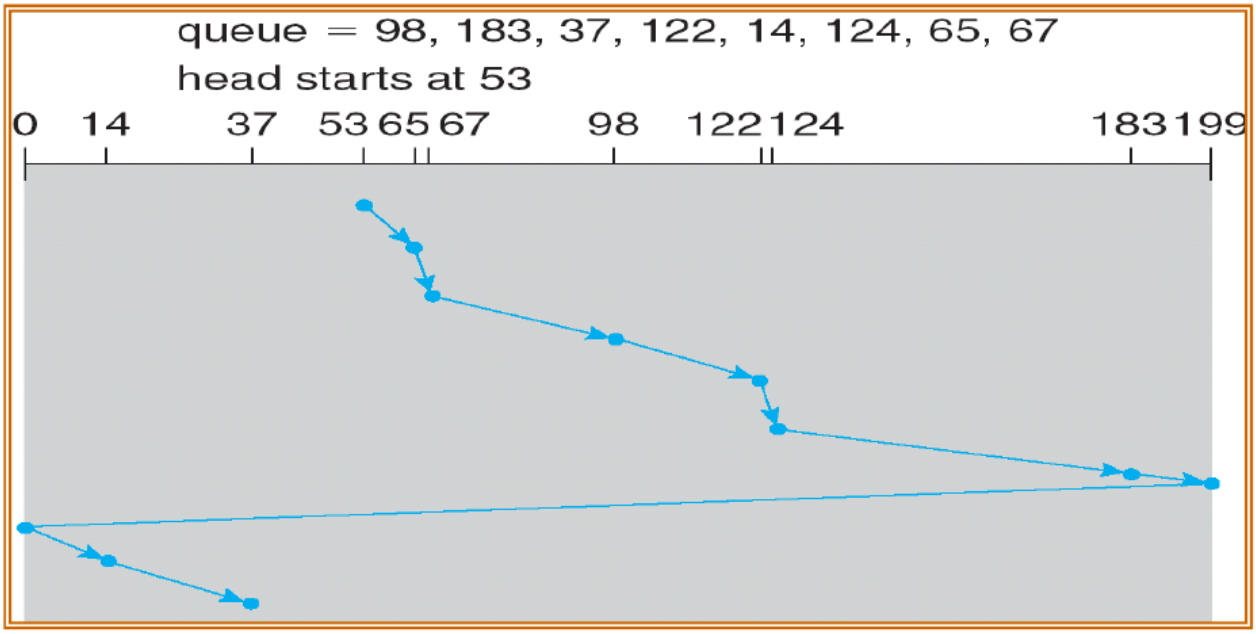
- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.



1.2.3 SCAN Scheduling:
The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. Sometimes called the elevator algorithm.



1.2.4 C-SCAN Scheduling: Provides a more uniform wait time than SCAN.
• The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
• Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

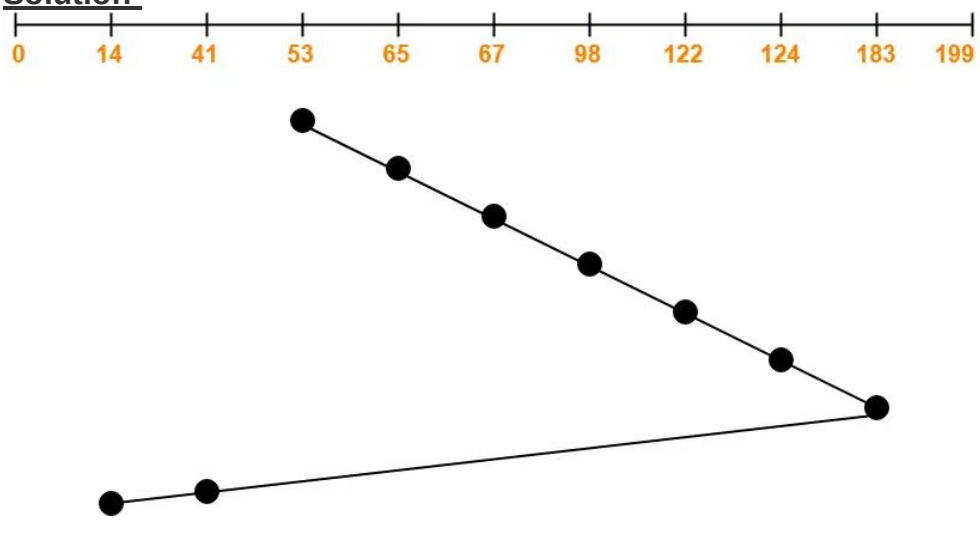


1.2.5 LOOK Disk Scheduling Algorithm-

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

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is

Solution-



Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 41) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 142 + 27

= 299

Alternatively,

Total head movements incurred while servicing these requests

= (183 – 53) + (183 – 14)

= 130 + 169

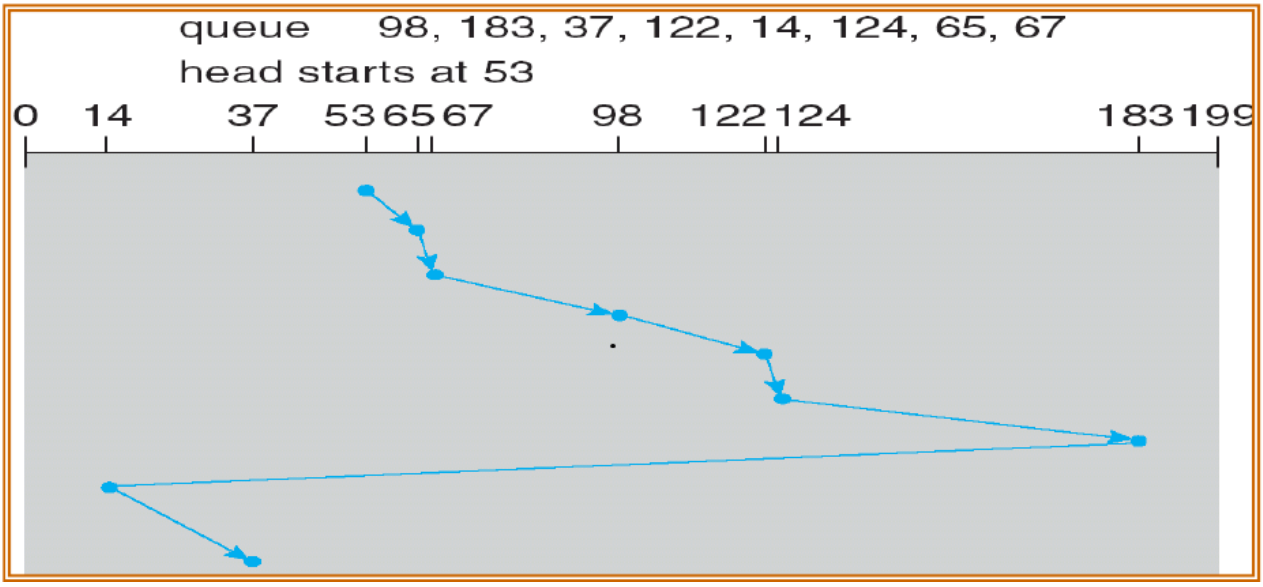
= 299

To gain better understanding about LOOK Disk Scheduling Algorithm,

1.2.6 C LOOK Scheduling:

Version of C-SCAN

• Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk



Criteria for Selecting a Disk-Scheduling Algorithm

• SSTF is common and has a natural appeal

• SCAN and C-SCAN perform better for systems that place a heavy load on the disk.

• Performance depends on the number and types of requests.

• Requests for disk service can be influenced by the file-allocation method.

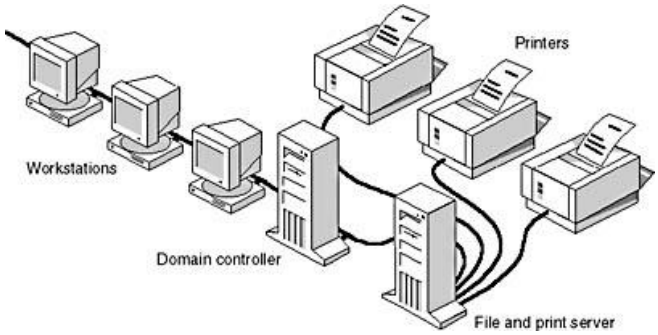
• The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.

• Either SSTF or LOOK is a reasonable choice for the default algorithm.

Network Operating System(NOS) :A combination of software programs that instruct computers and peripherals to accept requests for services across the network and then provide those services.

Without a NOS, a network is nothing more than a number of computer devices connected together. In order to transmit information and communicate across a network, it is necessary to have a NOS.

A network operating system such as the one shown in the Figure:



- Link together all computers and peripherals.
- Coordinates the functions of all computers and peripherals.
- Provides security by controlling access to data and peripherals

A network operating system (NOS) provides services to clients over a network.
Operating systems in general (client OS and NOS) coordinate the interaction between the computer and the programs.
It also controls the allocation and use of hardware resources such as:
Memory.
CPU time.
Disk space.
Peripheral devices

Types of NOS

- There are two major type of network models:
  - Peer-to-peer (P2P)
  - Client/server.
- Both the client/server and peer-to-peer networking models use network operating systems.
- Which NOS is the best is dependent on the end goal of the network**.**

Peer-to-Peer Networking

- In peer-to-peer networking there is a complete sharing of resources, both hardware and software.
- All systems act as both users (client) of resources and providers(server) of resources, but no one system is dedicated to a single function.
- Peer-to-peer networks are generally best suited to small networks and usually are less expensive than client/server networks

In a peer-to-peer (P2P) network operating system users are allowed to share resources and files located on their computers and access shared resources from others.

Examples:
AppleShare used for networking connecting Apple products.
Windows for Workgroups used for networking windows computers.

Advantages of P2P

- Ease of setup
- Less hardware needed
- no server needs to be purchased.

Disadvantages of P2P

- No central location for storage.
- Lack of security that a client/server type offers.

Client/Server Networks

- In client/server networks systems are dedicated to a single function.
- They are either users of network resources (clients) or providers of resources (servers).
- Client/server networks are typically more expensive and robust than peer-to-peer networks and generally support the building of larger networks.
- Network operating systems in client/server architecture enables multiple clients to share resources.
- Client/server network operating systems centralize functions and applications in one or more dedicated servers.
- The server is the center of the system, allowing access to resources and instituting security.
- The network operating system provides a mechanism to integrate all the components on a network to allow multiple users to simultaneously share the same resources regardless of physical location.
- **Examples:**
    - Novell NetWare
    - Windows Server
    - Banyan VINES
    - Linux
- **Advantages**
    - Centralized servers are more stable.
    - Security is provided through the server.
    - New technology and hardware can be easily integrated into the system.
    - Servers are able to be accessed remotely from different locations and types of systems.
- **Disadvantages**
    - Cost of buying and running a server are high.
    - Requires regular maintenance and updates.

Services for Network Operating Systems (NOSs)

- NOSs Services are functions provided by the OS and forms a base used by applications in the network.
- Service is provided by a server and accessed by clients.
- Client: is any process making a request to a server process
- Server: is a process or task that continuously monitors incoming service requests.
- Temporary Client: when a server obtain services from another server.

- For each server, there is a well-defined protocol defining the requests that can be made to that server and the responses that are expected.

Services provided by a network operating system includes:

1. Peripheral Sharing Service
2. File service
3. Directory or Name Service
4. Group Communication Service
5. Time, Memory and Locking Services
6. Mail Services
7. User Services
8. Publishing services

These concepts appear to be related to various services and functions that can be part of a computer or network environment. Let's briefly explain each concept:

1. Peripheral Sharing Service: This service allows multiple computers to share the same peripheral devices like printers or scanners. It ensures that these devices are accessible to all authorized users on the network.
2.File Service: File services provide the ability to create, store, access, and manage files and directories on a network. Examples include Network Attached Storage (NAS) systems and file servers.
3. Directory or Name Service: Directory services help manage and organize information about network resources, such as users, groups, computers, and printers. It typically involves systems like LDAP (Lightweight Directory Access Protocol).
4. Group Communication Service: This service enables communication and collaboration among members of a group or team. It may involve group messaging, conferencing, and file sharing, often seen in collaborative software and messaging platforms.
5. Time, Memory, and Locking Services: These services help manage system resources efficiently. Time services ensure synchronized clocks across a network, memory services optimize memory usage, and locking services prevent conflicts when multiple users or processes access shared resources.
6. Mail Services: Mail services facilitate electronic mail communication. This includes sending, receiving, and storing email messages. Examples include email servers like Microsoft Exchange and email clients like Microsoft Outlook.
7. User Services: User services encompass various functionalities related to user management and authentication. This includes user account creation, access control, and user-specific configurations.
8. Publishing Services: Publishing services involve making information or resources available to a wider audience, often over a network or the internet. It includes web publishing, document sharing, and content management systems.

Operating system issues:
Operating system issues can encompass a wide range of problems that users or administrators may encounter while using or managing a computer system. Here are some common operating system issues:

1.Crashes and Freezes: The operating system may crash or freeze due to software bugs, hardware conflicts, or insufficient resources. This can lead to loss of unsaved work and require a system reboot.

2. Slow Performance: Slow performance can result from resource-intensive applications, malware, or background processes consuming system resources.

3. Driver Problems: Incompatible or outdated drivers can cause hardware components to malfunction or not work as expected.

4. Blue Screen of Death (BSOD): On Windows systems, a BSOD occurs when the system encounters a critical error that it cannot recover from. It usually requires a system restart.

5. Software Compatibility Issues: Some applications may not work properly on certain operating systems or versions, leading to errors or crashes.

6. Virus and Malware Attacks: Malicious software can compromise the security of the operating system and steal sensitive information or cause damage.

7. File System Corruption: File system errors can result from improper shutdowns or hardware issues, leading to data loss or system instability.

8. Authentication and Access Problems: Users might face issues with logging in, incorrect permissions preventing access to files or folders, or account lockouts.

9. Network Connectivity Issues: Difficulty connecting to networks or the internet can arise from misconfigured settings, driver problems, or hardware issues.

10. Update and Patch Problems: Installing updates or patches might lead to compatibility issues, failed installations, or unintended changes.

11. Hardware Compatibility: Certain hardware components may not be recognized or work optimally due to lack of drivers or compatibility issues.

12. Disk Space Shortages: Running out of disk space can hinder system performance and prevent the installation of new software or updates.

13. User Profile Corruption: User profile corruption can result in loss of personal settings, documents, and preferences.

14. Boot Problems: The operating system may fail to boot due to damaged boot files, corrupted system files, or hardware failures.

15. Resource Leaks: Some applications or processes might not release system resources properly, leading to memory leaks and degraded performance.

16. Licensing and Activation Issues: Problems related to software licensing and activation can prevent users from accessing certain features or using the operating system.

17. Compatibility with New Hardware: When upgrading hardware components, the operating system might have trouble recognizing or working with the new hardware.

Dealing with operating system issues often requires troubleshooting, applying updates, running diagnostic tools, checking for malware, and sometimes seeking assistance from technical support or professionals. Regular maintenance, backups, and keeping software up to date can help prevent or mitigate these problems.