UNIT-V

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.
Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks.

DEADLOCKS

System model:

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, I/O devices are examples of resource types. If a system has 2 CPUs, then the resource type CPU has 2 instances.
A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its task. The number of resources as it requires to carry out its task. The number of resources requested may not exceed the total number of resources available in the system. A process cannot request 3 printers if the system has only two.
A process may utilize a resource in the following sequence:
(I) REQUEST: The process requests the resource. If the request cannot be granted immediately (if the resource is being used by another process), then therequesting process must wait until it can acquire theresource.
(II) USE: The process can operate on the resource .if the resource is a printer, the process can print on theprinter.
(III) RELEASE: The process release theresource.
For each use of a kernel managed by a process the operating system checks that the process has requested and has been allocated the resource. A system table records whether each resource is free (or) allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.
To illustrate a deadlocked state, consider a system with 3 CDRW drives. Each of 3 processes holds one of these CDRW drives. If each process now requests another drive, the 3 processes will be in a deadlocked state. Each is waiting for the event "CDRW is released" which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.
Deadlocks may also involve different resource types. Consider a system with one printer and one DVD drive. The process P_i is holding the DVD and process P_j is holding the printer. If P_i requests the printer and P_j requests the DVD drive, a deadlock occurs.

DEADLOCK CHARACTERIZATION:

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

NECESSARY CONDITIONS:

A deadlock situation can arise if the following 4 conditions hold simultaneously in a system:

1. MUTUAL EXCLUSION: Only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until theresource has beenreleased.
2. HOLD AND WAIT: A process must be holding at least one resource and waitingto acquire additional resources that are currently being held by otherprocesses.
3. NO PREEMPTION: Resources cannot be preempted. A resource can be released only voluntarily by the process holding it, after that process has completed itstask.
4. CIRCULAR WAIT: A set {P_0,P_1,…..P_n} of waiting processes must exist such that P_0 is waiting for resource held by P_1, P_1 is waiting for a resource held by P_2,……,P_{n-1} is waiting for a resource held by P_n and P_n is waiting for a resource held byP_0.

RESOURCE ALLOCATION GRAPH

Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. the set of vertices V is partitioned into 2 different types of nodes:

P = {P_1, P_2….P_n}, the set consisting of all the active processes in the system. R= {R_1, R_2….R_m}, the set consisting of all resource types in the system.

A directed edge from process P_i to resource type R_j is denoted by P_i ->R_j. It signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource.

A directed edge from resource type R_j to process P_i is denoted by R_j ->P_i, it signifies that an instance of resource type R_j has been allocated to process P_i.

A directed edge P_i ->R_j is called a requested edge. A directed edge R_j->$P_i$is called an assignmentedge.

We represent each process P_i as a circle, each resource type R_j as a rectangle. Since resource type R_j may have more than one instance. We represent each such instance as a dot within the rectangle. A request edge points to only the rectangle R_j. An assignment edge must also designate one of the dots in therectangle.

When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource, as a result, the assignment edge is deleted.

The sets P, R, E:

P= {P_1, P_2, P_3}

R= {R_1, R_2, R_3, R_4}

E= {P_1 ->R_1, P_2 ->R_3, R_1 ->P_2, R_2 ->P_2, R_2 ->P_1, R_3 ->P_3}

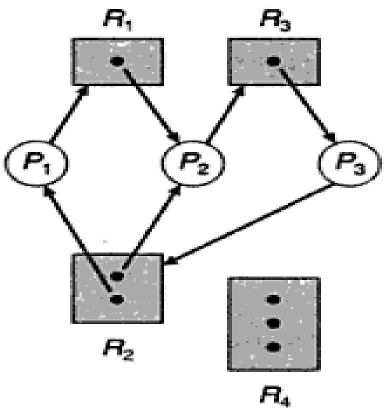


Figure Resource-allocation graph with a deadlock.

One instance of resource type R1
Two instances of resource type R2
One instance of resource type R3
Three instances of resource type R4

PROCESS STATES:

Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
Process P2 is holding an instance of R1 and an instance of R2 and is waiting for instance of R3.
Process P3 is holding an instance of R3.
If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.
Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3 ->R2 is added to the graph.
2 cycles:
P1 ->R1 ->P2 ->R3 ->P3 ->R2 ->P1
P2 ->R3 ->P3 ->R2 ->P2

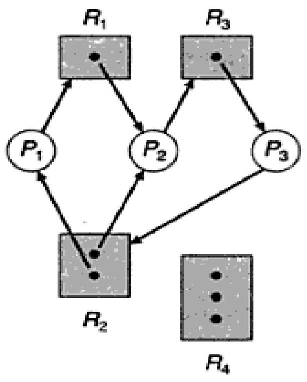


Figure Resource-allocation graph with a deadlock.

Processes P1, P2, P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3.process P3 is waiting for either process P1 (or) P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.

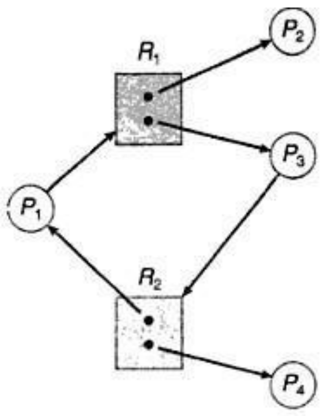


Figure Resource-allocation graph with a cycle but no deadlock.

We also have a cycle: P1 ->R1 ->P3 ->R2 ->P1
However there is no deadlock. Process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

DEADLOCK PREVENTION

For a deadlock to occur, each of the 4 necessary conditions must held. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

Mutual Exclusion – not required for sharable resources; must hold for non sharable resources

Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources

- Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none
- Low resource utilization; starvation possible

No Preemption –

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
- Preempted resources are added to the list of resources for which the process is waiting
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

Requires that the system has some additional *a priori* information available

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes .

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

System is in **safe state** if there exists a sequence <*P1, P2, …, Pn*> of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the *Pj*, with *j <I*

That is:

○ If Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished

○ When *Pj* is finished, *Pi* can obtain needed resources, execute, return allocated resources, and terminate

○ When *Pi* terminates, *Pi* +1 can obtain its needed resources, and so on If a system is in safe state no deadlocks

If a system is in unsafe state possibility of deadlock

Avoidance ensure that a system will never enter an unsafe state

Avoidance algorithms

Single instance of a resource type

○ Use a resource-allocation graph Multiple instances of a resource type

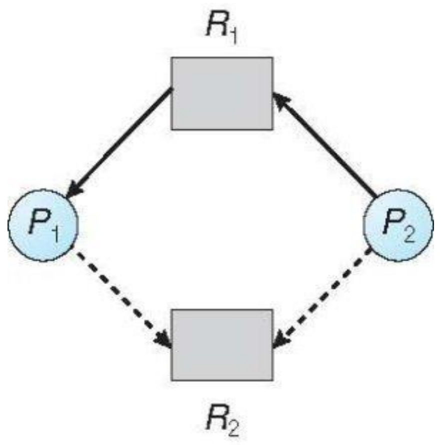
○ Use the banker's algorithm

Resource-Allocation Graph Scheme

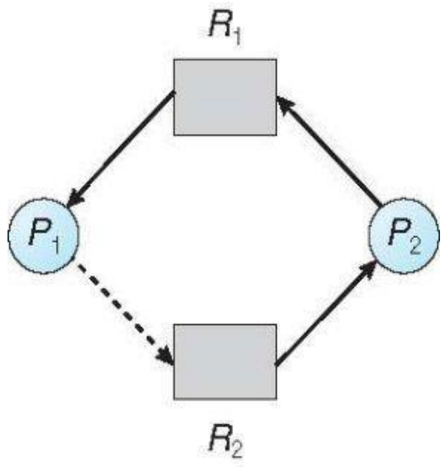
Claim edge*PiÆRj* indicated that process *Pj* may request resource *Rj*; represented by a dashed line

Claim edge converts to request edge when a process requests a resource

Request edge converted to an assignment edge when the resource is allocated to the process When a resource is released by a process, assignment edge reconverts to a claim edge Resources must be claimed *a priori* in the system



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

Multiple instances

Each process must a priori claim maximum use

When a process requests a resource it may have to wait

When a process gets all its resources it must return them in a finite amount of time Let *n* = number of processes, and *m* = number of resources types.

Available*:* Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resource type *Rj*available

Max*: n x m* matrix. If *Max* [*i,j*] = *k*, then process *Pi*may request at most *k* instances of resource type *Rj*

Allocation*: n* x *m* matrix. If Allocation[*i,j*] = *k* then *Pi* is currently allocated *k* instances of *Rj*

Need*: n* x *m* matrix. If *Need*[*i,j*] = *k*, then *Pi* may need *k* more instances of *Rj*to complete its task

Need [*i,j] = Max*[*i,j*] – *Allocation* [*i,j*]

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize: Work = Available

Finish [i] = false fori = 0, 1, …,n- 1

2. Find an isuch that both:

(a) Finish [i] = false

(b) Needi=Work

If no such iexists, go to step 4

3. Work = Work + $Allocation_i$

Finish[i] = true

go to step 2

4. IfFinish [i] == true for all i, then the system is in a safe state

Resource-Request Algorithm for Process *Pi*

Request = request vector for process *Pi*. If *Requesti*[*j*] = *k* then process *Pi* wants *k* instances of resource type *Rj*

1. If *Requesti*£*Needi*go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim

2. If *Requesti*£*Available*, go to step 3. Otherwise *Pi* must wait, since resources are not available

3. Pretend to allocate requested resources to *Pi* by modifying the state as follows:

Available = Available – Request;

Allocationi= Allocationi + *Requesti*;

Needi=Needi – Requesti;

- *If safe the resources are allocated to Pi*
- *If unsafe Pi must wait, and the old resource-allocation state is restored*

Example of Banker's Algorithm(REFER CLASS NOTES)

consider 5 processes *P*0 through *P*4; 3 resource types:

A (10 instances), *B* (5instances), and *C* (7 instances)

Snapshot at time *T*0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
*P*0	0 1 0	7 5 3	3 3 2
*P*1	2 0 0	3 2 2	
*P*2	3 0 2	9 0 2	
*P*3	2 1 1	2 2 2	
*P*4	0 0 2	4 3 3	

Σ The content of the matrix *Need* is defined to be *Max*

– Allocation Need

A B C

The system is in a safe state since the sequence <*P*1, *P*3, *P*4, *P*2, *P*0>

satisfies safety criteria

***P*1 Request (1,0,2)**
Check that Request £ Available (that is, (1,0,2) £ (3,3,2) true

Allocation	*Need*	*Available*
A B C	*A B C*	*A B C*
*P*0 0 1 0	7 4 3	2 3 0
*P*1 3 0 2	0 2 0	
*P*2 3 0 2	6 0 0	
*P*3 2 1 1	0 1 1	
*P*4 0 0 2	4 3 1	

Executing safety algorithm shows that sequence <*P*1, *P*3, *P*4, *P*0, *P*2> satisfies safety requirement

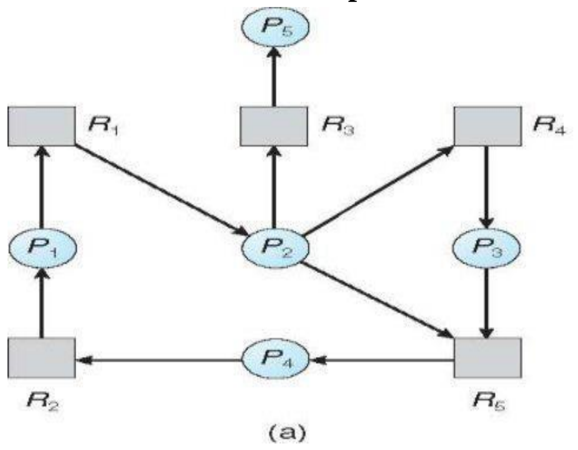
Deadlock Detection

Allow system to enter deadlock state
Detection algorithm
Recovery scheme

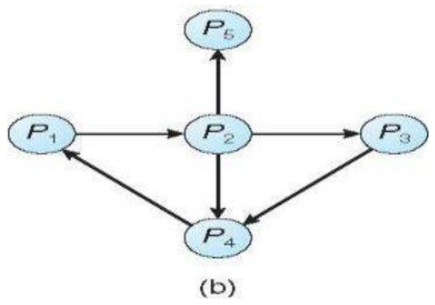
Single Instance of Each Resource Type

Maintain *wait-for* graph
Nodes are processes *Pi*Æ*P*
*j*if *Pi*is waiting for*Pj*
Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
An algorithm to detect a cycle in a graph requires an order of *n*2 operations, where *n* is the number of vertices in the graph

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph



Corresponding wait-for graph

Several Instances of a Resource Type

Available*:* A vector of length *m* indicates the number of available resources of each type. **Allocation***:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

Request*:* An *n* x *m* matrix indicates the current request of each process.

If *Request* [*i*][*j*] = *k*, then process *Pi* is requesting *k* more instances of resource type.*Rj*.

Detection Algorithm

Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if Allocationiπ 0, then

Finish[i] = false; otherwise, Finish[i] = true

2. Find an index isuch that both:

(a) Finish[i] == false

(b) Requesti£Work

If no such i exists, go to step 4

3. Work = Work + Allocation$_i$

Finish[*i*] = *true*

go to step 2

4. If *Finish*[*i*] == false, for some *i*, 1 £*i*£*n*, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then *Pi* is deadlocked

Recovery from Deadlock:

Process Termination

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

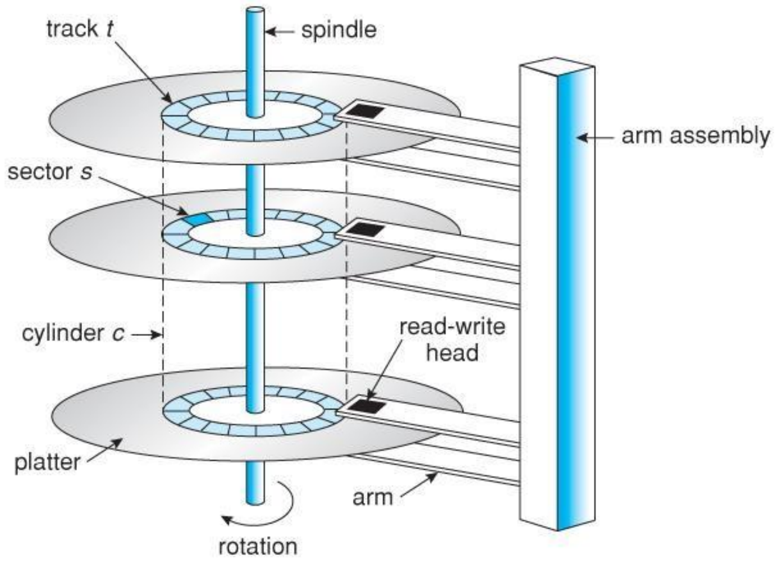
Rollback – return to some safe state, restart process for that state

Starvation – same process may always be picked as victim, include number of rollback in cost factor

Secondary storage structure:

Overview of mass storage structure

Magnetic disks: Magnetic disks provide the bulk of secondary storage for modern computer system. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 inches. The two surfaces of a platter are covered with a magnetic material. We store information by it magnetically on the platters.



Moving head disk mechanism

A read /write head files just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are sub divided into sectors. The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.

When the disk in use, a driver motor spins it at high speed. Most drivers rotate 60 to 200 times per second. Disk speed has 2 parts. The transfer rate is the at which data flow between the drive and the computer. To read/write, the head must be positioned at the desired track and at the beginning of the desired sector on the track, the time it takes to position the head at the desired track is called seek time. Once the track is selected the disk controller waits until desired sector reaches the read/write head. The time it takes to reach the desired sector is called ***latency time or rotational dealy-access time***. When the desired sector reached the read/write head, then the real data transferring starts.

A disk can be removable. Removable magnetic disks consist of one platter, held in a plastic case to prevent damage while not in the disk drive. Floppy disks are in expensive removable magnetic disks that have a soft plastic case containing a flexible platter. The storage capacity of a floppy disk is 1.44MB.

A disk drive is attached to a computer by a set of wires called an I/O bus. The data transfer on a bus are carried out by special processors called controllers. The host controller is the controller at the computer end of the bus. A disk controller is built into each disk drive . to perform i/o operation, the host controller operates the disk drive hardware to carry out the command. Disk controllers have built in cache, data transfer at the disk drive happens b/w cache and disk surface. Data transfer at the host, occurs b/w cache and host controller.

Magnetic Tapes: magnetic tapes was used as an early secondary storage medium. It is permanent and can hold large amount of data. It access time is slow compared to main memory and magnetic disks. Tapes are mainly used for back up, for storage of infrequently used information. Typically they store 20GB to 200GB.

Disk Structure: most disks drives are addressed as large one dimensional arrays of logical blocks. The one dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. sector 0 is the fist 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 cylinder from outermost to inner most. As we move from outer zones to inner zones, the number of sectors per track decreases. Tracks in outermost zone hold 40% more sectors then innermost zone. The number of sectors per track has been increasing as disks technology improves, and the outer zone of a disk usually has several hundred sectors per track. Similarly, the number of cylinders per disk has been increasing; large disks have tens of thousands of cylinders.

Disk attachment

Computer access disk storage is 2 ways.

1. Via I/O ports(host attachedstorage)
2. Via a remote host in a distributed file system(network attachedstorage).

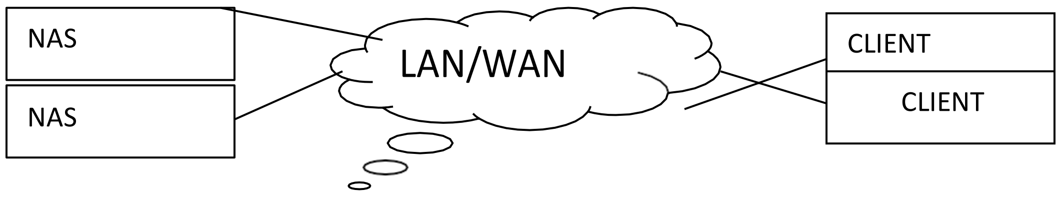
1 .Host attached storage : host attached storage are accessed via local I/O ports. The desktop pc uses an I/O bus architecture called IDE. This architecture supports maximum of 2 drives per I/O bus. High end work station and servers use SCSI and FC.

SCSI is an bus architecture which have large number of conductor's in a ribbon cable (50 or 68) scsi protocol supports maximum of 16 drives an bus. Host consists of a controller card (SCSI Initiator) and upto 15 storage device called SCSI targets.

Fc(fiber channel) is the high speed serial architecture. It operates mostly on optical fiber (or) over 4 conductor copper cable. It has 2 variants. One is a large switched fabric having a 24-bit address space. The other is an (FC-AL) arbitrated loop that can address 126 devices.

A wide variety of storage devices are suitable for use as host attached.(hard disk,cd ,dvd,tape devices)

2. Network-attached storage: A(NAS) is accessed remotely over a data network .clients access network attached storage via remote procedure calls. The rpc are carried via tcp/udp over an ip network-usually the same LAN that carries all data traffic to theclients.



NAS provides a convenient way for all the computers on a LAN to share a pool of storage with the same ease of naming and access enjoyed with local host attached storage .but it tends to be less efficient and have lower performance than direct attached storage.

3. Storage area network: The drawback of network attached storage(NAS) is storage I/O operations consume bandwidth on the data network. The communication b/w servers and clients competes for bandwidth with the communication among servers and storagedevices.

A storage area network(SAN) is a private network using storage protocols connecting servers and storage units. The power of a SAN is its flexibility. multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. SANs make it possible for clusters of server to share the same storage

Disk Scheduling Algorithms

Disk scheduling algorithms are used to allocate the services to the I/O requests on the disk . Since seeking disk requests is time consuming, disk scheduling algorithms try to minimize this latency. If desired disk drive or controller is available, request is served immediately. If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the Operating System has to choose which pending request to service next. The OS relies on the type of algorithm it needs when dealing and choosing what particular disk request is to be processed next. The objective of using these algorithms is keeping Head movements to the amount as possible. The less the head to move, the faster the seek time will be. To see how it works, the different disk scheduling algorithms will be discussed and examples are also provided for better understanding on these different algorithms.

1. First Come First Serve(FCFS)

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processes according to their arrival. The request arrives first will be accessed and served first. Since it follows the order of arrival, it causes the wild swings from the innermost to the outermost tracks of the disk and vice versa . The farther the location of the request being serviced by the read/write head from its current location, the higher the seek time will be.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement (THM) of the read/write head :

95, 180, 34, 119, 11, 123, 62, 64

Consider that the read/write head is positioned at location 50. Prior to this track location 199 was serviced. Show the total head movement for a 200 track disk (0-199).
Solution:

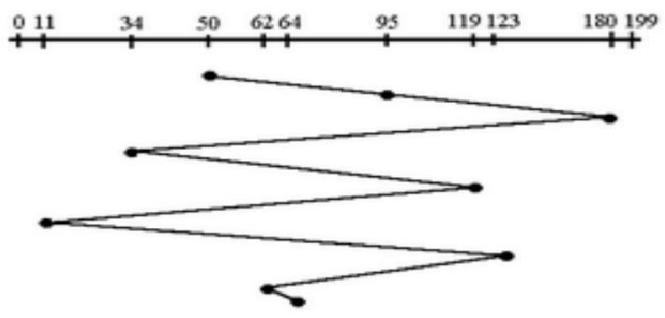


Fig. FCFS Representation

Total Head Movement Computation: (THM) =

(180 - 50) + (180-34) + (119-34) + (119-11) + (123-11) + (123-62) + (64-62) =

130 + 146 + 85 + 108 + 112 + 61 + 2 (THM) = 644 tracks

Assuming a seek rate of 5 milliseconds is given, we compute for the seek time
using the formula: Seek Time = THM * Seek rate
=644 * 5 ms
Seek Time = 3,220 ms.

2. Shortest Seek Time First(SSTF):

This algorithm is based on the idea that that he R/W head should proceed to the track that is closest to its current position . The process would continue until all the track requests are taken care of. Using the same sets of example in FCFS the solution are as follows:

Solution:

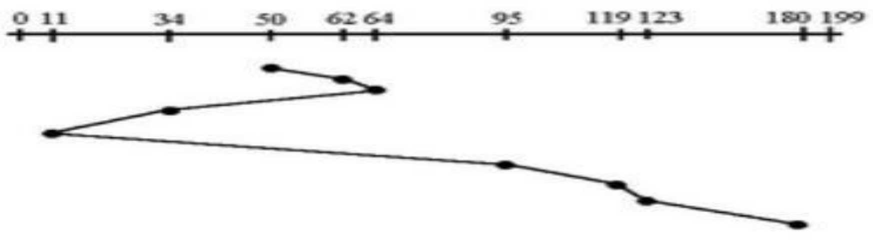


Fig. SSTF Representation

(THM) = (64-50) + (64-11) + (180-11) =

14 + 53 + 169 (THM) = 236 tracks

Seek Time = THM * Seek rate

= 236 * 5ms
Seek Time = 1,180 ms
In this algorithm, request is serviced according to the next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34 . The process would continue up to the last track request. There are a total of 236 tracks and a seek time of 1,180 ms, which seems to be

a better service compared with FCFS which there is a chance that starvation3 would take place. The reason for this is if there were lots of requests closed to each other, the other requests will never be handled since the distance will always be greater.

3. *SCAN Scheduling Algorithm*

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm. Using the same sets of example in FCFS the solution are as follows:

Solution:

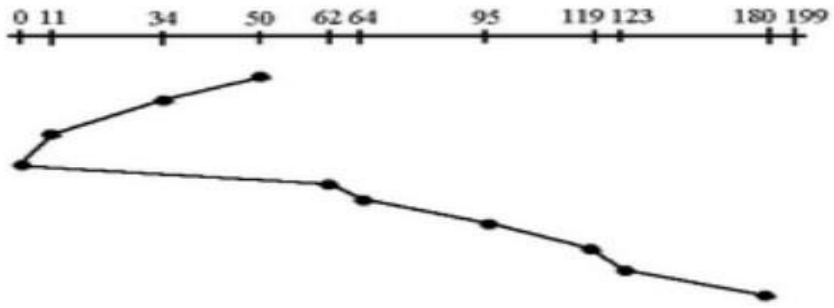


Fig. SCAN Representation

(THM) = (50-0) + (180-0)
= 50 + 180
(THM) = 230

*Seek Time = THM * Seek rate*
*= 230 * 5ms*
Seek Time = 1,150 ms

This algorithm works like an elevator does. In the algorithm example, it scans down towards the nearest end and when it reached the bottom it scans up servicing the requests that it did not get going down. If a request comes in after it has been scanned, it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks and a seek time of 1,150. This is optimal than the previous algorithm.

4 .Circular SCAN (C-SCAN)Algorithm

This algorithm is a modified version of the SCAN algorithm. C-SCAN sweeps the disk from end-to-end, but as soon it reaches one of the end tracks it then moves to the other end track without servicing any requesting location. As soon as it reaches the other end track it then starts servicing and grants requests headed to its direction. This algorithm improves the unfair situation of the end tracks against the middle tracks. Using the same sets of example in FCFS the solution are as follows:

Solution:

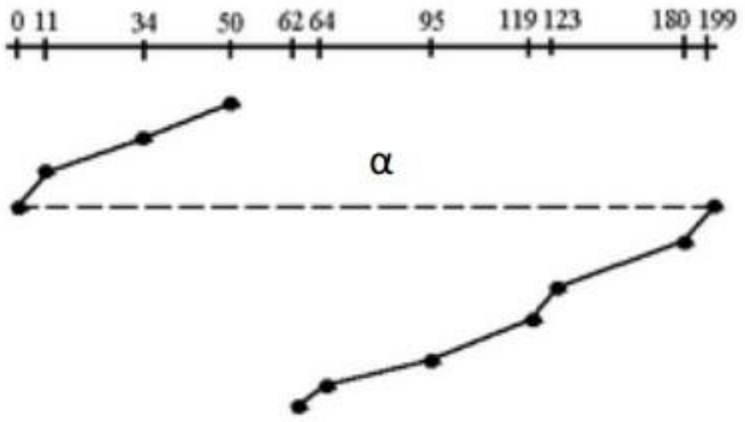


Fig. C-SCAN Representation

Notice that in this example an alpha3 symbol (α) was used to represent the dash line. This return sweeps is sometimes given a numerical value which is included in the computation of the THM . As analogy, this can be compared with the carriage return lever of a typewriter. Once it is pulled to the right most direction, it resets the typing point to the leftmost margin of the paper . A typist is not supposed to type during the movement of the carriage return lever because the line spacing is being adjusted . The frequent use of this lever consumes time, same with the time consumed when the R/W head is reset to its starting position.

Assume that in this example, α has a value of 20ms, the computation would be as follows: (THM) = (50-0) + (199-62) + α
= 50 + 137 + 20 (THM)

= 207 tracks

Seek Time = THM * Seek rate

= 187 * 5ms Seek Time = 935 ms .

The computation of the seek time excluded the alpha value because it is not an actual seek or search of a disk request but a reset of the access arm to the starting position .

Disk management

Disk formatting*:* A magnetic disk is a blank slate. It is just a platter of **a magnetic recording material.** before a disk can store data , it must be divided into sectors that the disk controller can read and write. This process is called low level formatting (or)physical formatting. low level formatting fills the disk with a special data structure for each sector .the Data structure **for a sector** typically consists of a header, a data **area**, a trailer . the header and trailer contain information used by the disk controller ,such as a sector number and a**n** error correcting code(ECC). When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area . when the sector is read ,the EC**C** is recalculated and compared with the stored value. If the stored and calculated **numbers** are different, this mismatch indicates that the data area of this sector has become corrupted, and that the disk **sector** may be bad. ECC contains enough information, **if** only few bits of data have been corrupted, to enable the controller to identify which bits have changed and calculate what their correct values should be. The controller automatically does the ECC processing what ever a sector is read/written for many hard disks, when the disk controller is instructed to low level format the disk, it can also be told how many bytes of data space to leave between the header and trailer of all sectors.

Before it can use a disk to hold files , OS still needs to record its own data structures on the disk. It does in 2 steps. The first step is to partition the disk in to one/more groups of cylinders. OS can treat each partition as a separate disk. The second step is logical formatting (or)creation of file system. In this step, OS stores the initial File system **d**ata structures on to the disk. These data structures include maps of free and allocate space and initial empty director**y**.

***Boot block*:-**

When a computer is powered up -it must have an initial program to run. This initial bootstrap program initializes all aspects of the system, from CPU registers to device controllers, and the contents of main memory, and then starts the OS. To do its job, the bootstrap program finds the OS kernel on disk, loads that kernel into memory and jumps to an initial address to begin the OS execution. For most computers, the bootstrap is stored in ROM. This location is convenient, because ROM needs no initialization and is at a fixed location that the CPU can start executing when powered up, ROM is read only, it cannot be infected by computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips. For this reason, most systems store a tiny bootstrap loader program in the boot ROM whose job is to bring in a full bootstrap program from disk. The full bootstrap program is stored in the boot blocks at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk. The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.

Bad blocks:-

A Block in the disk damaged due to the manufacturing defect or virus or physical damage. This defector block is called Bad block. MS-DOS format command, scans the disk to find bad blocks. If format finds a bad block, it tells the allocation methods not to use that block. Chkdsk program search for the bad blocks and to lock them away. Data that resided on the bad blocks usually are lost. The OS tries to read logical block 87.
The controller calculates ECC and finds that the sector is bad. It reports this finding to the OS. The next time the system is rebooted, a special command is run to tell the SCS controller to replace the bad sector
with a spare.
After that, whenever the system requests logical block 87, the request is translated into the replacement sectors address by the controller.

Sector slipping:-

Logical block 17 becomes defective and the first available spare follows sector 202. Then, sector slipping remaps all the sectors from 17 to 202, sector 202 is copied into the spare, then sector 201 to 202, 200 to 201 and so on. Until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18.

Swap space management:-

System that implements swapping may use swap space to hold an entire process image, including the code and data segments. Paging systems may simply store pages that have been pushed out of main memory. Note that it may be safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort processes. Overestimation wastes disk space that could otherwise be used for files, but it does no other harm. Some systems recommend the amount to be set aside for swap space. Linux has suggested setting swap space to double the amount of physical memory. Some OS allow the use of multiple swap spaces. These swap spaces as put on separate disks so that load placed on the (I/O) system by paging and swapping can be spread over the systems I/O devices.

Swap space location:-

A Swap space can reside in one of two places. It can be carved out of normal file system (or) it can be in a separate disk partition. If the swap space is simply a large file, within the file system, normal file system methods used to create it, name it, allocate its space. It is easy to implement but inefficient. External fragmentation can greatly increase swapping times by forcing multiple seeks during reading/writing of a process image. We can improve performance by caching the block location information in main memory and by using special tools to allocate physically contiguous blocks for the swap file. Alternatively, swap space can be created in a separate raw partition. a separate swap space storage manager is used to allocate
/deal locate the blocks from the raw partition. this manager uses algorithms optimized for speed rather than storage efficiency. Internal fragmentation may increase but it is acceptable because life of data in swap space is shorter than files. since swap space is reinitialized at boot time, any fragmentation is short lived. the raw partition approach creates a fixed amount of swap space during disk partitioning adding more swap space requires either repartitioning the disk (or) adding another swap space elsewhere.