OPERATING SYSTEM (BISOS304)

Module-5

DEADLOCK

When processes request a resource and if the resources are not available at that time the process enters into waiting state. Waiting process may not change its state because the resources they are requested are held by other process. This situation is called deadlock.

The situation where the process waiting for the resource i.e., not available is called deadlock.

System Model:-

A system may consist of finite number of resources and is distributed among number of processes. There resources are partitioned into several instances each with identical instances.

A process must request a resource before using it and it must release the resource after using it. It can request any number of resources to carry out a designated task. The amount of resource requested may not exceed the total number of resources available.

A process may utilize the resources in only the following sequences:-

1. Request:- If the request is not granted immediately then the requesting process must wait it can acquire the resources.
2. Use:- The process can operate on the resource.

3. Release:- The process releases the resource after using it.

Deadlock may involve different types of resources.

For eg:- Consider a system with one printer and one tape drive. If a process Pi currently holds a printer and a process Pj holds the tape drive. If process Pi request a tape drive and process Pj request a printer then a deadlock occurs.

Multithread programs are good candidates for deadlock because they compete for shared resources.

Deadlock Characterization:-

Necessary Conditions:-

A deadlock situation can occur if the following 4 conditions occur simultaneously in a system:-

1. Mutual Exclusion:- Only one process must hold the resource at a time. If any other process requests for the resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait:- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by the other process.

3. No Preemption:- Resources can't be preempted i.e., only the process holding the resources must release it after the process has completed its task.

4. Circular Wait:- A set {P0,P1.....Pn} of waiting process must exist such that P0 is waiting for a resource i.e., held by P1, P1 is waiting for a resource i.e., held by P2. Pn-1 is waiting for resource held by process Pn and Pn is waiting for the resource i.e., held by P1. All the four conditions must hold for a deadlock to occur.

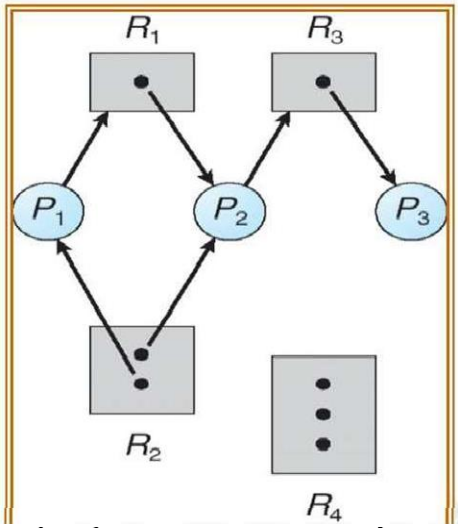
Resource Allocation Graph:-

Deadlocks are described by using a directed graph called system resource allocation graph. The graph consists of set of vertices (v) and set of edges (e).

The set of vertices (v) can be described into two different types of nodes P={P1,P2.....Pn} i.e., set consisting of all active processes and R={R1,R2.....Rn}i.e., set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj denoted by Pi → Ri indicates that Pi requested an instance of resource Rj and is waiting. This edge is called Request edge.

A directed edge Ri →Pj signifies that resource Rj is held by process Pi. This is called Assignment edge.



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

If each resource type has exactly one instance than a cycle implies that a deadlock has occurred. If each resource has several instances then a cycle do not necessarily implies that a deadlock has occurred.

Methods for Handling Deadlocks:-

There are three ways to deal with deadlock problem

- We can use a protocol to prevent deadlocks ensuring that the system will never enter into the deadlock state.
- We allow a system to enter into deadlock state, detect it and recover from it.
- We ignore the problem and pretend that the deadlock never occur in the system. This is used by most OS including UNIX.

To ensure that the deadlock never occur the system can use either deadlock avoidance or a deadlock prevention.

Deadlock prevention is a set of method for ensuring that at least one of the necessary conditions does not occur.

Deadlock avoidance requires the OS is given advance information about which resource a process will request and use during its lifetime.

If a system does not use either deadlock avoidance or deadlock prevention then a deadlock situation may occur. During this it can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and algorithm to recover from deadlock. Undetected deadlock will result in deterioration of the system performance.

Deadlock Prevention:-

For a deadlock to occur each of the four necessary conditions must hold. If at least one of the there condition does not hold then we can prevent occurrence of deadlock.

1. Mutual Exclusion:-
   This holds for non-sharable resources.

Eg:- A printer can be used by only one process at a time.
Mutual exclusion is not possible in sharable resources and thus they cannot be involved in deadlock. Read-only files are good examples for sharable resources. A process never waits for accessing a sharable resource. So we cannot prevent deadlock by denying the mutual exclusion condition in non-sharable resources.

2. Hold and Wait:-
   This condition can be eliminated by forcing a process to release all its resources held by it when it request a resource i.e., not available.

One protocol can be used is that each process is allocated with all of its resources before its start execution.

Eg:- consider a process that copies the data from a tape drive to the disk, sorts the file and then prints the results to a printer. If all the resources are allocated at the beginning then the tape drive, disk files and printer are assigned to the process. The main problem with this is it leads to low resource utilization because it requires printer at the last and is allocated with it from the beginning so that no other process can use it.
Another protocol that can be used is to allow a process to request a resource when the process has none. i.e., the process is allocated with tape drive and disk file. It performs the required operation and releases both. Then the process once again request for disk file and the printer. Problem with this is starvation is possible.

3. No Preemption:-

To ensure that this condition never occurs the resources must be preempted. The following protocol can be used.
If a process is holding some resource and request another resource that cannot be immediately allocated to it, then all the resources currently held by the requesting process are preempted and added to the list of resources for which other processes may be waiting. The process will be restarted only when it regains the old resources and the new resources that it is requesting.
When a process request resources, we check whether they are available or not. If they are available we allocate them else we check that whether they are allocated to some other waiting process. If so we preempt the resources from the waiting process and allocate them to the requesting process. The requesting process must wait.

4. Circular Wait:-

The fourth and the final condition for deadlock is the circular wait condition. One way to ensure that this condition never, is to impose ordering on all resource types and each process requests resource in an increasing order.

Let R={R1,R2,.....Rn} be the set of resource types. We assign each resource type with a unique integer value. This will allows us to compare two resources and determine whether one precedes the other in ordering.

Eg:-we can define a one to one function as follows :-

F(disk drive)=5 F(printer)=12 F(tape drive)=1

Deadlock can be prevented by using the following protocol:-

Each process can request the resource in increasing order. A process can request any number of instances of resource type say Ri and it can request instances of resource type Rj only F(Rj) > F(Ri).

Alternatively when a process requests an instance of resource type Rj, it has released any resource Ri such that F(Ri) >= F(Rj). If these two protocol are used then the circular wait can't hold.

Deadlock Avoidance:-

Deadlock prevention algorithm may lead to low device utilization and reduces system throughput.

Avoiding deadlocks requires additional information about how resources are to be requested. With the knowledge of the complete sequences of requests and releases we can decide for each requests whether or not the process should wait. For each requests it requires to check the resources currently available, resources that are currently allocated to each processes future requests and release of each process to decide whether the current requests can be satisfied or must wait to avoid future possible deadlock.

A deadlock avoidance algorithm dynamically examines the resources allocation state to ensure that a circular wait condition never exists. The resource allocation state is defined by the number of available and allocated resources and the maximum demand of each process.

Safe State:-

A state is a safe state in which there exists at least one order in which all the process will run completely without resulting in a deadlock.

A system is in safe state if there exists a safe sequence.

A sequence of processes <P1,P2,.....Pn> is a safe sequence for the current allocation state if for each Pi the resources that Pi can request can be satisfied by the currently available resources.
If the resources that Pi requests are not currently available then Pi can obtain all of its needed resource to complete its designated task.

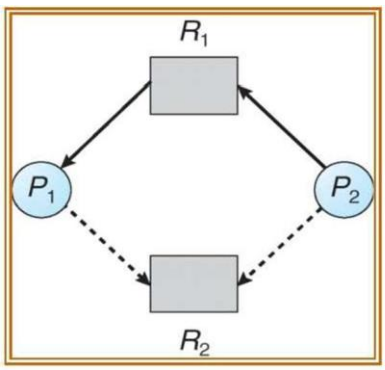
A safe state is not a deadlock state.

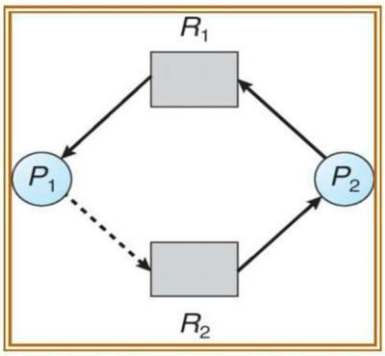
Whenever a process request a resource i.e., currently available, the system must decide whether resources can be allocated immediately or whether the process must wait. The request is granted only if the allocation leaves the system in safe state.
In this, if a process requests a resource i.e., currently available it must still have to wait. Thus resource utilization may be lower than it would be without a deadlock avoidance algorithm.

Resource Allocation Graph Algorithm:-

This algorithm is used only if we have one instance of a resource type. In addition to the request edge and the assignment edge a new edge called claim edge is used.
For eg:- A claim edge Pi Rj indicates that process Pi may request Rj in future. The claim edge is represented by a dotted line.

- When a process Pi requests the resource Rj, the claim edge is converted to a request edge.
- When resource Rj is released by process Pi, the assignment edge Rj Pi is replaced by the claim edge Pi Rj.
- When a process Pi requests resource Rj the request is granted only if converting the request edge Pi Rj to as assignment edge Rj Pi do not result in a cycle. Cycle detection algorithm is used to detect the cycle. If there are no cycles then the allocation of the resource to process leave the system in safe state





.Banker's Algorithm:-

This algorithm is applicable to the system with multiple instances of each resource types, but this is less efficient than the resource allocation graph algorithm.

When a new process enters the system it must declare the maximum number of resources that it may need. This number may not exceed the total number of resources in the system. The system must determine that whether the allocation of the resources will leave the system in a safe state or not. If it is so resources are allocated else it should wait until the process release enough resources.

Several data structures are used to implement the banker's algorithm. Let 'n' be the number of processes in the system and 'm' be the number of resources types. We need the following data structures:-

Available:- A vector of length m indicates the number of available resources. If Available[j]=k, then k instances of resource type Rj is available.

Max:- An n*m matrix defines the maximum demand of each process if Max[i,j]=k, then Pi may request at most k instances of resource type Rj.

Allocation:- An n*m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j]=k, then Pi is currently k instances of resource type Rj.

Need:- An n*m matrix indicates the remaining resources need of each process. If Need[i,j]=k, then Pi may need k more instances of resource type Rj to compute its task. So Need[i,j]=Max[i,j]-Allocation[i]

Safety Algorithm:-

This algorithm is used to find out whether or not a system is in safe state or not.

Step 1. Let work and finish be two vectors of length M and N respectively. Initialize work = available and

Finish[i]=false for i=1,2,3,.....n

Step 2. Find i such that both

Finish[i]=false
Need i <= work

If no such i exist then go to step 4

Step 3. Work = work + Allocation

Finish[i]=true
Go to step 2

Step 4. If finish[i]=true for all i, then the system is in safe state.

This algorithm may require an order of m*n*n operation to decide whether a state is safe.

Resource Request Algorithm:-

Let Request(i) be the request vector of process Pi. If Request[(i)[j]=k, then process Pi wants K instances of the resource type Rj. When a request for resources is made by process Pi the following actions are taken.

If Request(i) <= Need(i) go to step 2 otherwise raise an error condition since the process has exceeded its maximum claim.

If Request(i) <= Available go to step 3 otherwise Pi must wait. Since the resources are not available.

If the system want to allocate the requested resources to process Pi then modify the state as follows.

Available = Available – Request(i)
Allocation(i) = Allocation(i) + Request(i)
Need(i) = Need(i) - Request(i)

If the resulting resource allocation state is safe, the transaction is complete and Pi is allocated its resources. If the new state is unsafe then Pi must wait for Request(i) and old resource allocation state is restored.

Deadlock Detection:-

If a system does not employ either deadlock prevention or a deadlock avoidance algorithm then a deadlock situation may occur. In this environment the system may provide

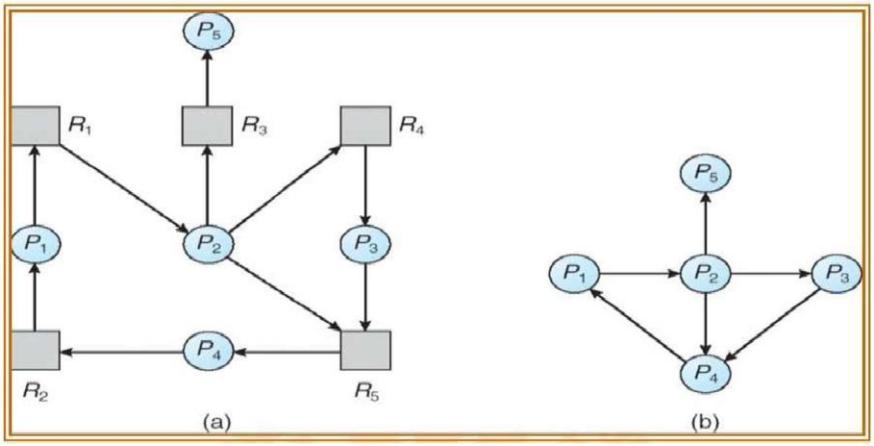
- An algorithm that examines the state of the system to determine whether a deadlock has occurred.
- An algorithm to recover from the deadlock.

Single Instances of each Resource Type:-

If all the resources have only a single instance then we can define deadlock detection algorithm that uses a variant of resource allocation graph called a wait for graph. This graph is obtained by removing the nodes of type resources and removing appropriate edges.

An edge from Pi to Pj in wait for graph implies that Pi is waiting for Pj to release a resource that Pi needs.

An edge from Pi to Pj exists in wait for graph if and only if the corresponding resource allocation graph contains the edges Pi →Rq and Rq→ Pj. Deadlock exists within the system if and only if there is a cycle. To detect deadlock the system needs an algorithm that searches for cycle in a graph.



Several Instances of a Resource Types:-

The wait for graph is applicable to only a single instance of a resource type. The following algorithm applies if there are several instances of a resource type. The following data structures are used:-

Available:- Is a vector of length m indicating the number of available resources of each type.
Allocation:- Is an m*n matrix which defines the number of resources of each type currently allocated to each process.
Request:- Is an m*n matrix indicating the current request of each process.
If request[i,j]=k then Pi is requesting k more instances of resources type Rj.

Step 1. let work and finish be vectors of length m and n respectively.
Initialize Work = available/expression.
For i=0,1,2.....n if allocation(i)!=0 then Finish[i]=0 else Finish[i]=true

Step 2. Find an index(i) such that both
Finish[i] = false
Request(i)<=work
If no such I exist go to step 4.

Step 3. Work = work + Allocation(i)
Finish[i] = true
Go to step 2.

Step 4. If Finish[i] = false for some i where m>=i>=1.
When a system is in a deadlock state. This algorithm needs an order of m*n square operations to detect whether the system is in deadlock state or not.

Recovery From Deadlock

Abort the processes

Abort all deadlocked processes.

Abort one process at a time until the deadlock cycle is eliminated. In which order should we choose to abort?

- Priority of the process.
- How long process has computed, and how much longer to completion.
- Resources the process has used.
- Resources process needs to complete.
- How many processes will need to be terminated.
- Is process interactive or batch?.

Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Approaches To Breaking a Deadlock

Process Termination

To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

Abort all the Deadlocked Processes: Aborting all the processes will certainly break the deadlock but at a great expense. The deadlocked processes may have been computed for a long time, and the result of those partial computations must be discarded and there is a probability of recalculating them later.

Abort one process at a time until the deadlock is eliminated: Abort one deadlocked process at a time, until the deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because, after aborting each process, we have to run a deadlock detection algorithm to check whether any processes are still deadlocked.

Advantages of Process Termination

It is a simple method for breaking a deadlock.

It ensures that the deadlock will be resolved quickly, as all processes involved in the deadlock are terminated simultaneously.

It frees up resources that were being used by the deadlocked processes, making those resources available for other processes.

Disadvantages of Process Termination

It can result in the loss of data and other resources that were being used by the terminated processes.

It may cause further problems in the system if the terminated processes were critical to the system's operation.

It may result in a waste of resources, as the terminated processes may have already completed a significant amount of work before being terminated.

For Process

Destroy a process: Although killing a process can solve our problem, choosing which process to kill is more important. The operating system typically terminates a process after it has completed the least amount of work.

End all processes: Although not suggestible, this strategy can be used if the issue worsens significantly. Because each process will have to start from scratch after being killed, the system will become inefficient.

Resource Preemption

To eliminate deadlocks using resource preemption, we preempt some resources from processes and give those resources to other processes. This method will raise three issues –

Selecting a victim: We must determine which resources and which processes are to be preempted and also in order to minimize the cost.

Rollback: We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means aborting the process and restarting it.

Starvation: In a system, it may happen that the same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called Starvation and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

Advantages of Resource Preemption

It can help in breaking a deadlock without terminating any processes, thus preserving data and resources.

It is more efficient than process termination as it targets only the resources that are causing the deadlock.

It can potentially avoid the need to restart the system.

Disadvantages of Resource Preemption

It may lead to increased overhead due to the need to determine which resources and processes should be preempted.

It may cause further problems if the preempted resources were critical to the system's operation.

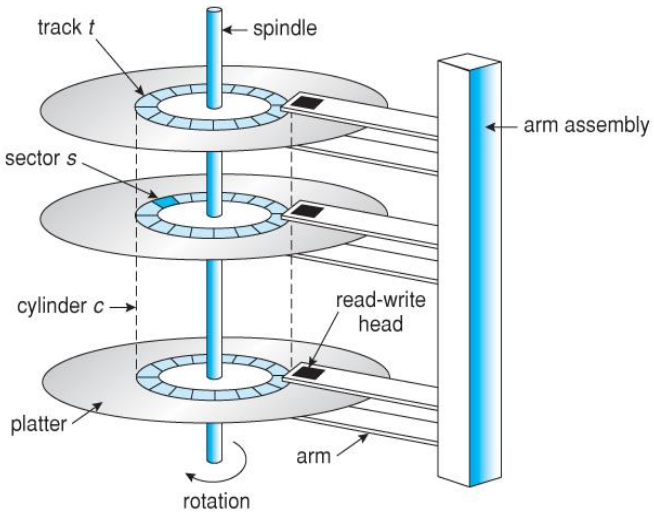
It may cause delays in the completion of processes if resources are frequently preempted.

Module 5

Secondary – Storage Structure

Magnetic disks provide a bulk of secondary storage. Disks come in various sizes and speed. Here the information is stored magnetically. Each disk platter has a flat circular shape like CD. The two surfaces of a platter are covered with a magnetic material. The surface of a platter is logically divided into circular **tracks,** which are subdivided into **sectors.** Sector is the basic unit of storage. The set of tracks that are at one arm position makes up a **cylinder.**

The number of cylinders in the disk dirve equals the number of tracks in each platter. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of disk drives is measured in gigabytes.



The head moves from the inner track of the disk to the outer track. When the disk drive is operating the disks is rotating at a constant speed.

To read or write the head must be positioned at the desired track and at the beginning of the desired sector on that track.

- Seek Time:-Seek time is the time required to move the disk arm to the required track.
- Rotational Latency(Rotational Delay):- Rotational latency is the time taken for the disk torotate so that the required sector comes under the r/w head.
- Positioning time or random access time is the summation of seek time and rotational delay.
- Disk Bandwidth:-Disk bandwidth is the total number of bytes transferred divided by total time between the first request for service and the completion of last transfer.
- Transfer rate is the rate at which data flow between the drive and the computer.

As the disk head flies on an extremely thin cushion of air, the head will make contact with the disk surface. Although the disk platters are coated with a thin protective layer, sometimes the head will damage the magnetic surface. This accident is called a **head crash.**

Magnetic Tapes

Magnetic tape is a secondary-storage medium. It is a permanent memory and can hold large quantities of data. The time taken to access data (access time) is large compared with that of magnetic disk, because here data is accessed sequentially. When the nth data has to be read, the tape starts moving from first and reaches the nth position and then data is read from nth position. It is not possible to directly move to the nth position. So tapes are used mainly for backup, for storage of infrequently used information.

DISK STRUCTURE

Each disk platter is divided into number of tracks and each track is divided into number of sectors. Sectors is the basic unit for read or write operation in the disk.

Modern disk drives are addressed as a large one-dimensional array. The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

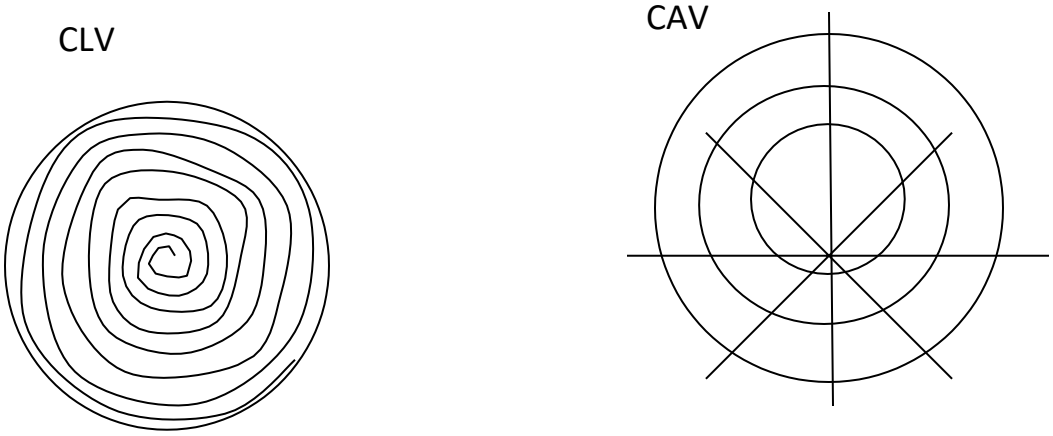
The disk structure (architecture) can be of two types –

i) Constant Linear Velocity (CLV)

ii) **Constant Angular Velocity** (CAV)

i) CLV - The density of bits per track is uniform. The farther a track is from the center of the disk, the greater its length, so the more sectors it can hold. As we move from outer zones to inner zones, the number of sectors per track decreases. This architecture is used in CD-ROM and DVD-ROM.

ii) CAV – There is same number of sectors in each track. The sectors are densely packed in the inner tracks. The density of bits decreases from inner tracks to outer tracks to keep the data rate constant.



DISK ATTACHMENT

Computers can access data in two ways.

i) via I/O ports (or host-attached storage)
ii) via a remote host in a distributed file system(or network-attached storage)

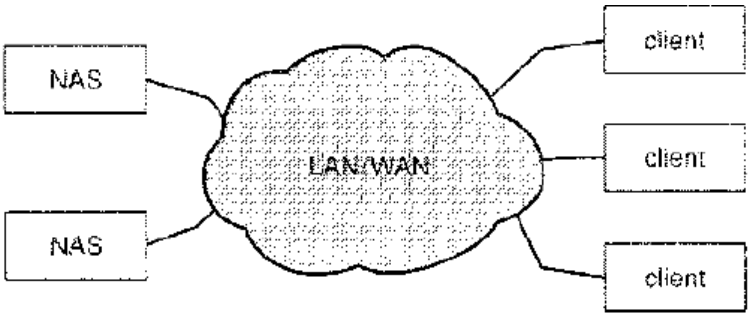
i) Host-Attached Storage

Host-attached storage is storage accessed through local I/O ports. Example : the typical desktop PC uses an I/O bus architecture called IDE or ATA. This architecture supports amaximum of two drives per I/O bus. The other cabling systems are – **SATA**(Serially Attached Technology Attachment), **SCSI**(Small Computer System Interface) and **fiber channel** (FC).

SCSI is a bus architecture. Its physical medium is usually a ribbon cable. FC is a high-speed serial architecture that can operate over optical fiber or over a four-conductor copper cable. An improved version of this architecture is the basis of **storage-area networks** (SANs).

ii) Network-Attached Storage

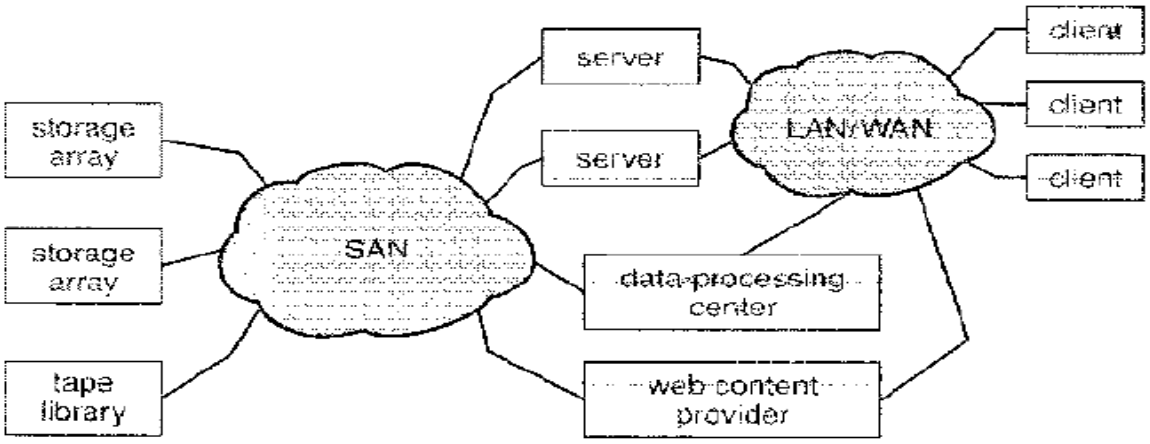
A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a network as shown in the figure. Clients access network-attached storage via a remote-procedure-call interface. The remote procedure calls (RPCs) are carried via TCP or UDP over an IP network—-usually the same local-area network (LAN) carries all data traffic to the clients.



Network-attached storage provides a convenient way for all the computers on a LAN to share a pool of storage files.

iii) Storage Area Network(SAN)

A storage-area network (SAN) is a private network connecting servers and storage units. The power of a SAN lies in its flexibility. Multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. A SAN switch allows or prohibits access between the hosts and the storage. Fiber Chanel is the most common SAN interconnect.

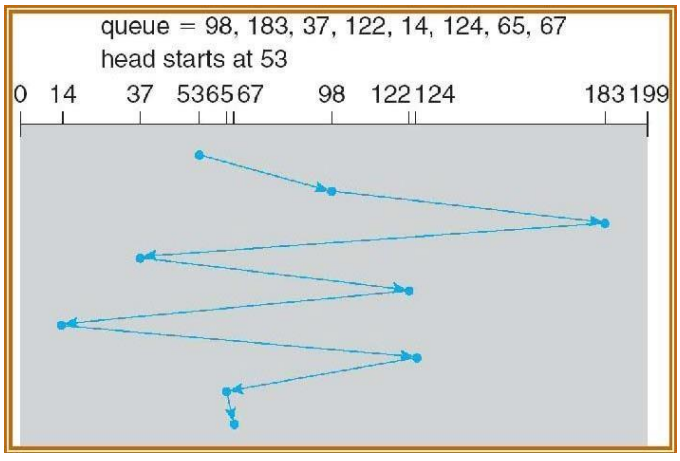


DISK SCHEDULING

Different types of disk scheduling algorithms are as follows:

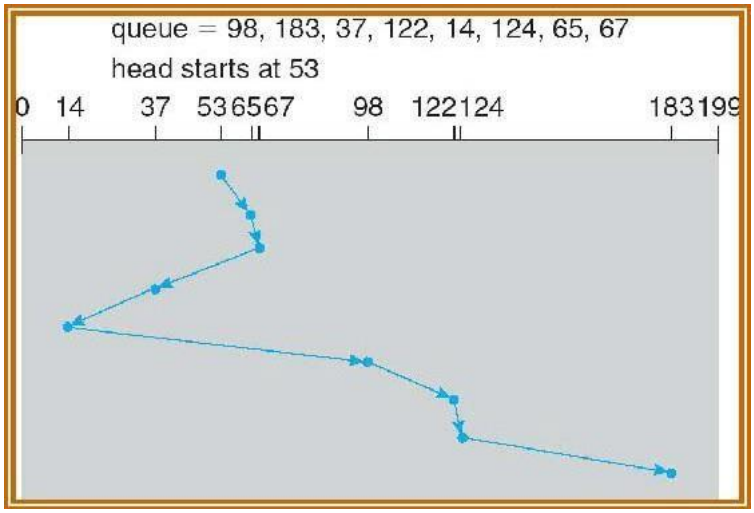
- FCFS (First Come First Serve)
- SSTF(Shortest Seek Time First)
- SCAN (Elecvator)
- C-SCAN
- LOOK
- C-LOOK

i) **FCFS scheduling algorithm**: This is the simplest form of disk scheduling algorithm. This services the request in the order they are received. This algorithm is fair but do not provide fastest service. It takes no special care to minimize the overall seek time. *Eg:-* consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37,122, 14, 124, 65, 67



If the disk head is initially at 53, it will first move from 53 to 98 then to 183 and then to 37, 122, 14, 124, 65, 67 for a total head movement of 640 cylinders. The wildswing from 122 to 14 and then back to 124 illustrates the problem with this schedule.

ii) **SSTF** (Shortest Seek Time First) **algorithm**: This selects the request with minimum seek time from the current head position. SSTF chooses the pending request closest to the current head position. *Eg*:- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67

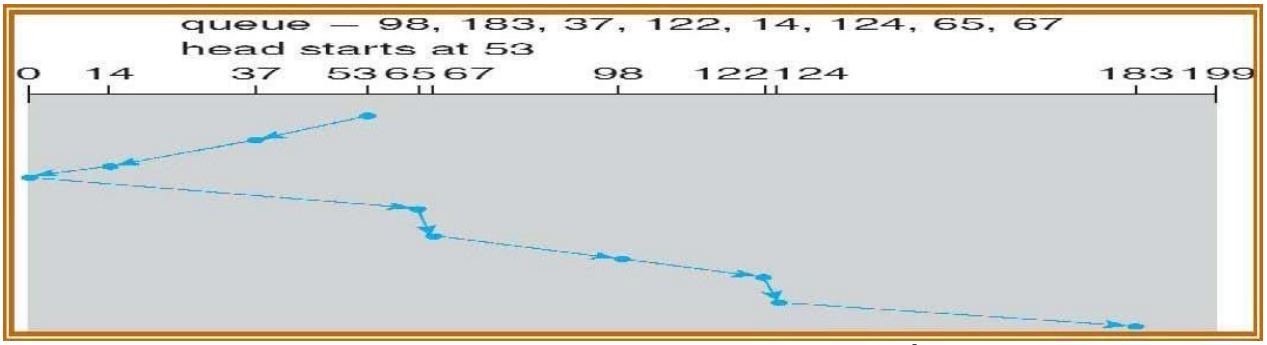


If the disk head is initially at 53, the closest is at cylinder 65, then 67, then 37 is closer then 98 to 67. So it services 37, continuing we service 14, 98, 122, 124 and finally 183. The total head movement is only 236 cylinders. SSTF is a substantial improvement over FCFS, it is not optimal.

iii) **SCAN algorithm**: In this the disk arm starts moving towards one end, servicing the request as it reaches each cylinder until it gets to the other end of the disk. At the other end, the direction of the head movement is reversed and servicing continues. The initial direction is chosen depending upon the direction of the head.

Eg:-:-consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67

If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and183. At cylinder 199 the arm will reverse and will move towards the other end of the disk servicing 37 and then 14. The SCAN is also called as **elevator** algorithm.



If the disk head is initially at 53 and if the head is moving towards **0th track**, it services 37 and then 14. At cylinder 0 the arm will reverse and will move towards the other end of the disk servicing 65, 67, 98, 122, 124 and 183.

iv) **C-SCAN** (Circular scan) **algorithm**:

C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from end of the disk to the other servicing the request along theway. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any request on the return.

Eg:- consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67

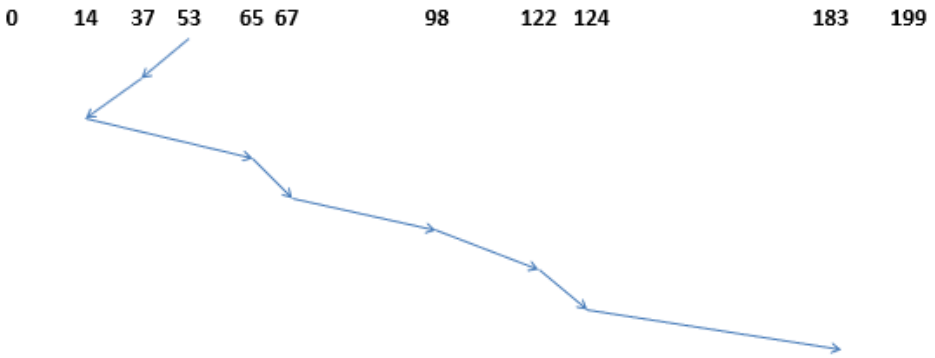
If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and 183. At cylinder 199 the arm will reverse and will move immediately towards the other end of the disk, then changes the direction of head and serves 14 and then 37.

Note: If the disk head is initially at 53 and if the head is moving towards track 0, it services 37 and 14 first. At cylinder 0 the arm will reverse and will move immediately towards the other end of the disk servicing 65, 67, 98, 122, 124 and183.

v) Look Scheduling algorithm:

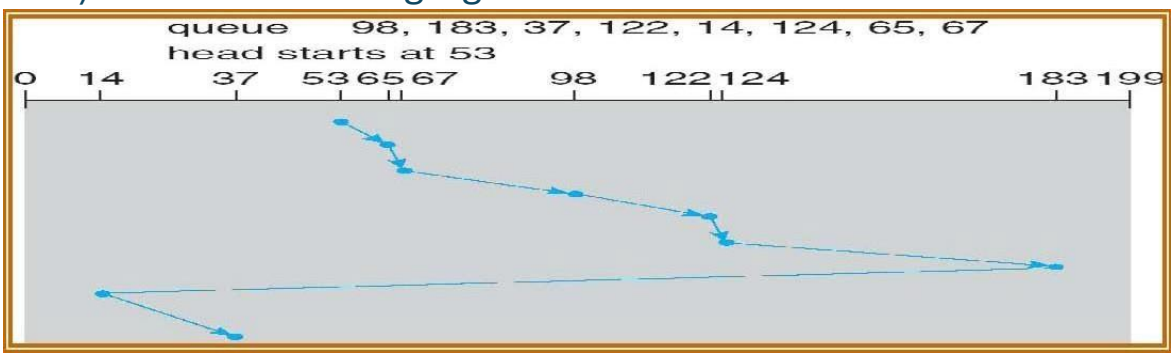
Look and C-Look scheduling are different version of SCAN and C-SCAN respectively. Here the arm goes only as far as the final request in each direction. Then it reverses, without going all the way to the end of the disk. The Look and C-Look scheduling look for a request before continuing to move in a given direction.

A drive has 200 cylinders numbered 0 to 199. The drive is currently serving a request 53 .The queue of pending requests in order is: 98, 183, 37, 122, 14, 124, 65, 67 starting from current head position, what is total distance travelled for LOOK Algorithm, if the head is moving towards 0?



If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and183. At the final request 183, the arm will reverse and will move towards the first request 14 and then serves 37.

vi) C-Look Scheduling algorithm:



If the disk head is initially at 53 and if the head is moving towards the outer track, it services 65, 67, 98, 122, 124 and 183. At the last request, the arm will reverse and will move immediately towards the first request 14 and then serves 37.

Selection of a Disk-Scheduling Algorithm

SSTF is commonly used and it increases performance over FCFS.SCAN and C-SCAN algorithm is better for a heavy load on disk. SCAN and C-SCAN have less starvation problem.
Disk scheduling algorithm should be written as a separate module of the operating system.SSTF or Look is a reasonable choice for a default algorithm.

SSTF is commonly used algorithms has it has a less seek time when compared with other algorithms. SCAN and C-SCAN perform better for systems with a heavy load on the disk, (ie. more read and write operations from disk).

Selection of disk scheduling algorithm is influenced by the file allocation method, if contiguous file allocation is choosen, then FCFS is best suitable, because the files are stored in contiguous blicks and there will be limited head movements required. A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.

The location of directories and index blocks is also important. Since every file must be opened tobe used, and opening a file requires searching the directory structure, the directories will be accessed frequently. Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder. The disk head has to move the entire width of the disk. If the directory entry were on the middle cylinder, the head would have to move, at most, one-half the width. Caching the directories and index blocks in main memory can also help to reduce the disk-arm movement,particularly for read requests.

Because of these complexities, the disk-scheduling algorithm is very important and is written asa separate module of the operating system.