Deadlock

A process is said to be deadlocked if it is waiting for some resource which is being used by another process and so on. A process requests resources & if resources are not available at that time, the process enters a wait state.

It may happen that a waiting process may never change state bcoz the resources they have requested are held by other waiting processes. This is called Deadlock.

A process may utilize resource ~~order the~~ in 'the sequence',

Request – Use – Release → system call
↓ Request: system call
↓ Use: operate on resource.

Deadlock characterization

1. Mutual Exclusion :- Atleast one resource must be held in nonsharable mode i.e only one process can at a time use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released

2. Hold & wait :- A process must be holding one resource & waiting to acquire additional resources that are currently being held by other processes.

Date: / /
Page No.

3. No preemption :- Resources can't be preempted i.e a resource can be released only by process holding it, after that process has completed its task.

4. Circular wait :- A set $\{P_0, P_1 \ldots P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, .. P_{n-1} is waiting for a resource that is held by P_n & P_n is waiting for resource that is held by P_0.

② Resource Allocation Graph :- Represented in terms of directed graph. It consists of set of vertices V & a set of edges E.

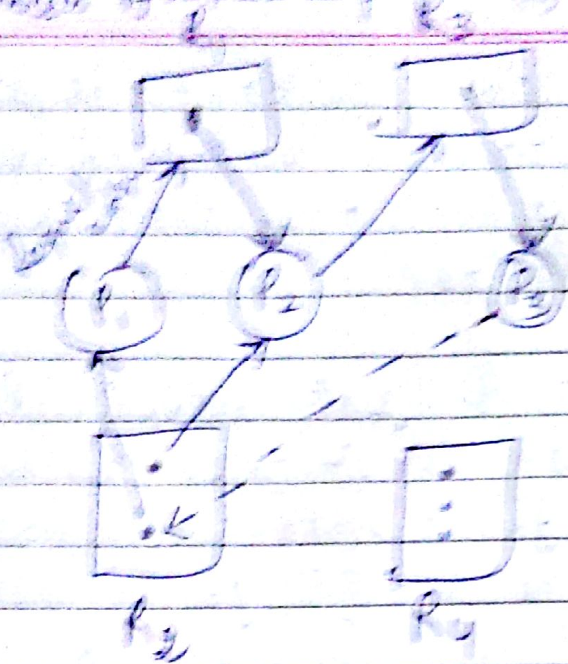
Vertices are divided into 2 diffn types of nodes $P = \{P_1, P_2, \ldots P_n\}$ → all active processes in the system

& $R = \{R_1, R_2 \ldots R_m\}$ set consisting of all resource types in the system.

[Request Edge] $P_i \rightarrow R_J$ [means Process P_i requested an instance of resource type R_j & is waiting for that resource].

[Assignment Edge]. $R_J \rightarrow P_i$ [means an instance of resource type R_J has been allocated to Process P_i]

Resource request by P1 & P2 takes place in the order.
(1) Process P1 requests the tape (3) Process P1 [illegible] the printer
(2) Process P2 requests the printer (4) Process P2 [illegible] the tape



✓ Here, Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of R_1.

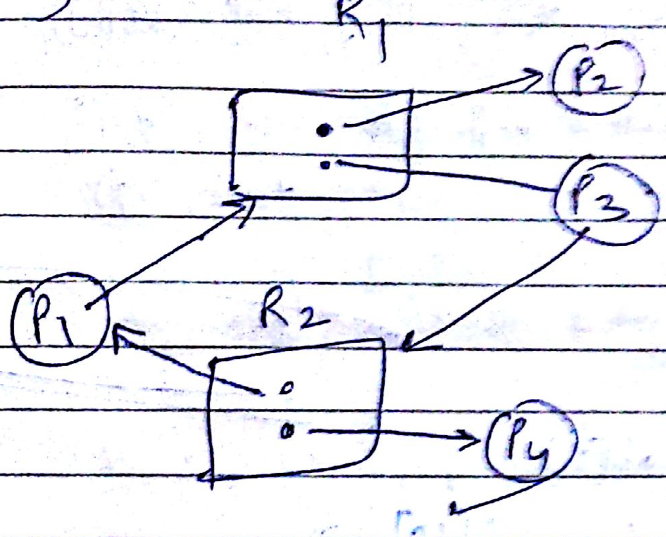
✓ Process P_2 is holding an instance of R_1 & R_2 & is waiting for an instance of resource type R_3.

✓ Process P_3 is holding an instance of R_3.

If the graph contains no cycle, then no process in the system is deadlocked.
If the cycle is there, then a deadlock may exist.
A cycle is necessary & sufficient condn for existence of deadlock
If several instances in a cycle - then not necessarily deadlock has occurred. If one instance of each resource in the cycle implies that a deadlock has occurred.

The first two (1) & (2) requests are granted immediately i.e. a tape & a printer exists in the system. now Process Pi holds the tape & Pj holds the printer. when Pi asks for the printer, it ~~is blocked until Pj releases the printer.~~ wait [illegible]

```
Printf ("%s %d %d", P[i], et[i], wt[i]);
Printf ("%d %f", tot, avg);
getch();
}
```



R.A.G with cycle but no deadlock

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

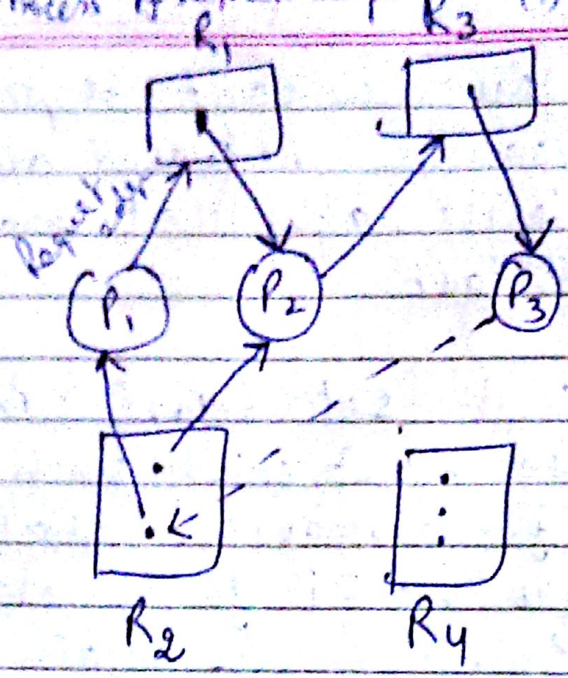
If P_4 releases R_2, then no deadlock.

Eg A system requests one tape & one printer & two processes Pi & Pj that use these resources as follows :-

Process Pi	Process Pj
Request Tape	Request Printer
Request Printer	Request Tape
Use Tape & Printer	Use Tape & Printer
Release Printer	Release Printer
Release Tape	Release Printer

Show that set of processes {Pi, Pj} is in a deadlock state.

Sol^n. Resource Request by P_i & P_j takes place in the order
(1) Process P_i requests the tape (3) Process P_i requests the printer
(2) Process P_j requests the printer (4) Process P_j requests the tape



✓ Here, Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of R_1.

✓ Process P_2 is holding an instance of R_1 & R_2 & is waiting for an instance of resource type R_3.

✓ Process P_3 is holding an instance of R_3.

If the graph contains no cycle, then no process in the system is deadlocked.
If the cycle is there, then a deadlock may exist.
A cycle is necessary & sufficient cond^n for existence of deadlock:
If several instances in a cycle – then not necessarily deadlock has occurred. If one instance of each resource,
(ii) Then cycle implies that a deadlock has occurred.

Methods for Handling Deadlocks

1. We can use a protocol to prevent or avoid deadlocks, ensuring that system will never enter a deadlock state.
2. We can allow the system to enter a deadlock state, detect it & recover it.
3. We can ignore the problem altogether & pretend that deadlock never occur in the system. This sol^n is used by most O.S. including UNIX.

To ensure deadlock never occur in the system, use Deadlock prevention
↓
It is a set of methods for ensuring that atleast one of the necessary cond^n can't hold.

Deadlock prevention – Four cond^n should not arise simultaneously.

a) Mutual Exclusion:- It must hold for nonshareable resources. Eg. A printer can't be simultaneously shared by several processes, shareable resources, don't require mutually exclusive access, & thus can't be involved in a deadlock.
eg Read only files.

b) Hold & Wait:- To ensure that hold & wait should never occur in the system, it is reqd. that whenever a process requires a resource, it doesn't hold any other resource.

i) One protocol used is – the resources can be requested & allocated to a process before it begins execution. [system call generation].

ii) Second protocol says – A process may request resources when it has none & use them. Before it requests another resources, it should release all resources that it is currently allocated.

III

Disadv:-

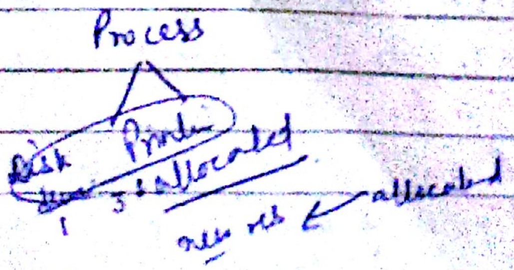
1. <u>Resource Utilization</u> - is low since many of the resources may be allocated but unused for a long pd. of time.
2. <u>Starvation</u>:

3. <u>No preemption</u> :- → To ensure that this condⁿ should not hold, we canuse:- If a process is holding some resources & requests another resources that can't be immediately allocated to it, then all the resources being held are preempted. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well the new ones that it is requesting.

<u>Second protocol</u> :- If a process requests some resources, we check if they are available. If they are, we allocate them.

<u>This is applied to</u> :- CPU registers & mem. space
<u>not applied to</u> :- Printers & Tape drives.

4. <u>Circular Wait</u>:- To ensure that C.W never holds, we may impose a total ordering of all resource types & each process request resources in an increasing order of enumeration.
In this method, resources can be numbered. Suppose we have 3 resources types: tape drive, disk drive, & printer. A process which meant to access disk drive & printer should firstly access disk drive & then printer.



(<) 9

So, rule can be extended to say that once a process has some resources allocated to it, it can be allocated a new resource only if no. of all its allocated resources is less than the no. assigned to the requested resource.

tape drive = 1
disk n = 2
Printer = 6

TD Printer
1 6 < 2
✗
Can't be allocated

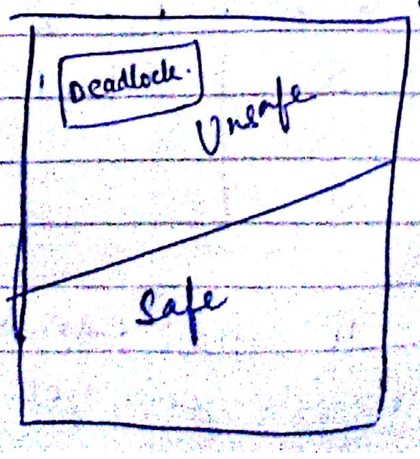
1 2 < 6
✓
yes allocated

Deadlock Avoidance :- In this, we can have addn information about how the resources are requested. Each request requires that the system considers the resources currently available, the resources currently allocated to each process & the future requests & releases of each process, to decide whether the current request can be satisfied or must wait to avoid a deadlock.

It ensures that circular wait never occurs in a system.

Safe state :- A state is safe if the system can allocate resources to each process (upto its maxm) & still avoid a deadlock.

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



Scanned by CamScanner

Ques. consider a system with 12 tape drives & 3 processes

Process	Max^m need	Current Need/Allocated	Available
P_0.	10.	5.	12
P_1.	4	2.	
P_2.	9.	2.	

(a)

12 → Available Tape drives

At time t0 process P_0 is holding 5 tapes
P_1 holds 2 tapes
P_2 holds 2 tapes
3 are free.

whether this system is in deadlock state or not ?

If the sequence is $< P_1, P_0, P_2 >$

$2 + 2 = 4$.

Available = $4 + 1 = \underline{5}$

$$P_0 = \overset{CN}{5} + \overset{Max^m\ need}{5}$$

$= \underline{10}$ (Release)

Now P_2 can be allocated

ⓑ What will happen if at time t1, Process P_2 requests & is allocated one more tape drive

Process	M.N	C.N	Available
P_0	10	5	12
P_1	4	2	
P_2	9	3	

<u>2 free</u> only, P_1 request can be granted.

④ Now, the system goes in deadlock state.

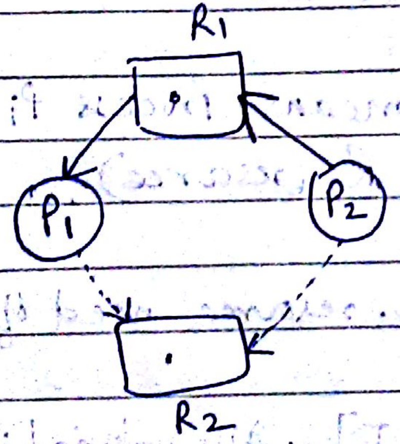
RAG for Deadlock Avoidance :-

~~If~~ This is applicable for single instance of each resource type.

In this, a new edge i.e claim edge is there. $P_i \rightarrow R_j$ [future request shown by dashed line]. It means Process P_i may request resource R_j in the near future.

When Process P_i will request R_J → claim edge will be converted to request edge.

Also, when resource will be released by P_i, the allocation edge $R_J \rightarrow P_i$ will be converted to a claim edge.



Suppose P_2 requests R_2. Though R_2 is free but can't allocate to P_2 since this will create a cycle in the graph.

Banker's Algorithm:- It is applicable when multiple instances of resources are available. It is less efficient. It is used in banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.

Let 'n' → no. of processes in the system
'm' → no. of resource types.

① **Available :-** available [J] = k means k instance of resorce type R_J are available.

'm' → tells no. of available resources of each type.

② **Max :-** n×m → max^m demand of each process in the system.

Max [i, J] = k means Process P_i may request atmost k instances of resorce type R_J.

③ **Allocation :-** n×m defines no. of resources of each type currently allocated to each process.

If allocation [i, J] = k means; Process P_i is allocated k instances of R_J (resorce).

④ **Need :-** n×m indicates rem. resorce need of each process.

need [i, J] = Max [i, J] − Allocation [i, J].

need [i, J] = k means Process P_i needs k instances of resorce type R_j.

2 Algo used
1) Safety Algo
2) Resorce Req. Algo

Q Consider a system with five Processes P_0 through P_4 & 3 res. types A, B, C. Resource type A has 10 instance
B - 5 "
C - 7 "

Suppose at time t_0, the snapshot has been taken

Process	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			

(i) What will be the content of Need matrix.

Sol^n

Need [i, J] = Max [i, J] - Allocation [i, J].

Need [P_0, A] = 7 - 0 = 7
[P_0, B] = 5 - 1 = 4
[P_0, C] = 3 - 0 = 3

Need	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

(ii) Is the system is in a safe state? If yes, what is the safe sequence.

Sol^n Applying Safety Algo., we have
For P_i If Need ≤ Available then P_i is in safe sequence. then Available = Available + Allocation

for P_0 Need is 7 4 3
Available is 3 3 2.
Need ≤ Available
[No, false cond^n so P_0 must wait].

for P_1 = 1 2 2 < 3 3 2
True Av = Av + All
= 3 3 2 + 2 0 0 = 5 3 2

For P_2 Need - 6 0 0
Available - 5 3 2
So, Need $\not<$ Available
cond^n is false & P_2 must wait.

for P_3 (i=3) Need = 0 1 1
Available = 5 3 2
0 1 1 < 5 3 2
So cond^n is true
Av = Av + All
= 5 3 2 + 2 1 1 = 7 4 3

for P_4
Need = 4 3 1 < 7 4 3
Again safe, ∴ AV = Av + All
= 7 4 3 + 0 0 2 = 7 4 5
Now, P_0 & P_2 are in waiting state.
- Take P_2 Need = 6 0 0 < 7 4 5
So, AV = Av + All
~~6 0 0 + 7 4 5 = 1 3 4 5~~
7 4 5 + 3 0 2 = 10 4 7

next P_0 Need 7 4 3 < 10 4 7
So, Av = Av + All
= 10 4 7 + 0 1 0 = 10 5 7

Seq < P_1, P_3, P_4, P_2, P_0 >.
& System is in safe state.

iii) What will happen if Process P_1 requests one addn instance of res. type A & two instances of resource type C.

Soln

Request P_i = (1 0 2)

To decide whether this request is immediately granted we first chk. that

Request $\le$ Available

ie (1,0,2) $\le$ (3,3,2)

This holds true.

So, request granted.

To confirm, that req. has been granted we chk the new state by applying safety algo, that our System is in safe state or not.

✓ Available = Av − Request

= (3,3,2) − (1,0,2)

= (2,3,0)

~~Allocation = Allocation + Req.~~

~~(0,1,0) + (1,0,2) = (1,1,2)~~

(7,5,3) − (1,1,2) = 6,4,1

Allocation$_i$ = Allocation$_i$ + Request$_i$

(for P_1) (2,0,0) + (1,0,2) = (3,0,2)

Need = Need$_i$ − Request$_i$ = (1,2,2) − (1,0,2) = (0,2,0)

Process	Allocation A B C	Need A B C	Available 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	

Now, applying safety algo, safe sequence is $< P_1, P_3, P_4, P_0, P_2 >$ so, request can be immediately granted.

Applying safety Algorithm, we have

P_0 = $Need_i \leq Available_i$

743 ≤ 230 Not in Safe Sequence

✓ P_1 = 0 20 ≤ (2 3 0) safe sequence

Av = Av + All

= 230 + 302 = 532

P_2 = 600 ≤ 532 false

✓ P_3 = 0 11 ≤ 5 3 2 safe sequence.

Av = Av + Allocation

= 532 + 211 = 743

✓ P_4 = 431 ≤ 743 safe sequence

Av = Av + All

= 743 + 002 745

$P_0 = 7\ 4\ 3 \leq 7\ 4\ 5$ safe seq.

$Av = Av + All$

$= 7\ 4\ 5 + 0\ 1\ 0 = 7\ 5\ 5$

$P_2 = 6\ 0\ 0 \leq 7\ 5\ 5$ safe.

$Av = Av + All$

$= 7\ 5\ 5 + 3\ 0\ 2 \quad 10\ 5\ 7$

(iv) If a request (3, 3, 0) by Process P_4 arrives in the state defined by (iii), can it be granted immediately.

Chk. Req. ≤ Available

$(3, 3, 0) \leq (2, 3, 0)$ Not granted.

(v) If a request (0, 2, 0) by Process P_0 arrives then chk. whether it is granted or not?

Sol^n Req ≤ Available

$(0, 2, 0) \leq (2, 3, 0)$

Req. granted.

If it is granted, then new state of the System be defined as:-

$$\begin{array}{r} 2\ 3\ 0 \\ 0\ 2\ 0 \\ \hline 2\ 1\ 0 \end{array}$$

$$\begin{array}{r} 0\ 1\ 0 \\ 0\ 2\ 0 \\ \hline 0\ 3\ 0 \end{array}$$

$$\begin{array}{r} 7\ 4\ 3 \\ 0\ 2\ 0 \\ \hline 7\ 2\ 3 \end{array}$$

$Av = Av - Req_0$

$(2,3,0) - (0,2,0)$

$= (2,1,0)$

$All = All + Req_0$

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

$= (0,3,0)$

$Need = Need_0 \cancel{+} - Req_0$

$(7,4,3) - (0,2,0) = (7,2,3)$

Process	Allocation A	Allocation B	Allocation C	Need A	Need B	Need C	Available A	Available B	Available C
P_0	0	3	0	7	2	3	2	1	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			

~~Applying Safety Algo on this new state~~

All five processes are in waiting state as condⁿ Need ≤ Available is not satisfied.

So, req. is not granted.

Recovery from deadlock

1. To abort one or more processes to break a circular wait.
   - Abort all deadlocked processes.
   - Abort one process at a time until the deadlock cycle is eliminated.

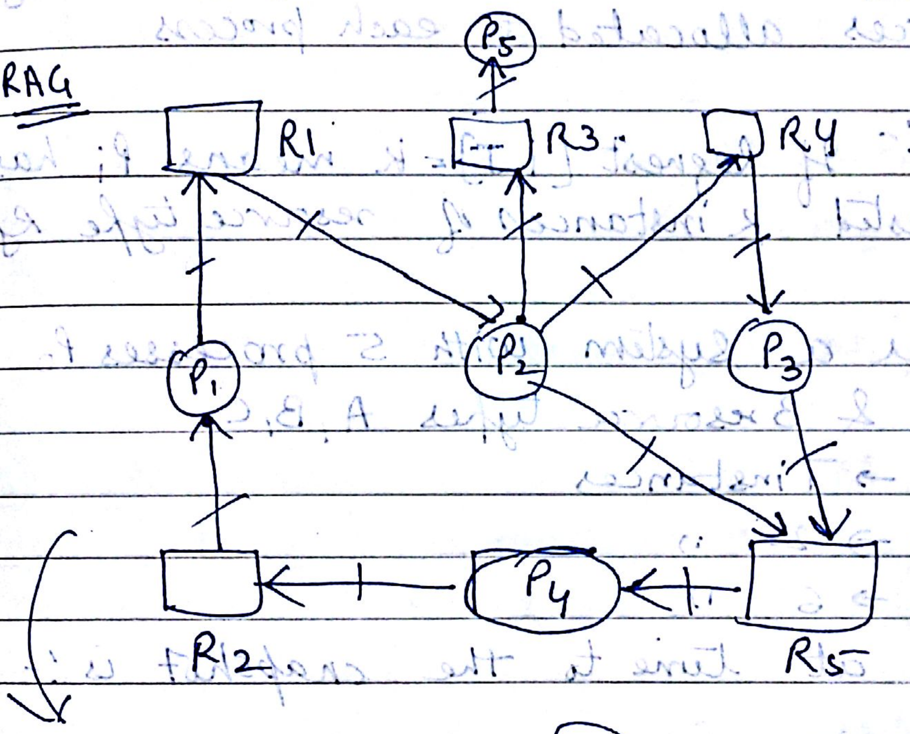
2. To preempt some resources from one or more of the deadlocked processes.
   - Selecting a victim :- Which resource & which process is to be preempted.
   - Rollback :- If we preempt a resource from a process then it can't continue with the normal execution; it is missing some needed resources. We must roll back the process to some safe state & restart it from that state.
   - Starvation :- If the victim is decided only on cost factor, the same process will be picked again & again leading to starvation. So, a process can be picked as a victim only a (small) finite no. of times.

Deadlock Detection :- Detect deadlock & recover from it.

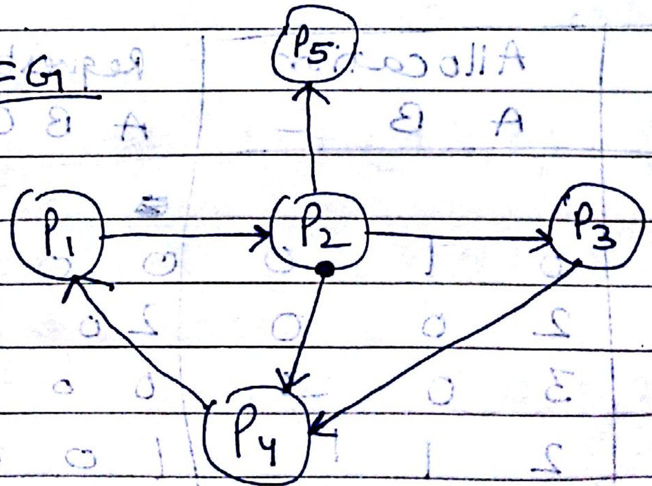
1) Single Instance of Each Resource type

Wait - for Graph exists in replacement of RAG.
$P_i \rightarrow P_J$ [means process P_i is waiting for process P_J to release a resource that P_i needs].

RAG



Corresponding WFG



It exists only if RAG contains $P_i \rightarrow R_q$ & $R_q \rightarrow P_j$ for some resources R_q. Now detect cycle in WFG & if it exists then deadlock is there.

2) Several Instances of a Resource type → WFG is applicable for single instance of resource.

Available :- m indicates no. of resources available.

Allocation :- n×m matrix defines no. of resources allocated to each process

Request :- If Request [i, J] = k means P_i has requested k instances of resource type R_j.

Ques Consider a system with 5 processes P_0 thru P_4 & 3 resource types A, B, C.

A → 7 instances
B → 2 "
C → 6 "

Suppose at time t_0 the snapshot is :-

Process	Allocation A	Allocation B	Allocation C	Request A	Request B	Request C	Available A	Available B	Available C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

(i) Is the system deadlocked

(ii) Suppose P_2 makes an addⁿ req $(0,0,1)$ what will be the effect of this request to the system?

Solⁿ If Request $\leq$ Available is true, then P_i is is safe state & $Av = Av + Allocation$. else P_i must wait

for $i=0$ Request$_0$ $= (0,0,0)$

P_0 Available $= (0,0,0)$

$\therefore$ True $Av = Av +$ Allocation

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

P_1 $(2,0,2) \leq (0,1,0)$ No, false; wait

P_2 $(0,0,0) \leq (0,1,0)$ Yes

$Av = Av +$ Allocation

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

P_3 $(1,0,0) \leq (3,1,3)$ Yes

$Av = Av +$ All

$= (3,1,3) + (2,1,1) = (5,2,4)$

P_4 $(0,0,0) \leq (5,2,4)$ yes

$Av = Av +$ All

$= (5,2,4) + (0,0,2) = (5,2,6)$

Again

P[illegible] $(2,0,2) \leq (5,2,6)$ Yes

$Av = Av +$ All

$= (5,2,6) + (2,0,0) = (7,2,6)$

↓

same as available.

So, given system is not in deadlock state
< P_0, P_2, P_3, P_4, P_1 >.

(ii) When P_2 makes an addⁿ request of (0,0,1)
table can be modified as

	A	B	C (Req)
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

Then deadlock will occur.

Ques Consider a system with processes P_0 thru P_4 & A, B, C resource. Snapshot is:

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	1	1	2	4	3	3	2	1	0
P_1	2	1	2	3	2	2			
P_2	4	0	1	9	0	2			
P_3	0	2	0	7	5	3			
P_4	1	1	2	11	2	3			

a) Determine the total amount of resource of each type

b) Content of Need Matrix

c) Determine if this state is safe using safety algo. [Unsafe]

d) Starting with the allocation resource state given above, suppose the current request for each process is given by Request matrix below. Further assume that the requests are granted

Req. Matrix	A	B	C
P_0	3	3	1
P_1	1	1	0
P_2	6	0	1
P_3	7	2	3
P_4	0	1	1

seq

$< P_1, P_4, P_0, P_2, P_3 >$

Will the system be in a deadlock state Determine using the "deadlock detection algorithm".

e) What, if anything, does this problem demonstrate about the rel^n b/w the "safeness" of an allocation state & deadlock itself.

Ans Even if we start from an unsafe state, deadlock doesn't necessarily occur. A safe state is a sufficient cond^n for the absence of deadlock, but not necessary cond^n.

Disk Scheduling :- The major responsibility of O.S. is to use the h/w in an efficient manner. For the disk drivers, meeting this responsibility means having a fast access time & disk bandwidth. To improve both access time & bandwidth, by scheduling the disk i/p, o/p requests in a good manner.

When a process needs i/p & o/p to or from the disk, it issues a system call to the O.S. This request specifies several piece of info:

1) Whether this opn is i/p or o/p.
2) Disk address for data transfer
3) Mem. " for data "
4) No. of bytes to be transferred.

So, various disk scheduling algo. are defined for accessing & transferring the data.

① FCFS Scheduling :-

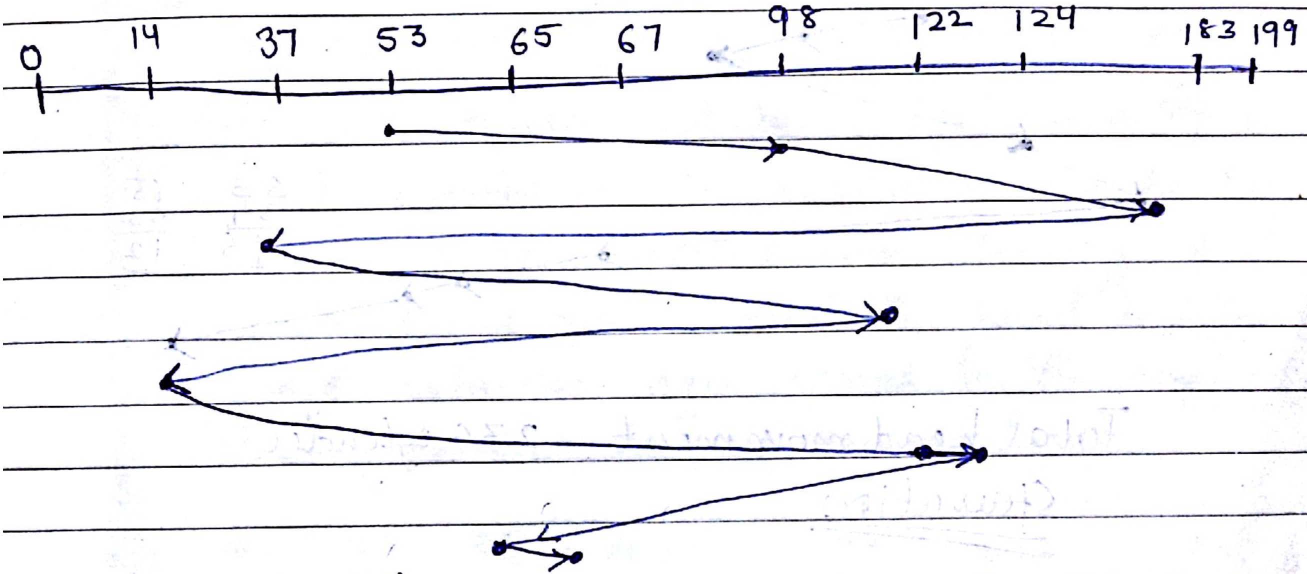
Ques Considering an ordered disk queue with request involving tracks 98, 183, 37, 122, 14, 124, 65, & 67. If the read/write head is initially at track 53. What is the total distance that the disk arm moves to satisfy all pending requests for FCFS?

→ accesstime . 3< D D

Seek time :- is the time for the disk arm to move the heads to the cylinder containg the desired sector.

Rotational latency :- The add^n time waiting for the disk to rotate the desired sector to the disk head.

Disk bandwidth :- Total no of bytes transferred divided by total time b/w the first request for service & completion of the last transfer.



Total (head) arm movement = 640 cylinder

Low performance in FCFS.

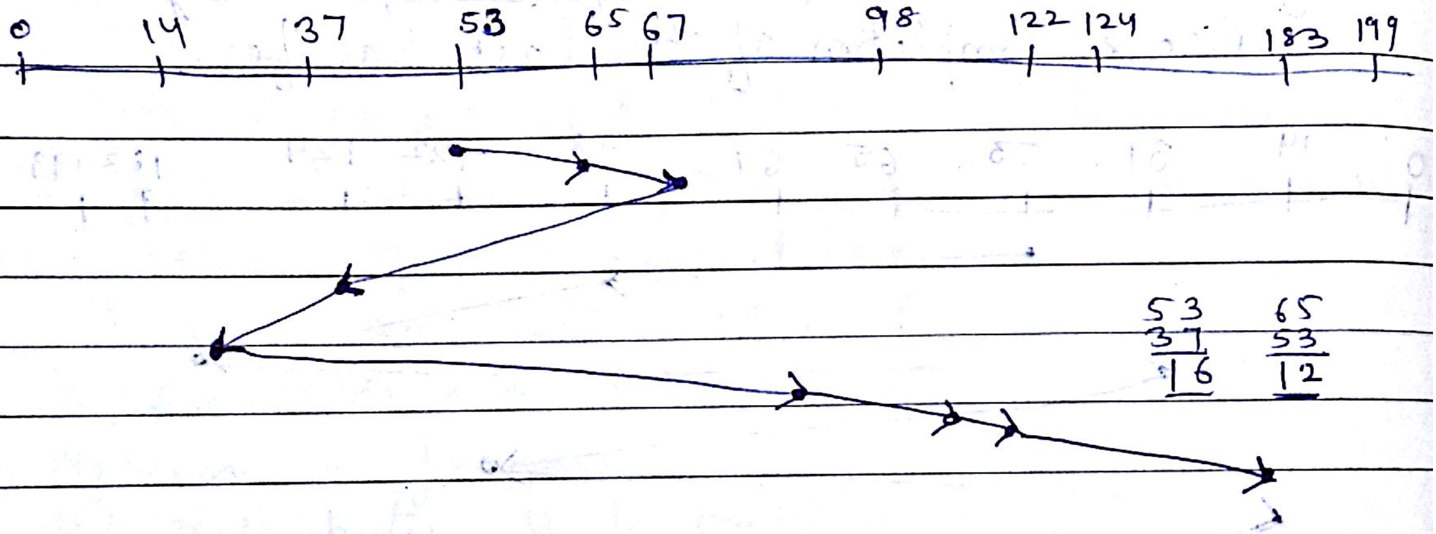
98 − 53 = 95
183 − 98 = 85
183 − 37 = 146
122 − 37 = 85
122 − 14 = 108
14 − 124 = 110
124 − 65 = 59
65 − 67 = 2

640

SSTF:- Shortest seek time first

It selects the request with the min^m seek time from the current head position. SSTF selects the pending request closest to the current head position.

queue = 98, 183, 37, 122, 14, 124, $\boxed{65}$ 67



Total head movement = 236 cylinder

Starvation

② Sector

header Trailer

Data area.

↓

infor like

sector no.

ECC

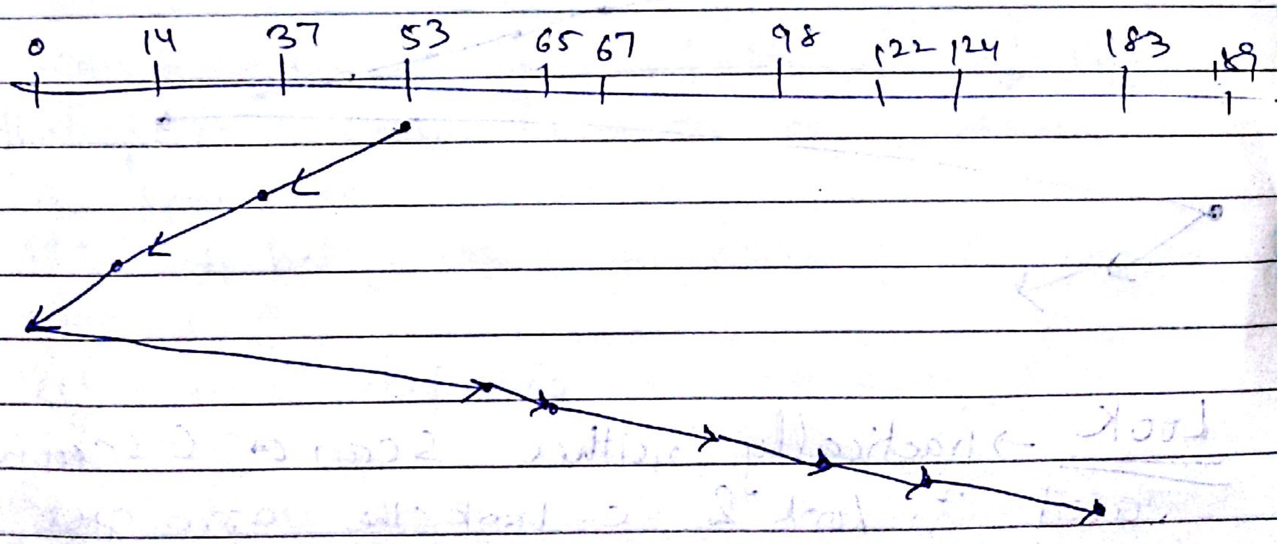
Elevator Algo
or

③ Scan Scheduling :-

The disk arm starts at one end of the disk & moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the dirⁿ of head movement is reversed & servicing continues. The head continuously scans back & forth across the disk.

But from where it will start.-

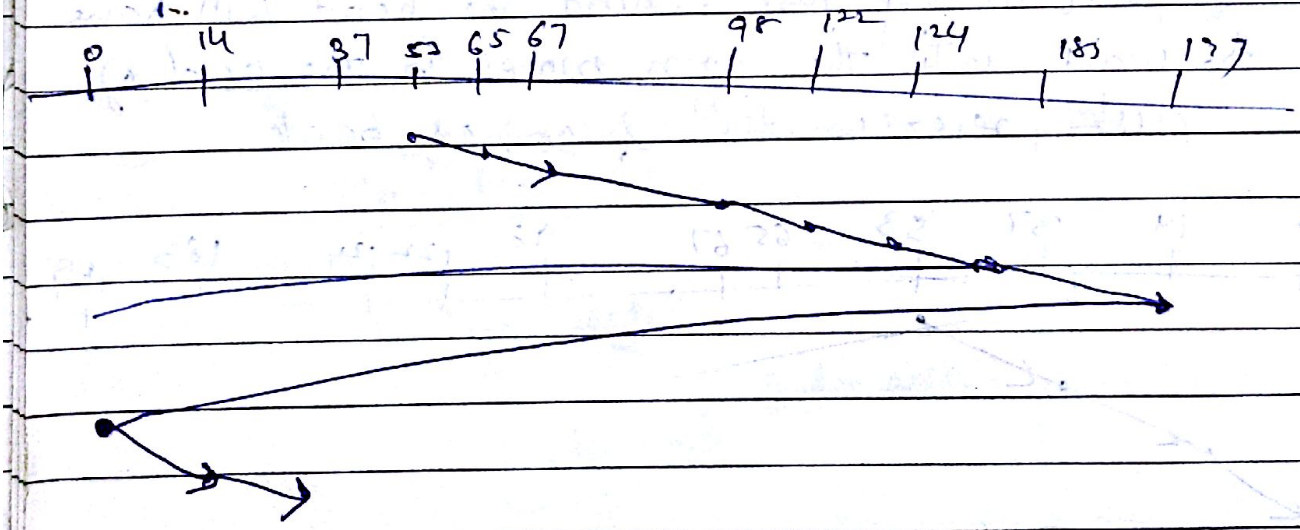
If the disk arm is moving towards 0, that side requests will be entertained. If a request arrives in the queue just in front of head, it will be serviced immediately, a request arriving just behind the head will have to wait until the arm moves to the end of disk, reverses dirⁿ & comes back.



C-scan

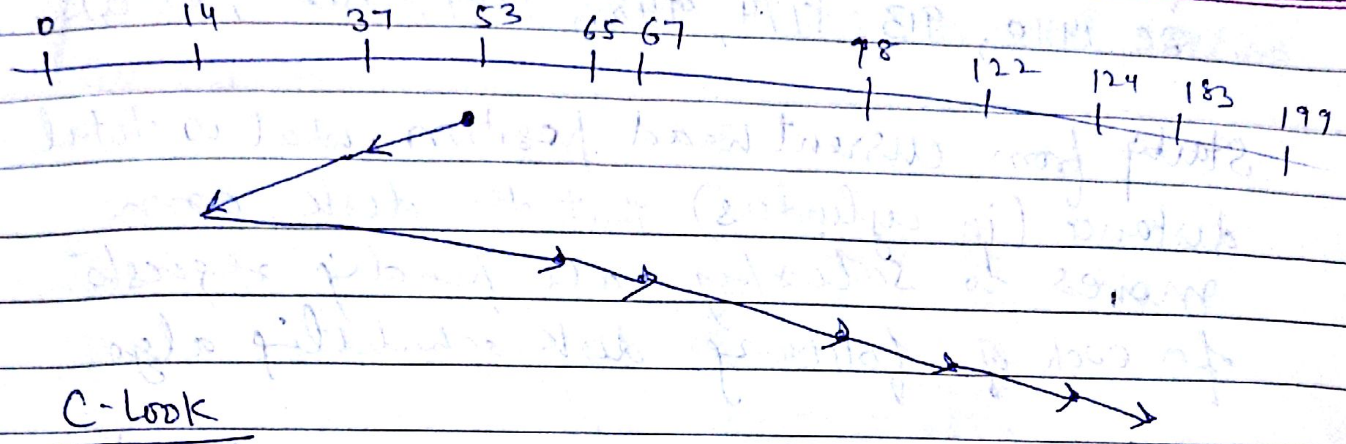
In above algo, we went towards 0, but only few requests are serviced while more requests are waitig towards reverse dir^n as the rev. dir^n has heaviest density of requests. So, we move towards that dir^n. This is the idea of C-scan.

And It moves the head from one end of disk to the other, servicing requests along the way. When the head reaches at at one end, it immediately returns back towards other end without servicing any request on rev. trip.

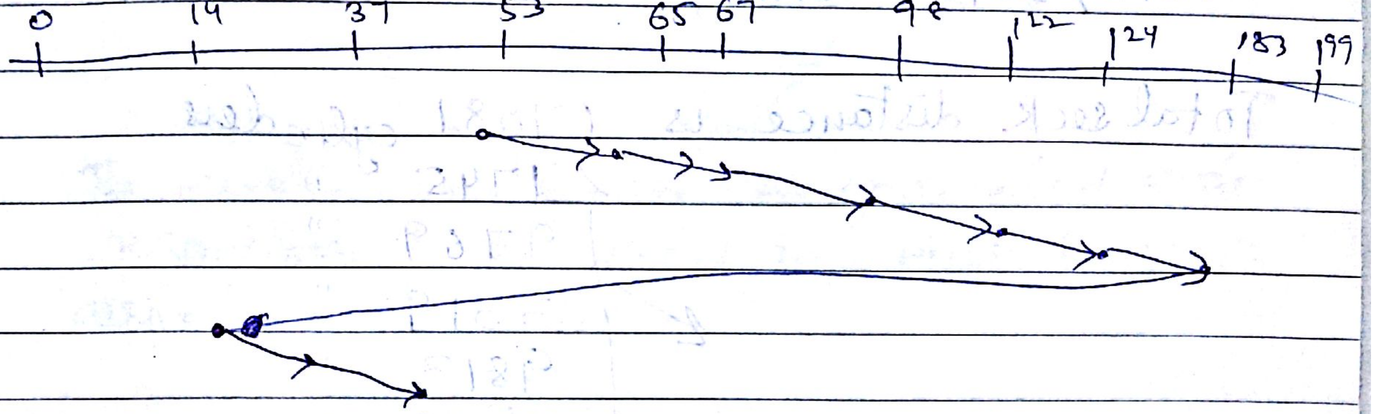


Look → Practically neither scan or C scan is used. In look & c-look the arm goes only as far as final req. in each dir^n.

Look



C-Look



Selection of Disk-scheduling Algo

1. Scan & C-scan perform better for systems that place a heavy load on disk, because no starvation.
2. SSTF is better as it increases performance over FCFS.
3. Performance depends on - no. & type of req.

Ques. Suppose that a disk drive has 5000 cylinders numbered 0 to 4999. The drive is currently serving requests at cylinder 143, & previous req was at cylinder 125. The queue of pending requests in FIFO order is

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

Starting from current head position, what is total distance (in cylinders) that the disk arm moves to satisfy all pending requests for each of following disk scheduling algo.

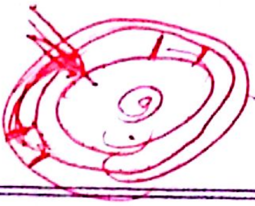
FCFS, SSTF, SCAN, LOOK, CScan, CLook.

Total seek distance is
7081 cylinders
1745 "
9769 "
3319 "
9813 "
3363 "

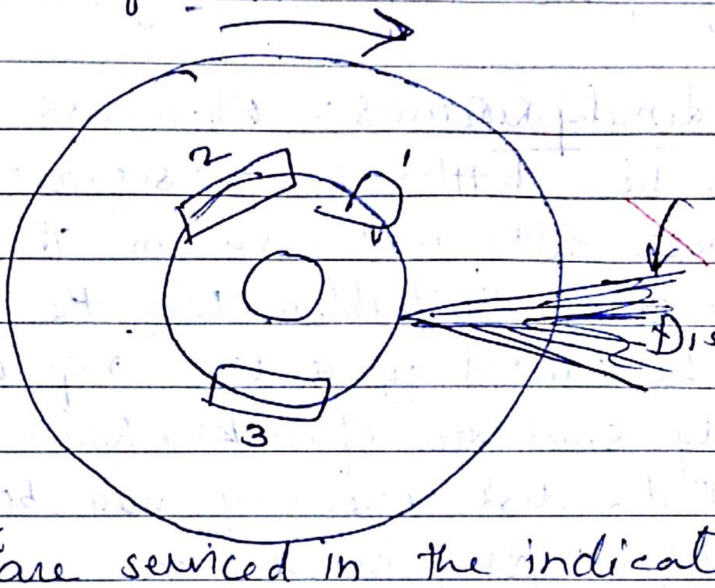
Deitel & Deitel Book (Pg 542-549)

Rotational optimization :- Dominant component of access time was seek time, so optimization of seek time is reqd.
So, SLTF scheduling & SPTF scheduling combine seek & rotational optimization techniques to achieve max^m performance

SLTF (shortest latency time first) :- Once the disk arm arrives at a particular cylinder, there might be many requests pending on various tracks of that cylinder. SLTF examines all these request & services the one with the shortest rotational delay first. It is referred to as sector queuing; request are queued by sector



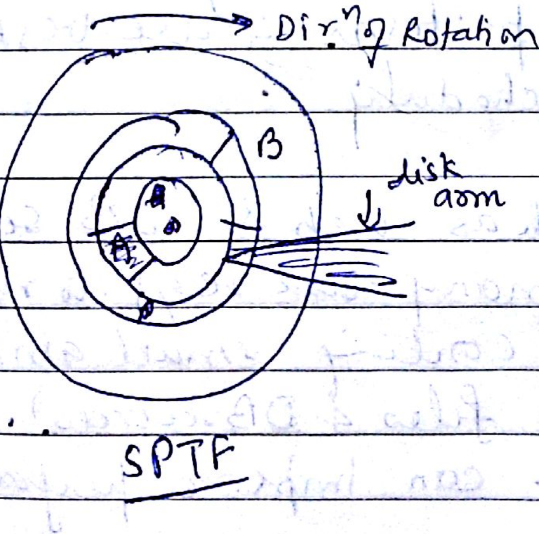
position around the disk & the nearest sectors are serviced first.



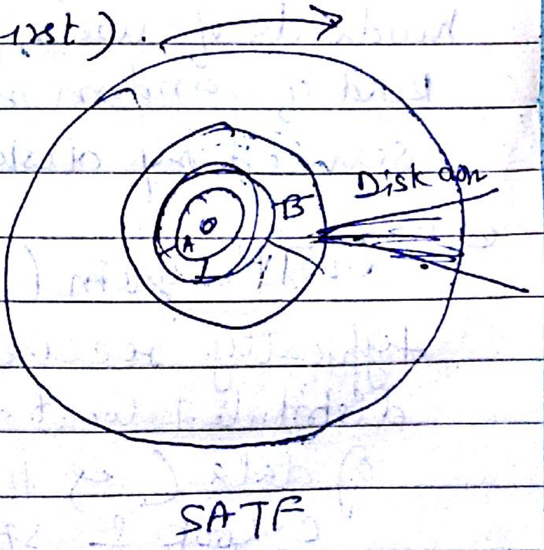
The request are serviced in the indicated order regardless of the order in which they arrived.

SPTF & SATF Scheduling:- The shortest positioning time first strategy next services the request that req. shortest positioning time (seek time + Rot. Latency time). It results in high throughput & low mean response time.

SATF (shortest access time first).



SPTF



SATF

System Consideration :- When is disk scheduling useful? When might it degrade performance?

① Storage as limiting Resources :- When disk storage needs to be bottleneck, designers recommend adding more disks to the system. This will not always solve the problem bcoz the bottleneck could be caused by a large req. load on a relatively small no. of disks. When this situation is detected, disk scheduling may be used as means of improving performance.

② System Load :- Disk scheduling might not be useful in a batch processing system with relatively low degree of multiprogramming. Scheduling is effective as the randomness of multiprogramming increases which increase the system load & leads to erratic req. patterns.

eg file servers in LAN can receive req. from hundreds of users, which normally results in kind of random req. patterns & are best serviced by disk scheduling.

eg2. OLTP system (such as web & DB servers typically receive many disk reqs to randomly distributed locations containing small amount of data (eg HTML files & DB records). Here, C-LOOK & SATF can improve performance