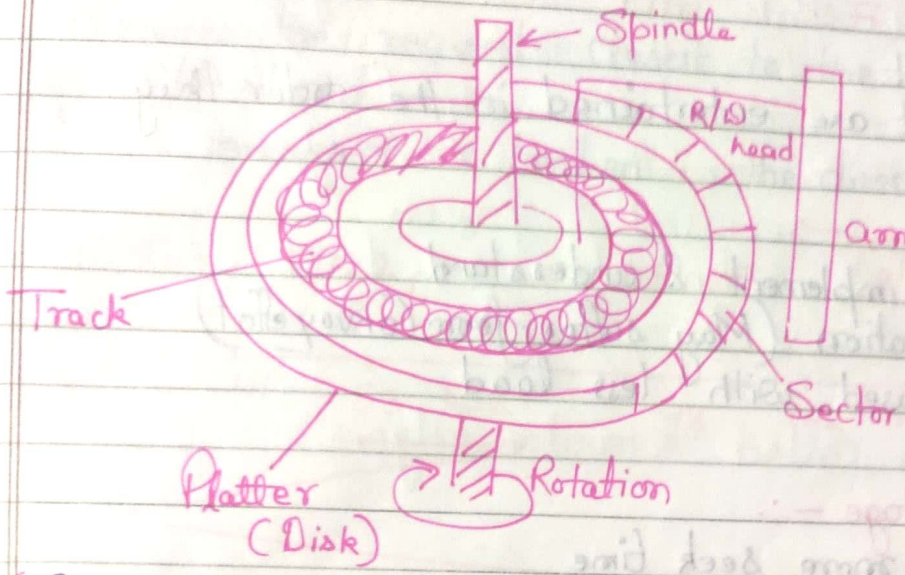
Disk Scheduling

APPU DATE _____ PAGE _____



- Each platter (disk) has a circular flat shape like a CD or DVD.
- Generally diameter ranges from 1.8 to 5.25 inches
- Both surfaces are covered by magnetic ~~diameter~~ material
- R/W head attached to disk arm
- Surface of disk is divided into circular tracks
- Track divided into sectors

Seek Time —: Time required to move the R/W head on the desired track

Disk Scheduling —: In case of multiple I/O request disk scheduling algo. must decide which request must be executed first.

1. FCFS —:

✓ Simplest
✓ Request are entertained in the order they arrive

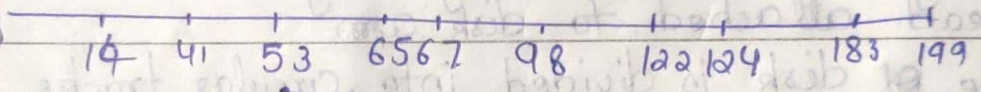
Advantages —:

1. Easy to implement & understand
2. No starvation (May suffer from Convoy effect)
3. Can be used with less load

Disadvantage —:

1. Require more seek time
2. More waiting time & response time
3. Inefficient

Q) 98, 183, 41, 122, 14, 124, 65, 67



R/D head at 53

98 - 53 = 45.
183 - 98 = 85
183 - 41 = 142
122 - 41 = 81
122 - 14 = 108
124 - 14 = 110
124 - 65 = 59
67 - 65 = 2

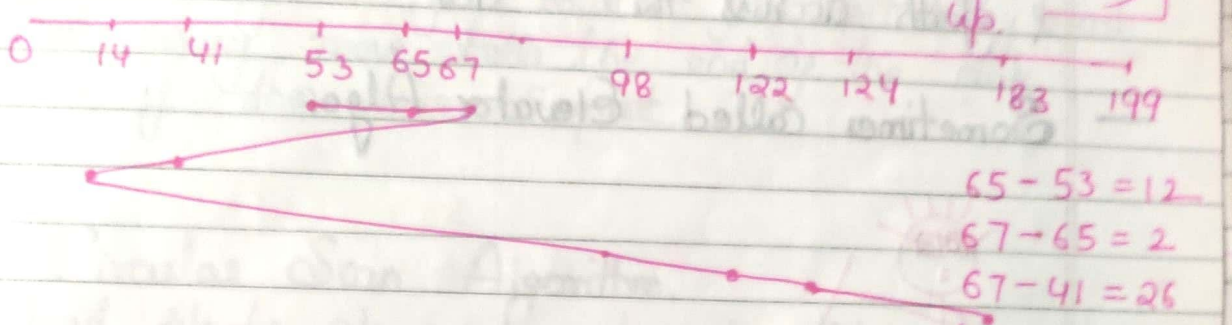
Total = 632 Moves we need todo

(2) Shortest Seek Time First (SSTF).

Services the request closest to the current position of R/W head.

Tie ~~breaks~~ is broken in the direction of R/W head.

(1) 98, 183, 41, 122, 14, 124, 65, 67 (R/W Head at 53) ↑ up.



65 - 53 = 12
67 - 65 = 2
67 - 41 = 26
41 - 14 = 27
98 - 14 = 84
122 - 98 = 24
124 - 122 = 2
183 - 124 = 59

Total = 236 header moves

Advantages -:

- Efficient in seek moves
- Less average response & waiting time
- Increases throughput

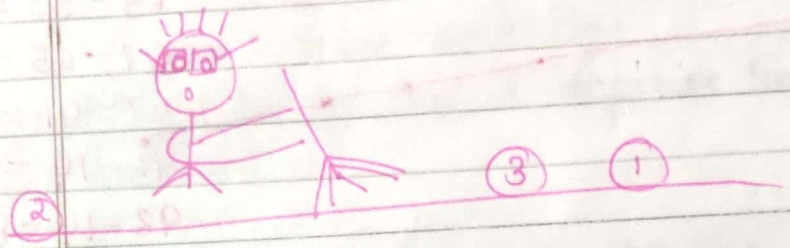
Disadvantages -:

- Overhead to find out closest request
- Request which are far from head will starve
- High variance & respons time & waiting time.

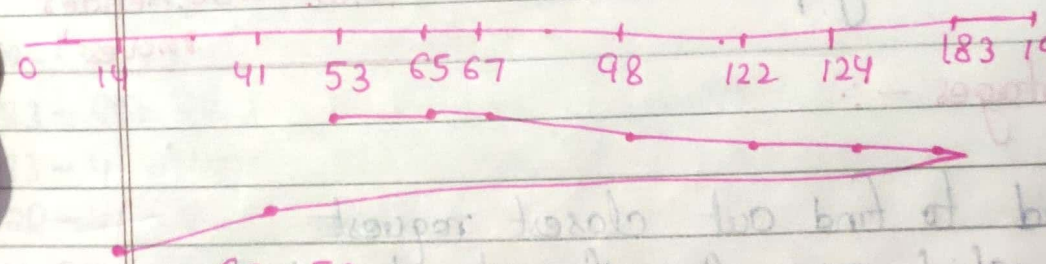
(3) SCAN Disk Scheduling Algorithm

SCAN -:. Head starts from one end of the disk towards the other end, servicing requests in between & reach the other end. & then the direction of the head is reversed & process continues & continuously scans back & forth across the disk

— Sometimes called Elevator Algo.



Q) 98, 183, 41, 122, 14, 124, 65, 67 (Head at 53) ↑



65 − 53
67 − 65
98 − 67
122 − 98
→ 199 − 183
→ 199 − 41
41 − 14.

= 331.

APPU DATE _____
PAGE _____

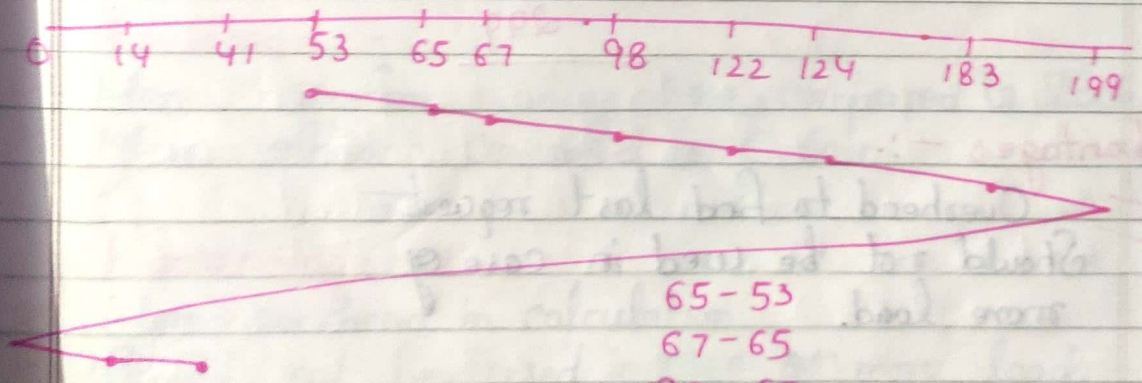
Advantages -:
- Simple & easy to implement & understand
- No starvation (Bounded Waiting)
- Low variance in response time & waiting time

Disadvantages -:
- Long waiting time for location just visited by head
- Unnecessary move tells the end of the disk even if there is no request.

Circular Scan Algorithm.

Head starts at one end of the disk & moves toward the other end, servicing request in between, & reach other end & then direction of head is reversed & head reaches first end without satisfying any req.

Q) 98, 183, 41, 122, 14, 124, 65, 67 Head = 53 ↑



65 - 53
67 - 65
98 - 67
122 - 98
124 - 122
183 - 124
199 - 183
199 - 0 →
14 - 0 →
41 - 14
386

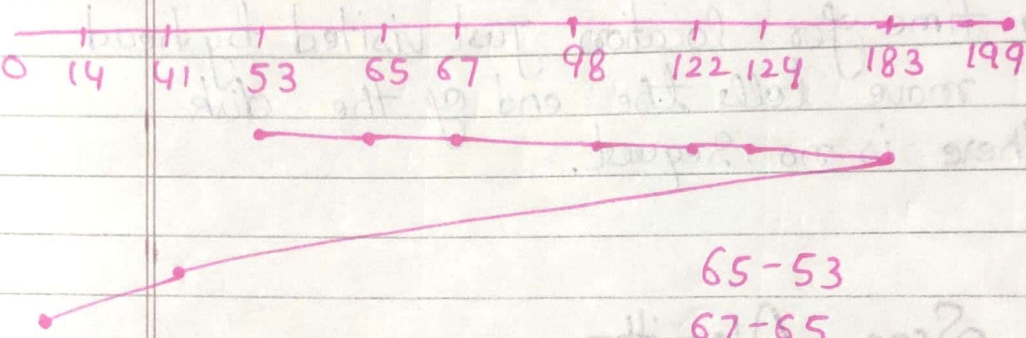
Advantage -:
- Provide uniform waiting time
- Better response time

Disadvantage -:
- More seeks movements. Compared to scan.

5

Look.

It is same as Scan algo but instead of going till the last track we go till last request & then change direction



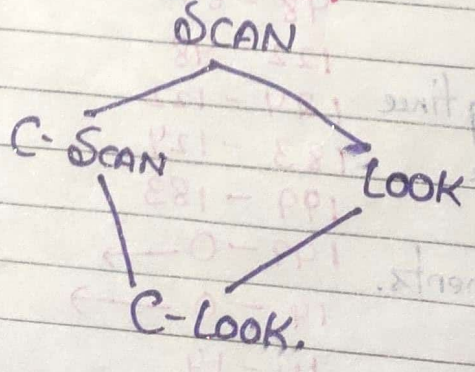
65-53
67-65
98-67
122-98
124-122
183-124
183-41
41-14
299

Advantage -:

- Better performance
- Should be used in case of less load

Disadvantages -:

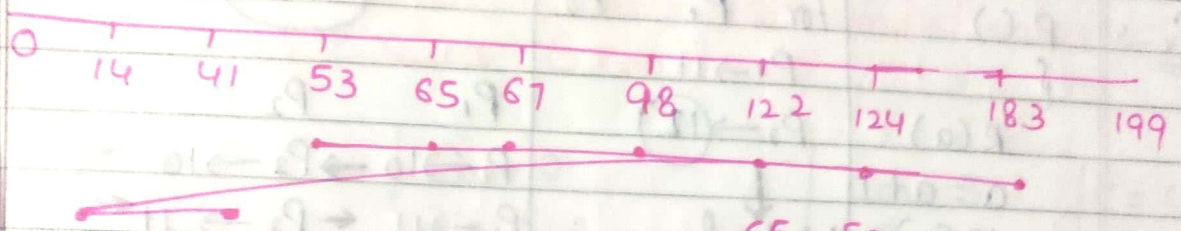
- Overhead to find last request
- Should not be used in case of more load.



APPU DATE _____
PAGE _____

→ C-Look Algo.

It take both the CSCAN & Look algo
Satisfy req. in only one direction
Will go till last request & return ~~last~~ but
not till last track.



65 - 53
67 - 65
98 - 67
122 - 98
124 - 122
183 - 124
41 - 14
326.

Advantages - :

- More uniform waiting time compared to Look.
- More efficient Compaired to C-Scan

Disadvantage - :

- More overhead in calculation
- Should not be used in case of more load.

APPU DATE ____
PAGE ____

Process Synchronization :-

For multiprogramming system, processes share the resources. But a resource is not sharable all the time. (Printer)

Ex :-

```
P()
{
  R(a)
  a = a+1
  W(a)
}
```

for P_1 P_2

$P_1 \rightarrow 11$

$P_2 \rightarrow (12)$

↓

Mutual exclusive

$a = 10$

P_1 P_2

$P_1 \rightarrow 10 \rightarrow P_2 \rightarrow 10$

$P_1 \rightarrow 11 \leftarrow P_2 \rightarrow 11$

Race Condition

* **Race Condition :-**
We run the process for both in mutual exclusive manner & context switched manner & the final answer is changing. from 12 & 11 (inconsistency).

* **Critical Section :-**
That part of program or process where the h/w or s/w try to access the shared resource is called Critical Section.

```
P()
{
  [ Critical Section ]
}
```

When there is Management there is Starvation...

APPU DATE _____ PAGE _____

Q. P1() { 1 C = B - 1; 2 B = 2 * C; }

P2() { 3 D = 2 * B; 4 B = D - 1; }

B is a shared variable with initial value 2? How many values B can have?

Case I	Case 2	Case 3.	C = 4
1, 2, 3, 4.	3, 4, 1, 2.	1, 3, 4, 2.	3, 1, 2, 4
C = 2 - 1 = 1.	D = 2 * 2 = 4	C = 2 - 1 = 1	D = 4
B = 2 * 1 = 2.	B = 4 - 1 = 3.	D = 2 * 2 = 4.	C = 1
D = 2 * 2 = 4	C = 3 - 1 = 2.	B = 4 - 1 = 3	B = 2
B = 4 - 1 = 3	B = 2 * 2 = 4	B = 2 * 1 = 2	B = 3

Race Condition

C = 5	C = 6.
1, 3, 2, 4.	3, 1, 4, 2
C = 1	D = 4
D = 4	C = 1
B = 2	B = 3
B = 3	B = 2

(3) Ans

* Critical Section Problem —:

Not necessary to have a round robin in every process.

P1 P2 P3

(1) Mutual Exclusion → (Mandatory).

(2) Progress = Only process should compete who actually want to.

(3) Bounded Wait = A time limit for which a process have to wait.

↳ (optional)

Q)

```
P0
{
While (1)
{
While (turn != 0);   → boolean value
critical section
turn = 1 ;
remainder section
}.
```

```
P1
While (1)
{
While (turn != 1);
critical section
turn = 0;
remainder section
}.
```

① Mutual Exclusion holds true
② Progress holds false
③ Baund Wait false

Not Optimal

```
P0
{
entry section
C.S.
exit section
rem. section
}
```

Solution —:

```
P0
While (1)
{
flag [0] = T        P0
while (flag [1]);
C.S.
flag [0] = f
}
```

```
P1
While (1)
{ While (1)
flag [1] = T        P1
while (flag [0]);
C.S.
flag [1] = f
}.
```

Flag: 0 | 1 : F | F

① M.A. ✓
② Progress ✗

* flag: T | T

Deadlock

Peterson's Method :-

```
P0
while (1)
{
flag[0] = T
 turn = 1
while (turn == 1 && flag[1] == T);
 C.S.
flag[0] = f
}.
```

```
P1
While (1)
{
 flag[1] = T
 turn = 0
while (turn == 0 && flag[0] == T);
C.S.
Flag[1] = f
}.
```

Turn = 0/1 , 0 , 1 P0 = Turn = 1

Flag | T | f | Flag = | T | F |

ME (True).
Progress ✓
BW ✓

When both processes are interested at the same time there is no deadlock & one of the processes is executed.

Semaphores

A semaphore is an integer variable that apart from initialization, is accessed only through 2 standard atomic operations

→ Wait() ↑ Reduce
→ Signal(). ↑ Increase

int S=1

Pi
do
{
entry section
// C.S.
exit section
// Rem. section
}
while (T)

(1) wait (S)
{
while (S<=0);
S=S-1
}

(2) Signal (S)
{
S=S+1
}

Signal (S)

① ME ✓
② Progress ✓
③ B.W. ×

Pi
do {
wait (S);
// C.S.
Signal (S);
// Rem. Section
}
while(T)

5.13 5.14

for deciding order of Execution

$P_2 \longrightarrow P_1 \longrightarrow P_3$

(B) (A) (C)

P_1	P_2	P_3
Wait (S_1)	Signal (S_1)	Wait S_2
Signal (S_1)	}	

$S_1 = 0$, $S_2 = 0$

Resource Management

```
Pi
{
Wait (S)
C.S.
Signal (S)
R.S.
}
```

S = 5

5 processes can be in C.S.