2019

1. What is an operating system? Discuss the evolution of operating system.

- An operating system (OS) is a software component that acts as an intermediary between computer hardware & the computer user. Its primary functions include managing hardware resources, providing services to applications, & facilitating communication between the hardware & software components of a computer system. The evolution of OS can be traced through several key stages:

1) No operating system (1940s)

- Early computers had no operating systems. Users interacted directly with the hardware, writing machine-level programs.
- Programs were loaded onto the computer using punched cards or other manual methods.

2) Batch Processing systems (1950s - 1960s)

- Batch processing system introduced the concept of running multiple jobs in sequence without user interaction.
- OS, like IBM's OS/360, managed the execution of batches of jobs, improving efficiency & resource utilization.

3) Time-sharing systems (1960s-1970s)

- Time sharing system allowed multiple users to interact with the computers simultaneously.
- OSs, like Multics & later Unix, provided each user with a virtual machine, giving the illusion of exclusive access to a system.

4) Personal computer (1980s)

- The advent of personal computers saw the rise of OS like

MS-DOS, Apple DOS, & early versions of Windows.

- Graphical User Interfaces (GUIs) became popular, making computers more accesible to non-technical users.

5) Networked Systems (1980s-1990s)

- The growth of networking led to the development of networked OS.
- Novell Netware & Microsoft Windows NT provided features for managing network resources.

6) Client-Server Architecture (1990s)

- Client-server computing became prevalent, with OS like Windows 95/98/ME, Windows NT, & Novell Netware.
- The Internet became widely accesible, leading to the development of web-based applications.

7) Mobile operating system (2000s-Present)

- The rise of smartphones brought mobile OS like iOS, Android, and Windows Mobile.
- These OSs are optimized for touch interfaces & provide app-based ecosystems.

8) Modern operating systems (2000-Present)

- contemporary operating systems include Microsoft Windows, macOS, Linux distributions, & Unix variants.
- Virtualization, & containerization technologies, such as VMware & Docker, have gained popularity.

9) Real-Time OS (RTOS)

- RTOS, like Free RTOS & VxWorks, are designed for applications with strict timing requirements, such as embedded systems & robotics.

10) Open source operating system

- Linux, an open-source Unix-like OS, gain widespread adopting in server environments & as the basis for Android.

The evolution in OS reflects advancements in hardware capabilities, changes in user needs, and innovations in software design.

2) What is critical section? Why mutual exclusion is required? How mutual exclusion can be achieved? Describe any one of them.

- Critical Section is the part of a program where shared variables and resources are accessed by multiple process. Mutual exclusion is required because

1) To control access to shared resources, maintaining program logic

2) To avoid deadlock by controlling access to resources

3) To ensures proper synchronization with condition variables

4) To prevent data inconsistency i.e. ensure shared data is accessed by only one process or thread at a time to avoid conflict.

5) To eliminate race condition by allowing only one thread to execute a critical section.

6) To enforces a consistent order of execution in a multi-thread environment.

Mutual Exclusion can be achieved by
1) Test & set lock
2) Strict Alternation method
3) Peterson's Method
4) Semaphore

Peterson's Method.
- It is a software mechanism implemented at user mode. It is busy waiting solution & can be used for 2 process.

```
#define N2
#define True 1
#define false 0
int interest[N]=false;
int True ;
void entrysection(int process)
    { int other ;
      other = 1-process ;
      interest[process] = True;
      turn = process;
      while (interest (other ] == True && turn == process );
    }
void exit_section(int process)
    {interest [process] = false;
    }
```

3) Define process & its different states. What are the various operation on semaphore. How message passing is useful on IPC?

- A process is an instance of a program in execution. A program by itself is not a process, a program is a passive entity such as file containing a list of instruction stored on disks. A process is an active entity with a program counter specifying the next instruction to execute & a set of associated resources.

The different states are:-

i) New
ii) Running
iii) Waiting
iv) Ready
v) Terminated

The various operation on semaphore are:-

a) Initialization- A semaphore is created & initialized with an initial value. Initial value represents the no. of units available for allocation.

b) Wait (P/Down) operation

- Decrementing the value of semaphore by one. If the resulting value is non-negative, the thread or process continues execution. If result value is negative, process is blocked. until next semaphore value become non-negative.

c) Signal (V/Up) operation

- Increment value of semaphore by 1. If there are any processes or threads waiting due to 'wait' operation, one of them is allowed to continue execution.

d) Blocking wait
- If a wait operation would result negative, semaphore value, process is blocked until value is non-negative.

e) Non-blocking wait (try wait)
Similar to wait but instead blocking, process continues execution even if semaphore value is negative.

f) Timed wait
- Similar to wait operation, but with additional time limit.

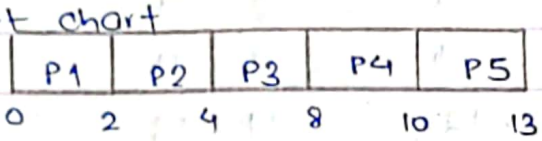
Message passing is useful for IPC because
a) Process can communicate through message without exposing internal details.
b) Enables processes to coordinate activities by exchanging messages.
c) Provides fault isolation, as the failure of one process does necessarily impact others.

Compute average waiting time & average turn around time using FCFS, SJF, Priority (lowest no. represents highest priority) & Round-Robin (quantum=30)s) scheduling algorithm for the following set of processes. Assume that all processes have arrived at time 0 in the order P1, P2, P3, P4 & P5

Process	BT(ms)	Priority
P1	2	1
P2	2	1
P3	4	3
P4	2	4
P5	3	2

Arrival time = 0

1) FCFS Gantt chart



waiting time for P1 = 0

waiting time for P2 = 2

P3 = 4

P4 = 8

P5 = 10

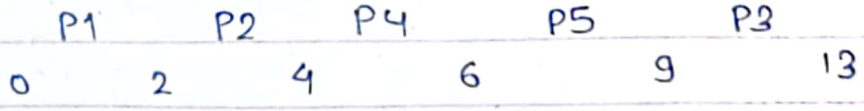
A.W.T = (0+2+4+8+10) /5

= 4.8

A.T.T = (2+4+8+10+13) /5

= 7.40 //

2) SJF , Gantt chart



$A.W.T = \frac{0+2+4+6+9}{5} = 4.2 ms$

$A.T.T = \frac{2+4+6+9+13}{5} = 6.8 ms$ //

3) Priority → Gantt chart

P1	P2	P5	P3	P4
0	2	4	7	11	13

Process	BT	AT	C.T	TAT = CT - AT	WT = TAT - BT
P1	2	0	2	2	0
P2	2	0	4	4	2
P3	4	0	11	11	7
P4	2	0	13	13	11
P5	3	0	7	7	4

A.W.T = (0+2+7+11+4) /5 = 4.80 ms

A.T.T = (2+4+11+13+7) /5 = 7.40 ms

4) Round - Robin (Q.T) = 3ms

Gantt chart is

P1	P2	P3	P4	P5	P3
0	2	4	6	8	10	12

A.W.T = (0+2+(4-0)+(10-6)+6+8)/5

= 4.8ms

A.T.T = (2+4+12+8+10)/5

= 7.2 ms

5. Define the term Deadlock. What are the necessary conditions for a deadlock? Explain the deadlock avoidance algorithm.

→ A deadlock is a situation where a set of processes are blocked because each processes is holding some resources & waiting for another resources acquired by other processes.
The necessary conditions for a deadlock are:-

1) Mutual exclusion
- Two or more resources are non-shareable (only one process can use at a time)

2) Hold & wait
- A process must be holding at least one resource & waiting for at least one resource that is currently being held by some other processes.

3) No preemptive
- Resources cannot be preempted i.e. Once a process is holding a resources then that resources cannot be taken from a process unless the process releases the resources.

4) Circular wait - A set {P0, P1, P2, ... Pn} of waiting processes must exist.

such that
→ P0 is waiting for a resources held by P1.
→ P1 is waiting for a resources held by P2.
→ P2 is waiting for a resources held by P3.
→ Pn-1 is waiting for a resources held by Pn.
→ Pn is waiting for a resources held by P0.

Deadlock avoidance

Deadlock avoidance employs the most famous deadlock avoidance algorithm that is Banker's algorithm.
Banker's Algorithm is a resource allocation & deadlock avoidance algorithm develop by Dijkstra that test for the safety by simulating the allocation of predetermine maximum possible amount of resources. So, this algorithm can be used in banking system to ensure that the bank never allocates all its available cash such that it can no longer satisfy the need of entire customer. This algorithm is applicable to a system with multiple instance of resources of each type. When a new process enter into a system it must declare maximum no. of instances of each resource type that it may need. This algorithm prevents deadlock by denying the request if it determine that accepting a request put the system in unsafe state.

Deadlock avoidance algorithm (Banker's algorithm) contain safety algorithm and resource request allocation which removes the deadlock or avoid deadlock in the system.

6. Consider the following snapshot of a system.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Answer the following questions with Banker's algorithm:-

a) What is the content of the matrix need?
b) Is the system in a safe state?
c) If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

→ Soln:-

We know,

i) The contents of need matrix is

Need [i,j] = Max [i,j] - Allocation [i,j]

Need matrix

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

ii) Applying the safety algorithm

For Pi,
if Need < Available
then Pi is in safe state.

For i=0, P0,
Need0 = (0,0,0,0)
Available = (1,5,2,0)

Need0 ≤ Available i.e. (0,0,0,0) ≤ (1,5,2,0) (True)

So, P0 will be in safe sequence.

Now, Available = Available + Allocation
= (1,5,2,0) + (0,0,1,2)
= (1,5,3,2)

For i=1, P1,
Need1 = (0,7,5,0)
Available = (1,5,3,2)

Need1 ≤ Available i.e. (0,7,5,0) ≤ (1,5,3,2) (false)

So, P1 must be waiting.

For i=2, P2,

Need 2 = (1,0,0,2)

Available = (1,5,3,2)

Need 2 ≤ Available i.e. (1,0,0,2) ≤ (1,5,3,2) (true)

So, P2 will be in safe sequence.

Now, Available = Available + Allocation

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

= (2,8,8,6)

For i=3, P3,

Need 3 = (0,0,2,0)

Available = (2,8,8,6)

Need 3 ≤ Available i.e. (0,0,2,0) ≤ (2,8,8,6) (True)

So, P3 will be in safe sequence.

Now, Available = (2,8,8,6) + (0,6,3,2)

= (2,14,11,8)

For i=4, P4,

Need 4 = (0,6,4,2)

Available = (2,14,11,8)

Need 4 ≤ Available i.e. (0,6,4,2) ≤ (2,14,11,8) (True)

So, P4 will be in safe sequence.

Available = (2,14,11,8) + (0,0,1,4)
= (2,14,12,12)

For i = 1, P1
Need 1 = (0,7,5,0)
Available = (2,14,12,12)

Need 2 ≤ Available i.e. (0,7,5,0) ≤ (2,14,12,12) (True)

So, P2 is in safe sequence.

Available = (2,14,12,12) + (1,0,0,0)
= (3,14,14,12)

Hence, the safe sequence is

<P0,P2,P3,P4,P1> //

iii) Since the process P1 request some additional resource request (0,4,2,0).
To decides whether this request can granted or not.

we first check,
Requesti ≤ Available
i.e. (0,4,2,0) ≤ (1,5,2,0) (True)

So, the request of P1 may be granted.
To confirm that this request is granted we check the new state of system by applying safety algorithm that

the system is safe or not.

To define the state of new system because of arrival of P1, we follow the resource request algorithm as:

	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	0	0				
P1	1	4	2	0	0	3	3	0	1	1	0	0
P2	1	3	5	4	1	0	0	2				
P3	0	6	3	2	0	0	2	0				
P4	0	0	1	4	0	6	4	2				

We must determine whether the state is safe or not.
Applying safety algorithm.

For Pi,
if Need $\leq$ Available
Pi is in safe state.

For i = 0, P0,
Need0 = (0,0,0,0)
Available = (1,1,0,0)
Need0 $\leq$ Available i.e. (0,0,0,0) $\leq$ (1,1,0,0) (True)

So, P0 will be in safe state.

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

For i = 1, P1,

Need 1 = (0, 3, 3, 0)

Available = (1, 1, 1, 2)

Need1 ≤ Available i.e. (0, 3, 3, 0) ≤ (1, 1, 1, 2) (false)

So, P1 must be waiting.

For i = 2, P2,

Need 2 = (1, 0, 0, 2)

Available = (1, 1, 1, 2)

Need 2 ≤ Available i.e. (1, 0, 0, 2) ≤ (1, 1, 1, 2) (True)

So, P2 will be in safe sequence.

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

= (2, 4, 6, 6)

For i = 3, P3,

Need 3 = (0, 0, 2, 0)

Available = (2, 4, 6, 6)

Need 3 ≤ Available i.e. (0, 0, 2, 0) ≤ (2, 4, 6, 6) (True)

So, P3 will be in safe sequence.

Available = (2, 4, 6, 6) + (0, 6, 3, 2)

= (2, 10, 9, 8)

For i = 4, P4,

Need 4 = (0, 6, 4, 2)

Available = (2, 10, 9, 8)

Need 4 ≤ Available i.e. (0, 6, 4, 2) ≤ (2, 10, 9, 8)

So, P4 will be in safe sequence.

Available = (2, 10, 9, 8) + (0, 0, 1, 4)

= (2, 10, 10, 12)

For i = 1, P1

Need 1 = (0, 3, 3, 0)

Available = (2, 10, 10, 12)

Need 1 ≤ Available i.e. (0, 3, 3, 0) ≤ (2, 10, 10, 12) (True)

So, P1 will be in safe sequence.

The safe sequence is

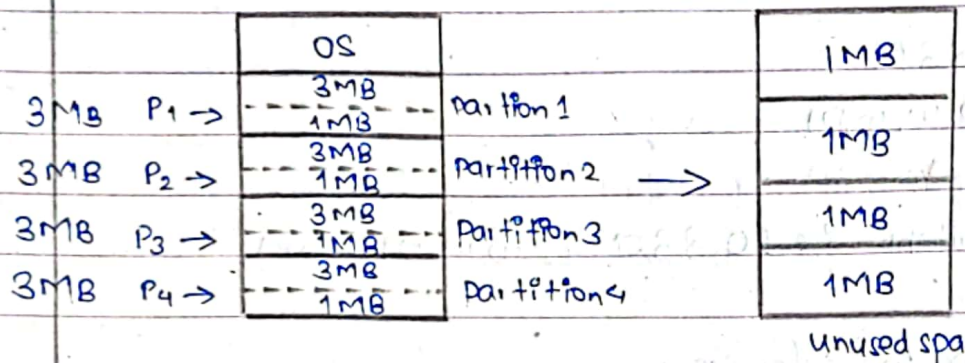
< P0, P2, P3, P4, P1 >

Hence, we can immediately grant the grant request of P1.

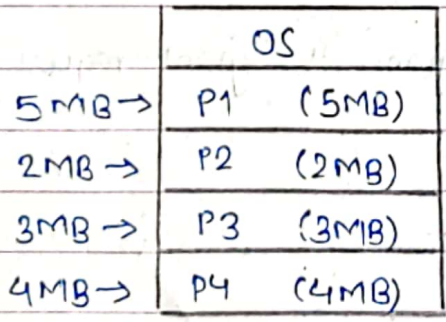
7. Explain the static & dynamic memory allocation techniques with examples. Use FIFO page replacement algorithm in the following reference string having 4 frames & calculate the no. of page faults.

0,1,2,3,0,1,2,3,0,1,2,3,4,5,6,7

- Static memory allocation has main memory divided into several fixed partition. They can be of same or different sizes. Each partition can hold single processes. In fixed position, degree of multiprogramming is fixed position & v. less due to fact that size cannot be changed.



Dynamic memory allocation tries to overcome problem of static partitioning. Hence, partition size is declared initially. Each process occupies only as much memory they require.



Dynamic Partitioning

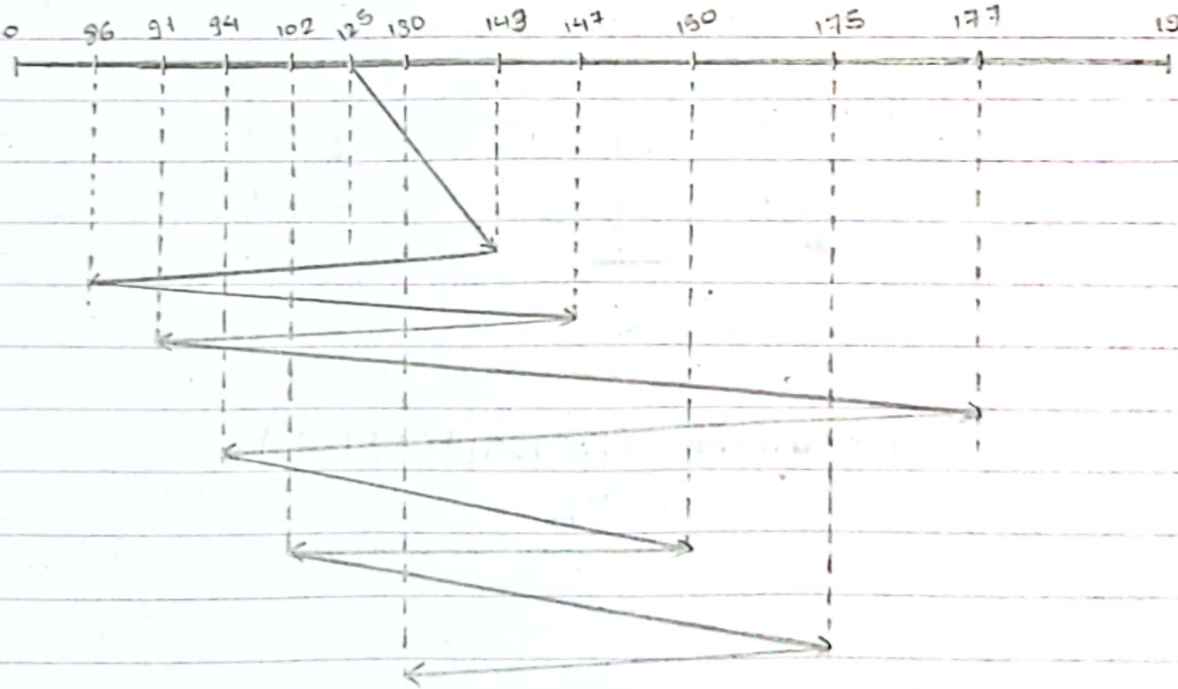
Using FIFO

0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	ø	4	4	4	4
	1	1	1	1	1	1	1	1	1	1	1	1	5	5	5
		2	2	2	2	2	2	2	2	2	2	2	2	6	6
			3	3	3	3	3	3	3	3	3	3	3	8	7
*	*	*	*	H	H	H	H	H	H	H	H	*	*	*	*

Total no. of page faults = 8 //

8. Suppose that the head of moving head disk with 200 tracks numbered 0-199, currently serving request at track 143, & just finished request at track 125. The queue of request is in FIFO order 86, 147, 91, 177, 94, 150 & 102, 175, 130. What is total head movement using FCFS, c-SCAN, SSTF & LOOK.

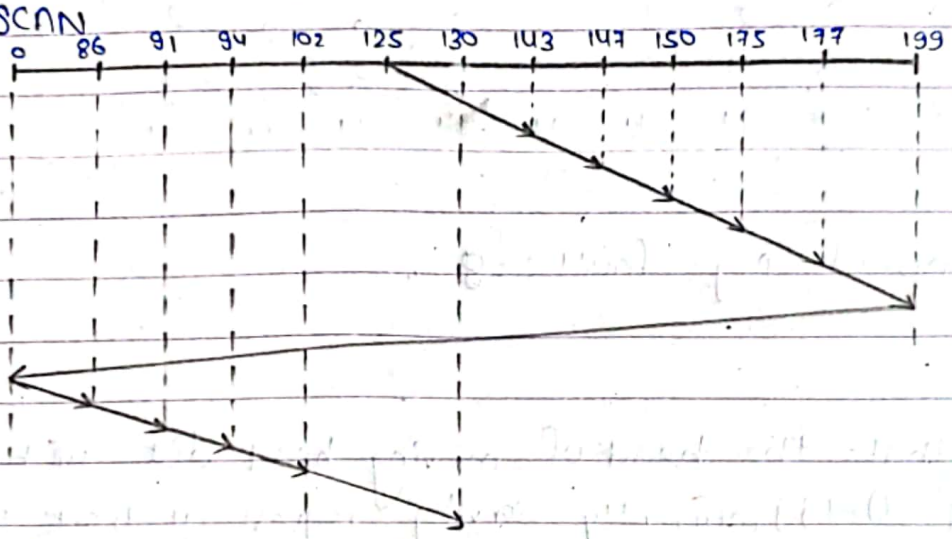
1) FCFS



Total head movement = (143-125) + (143-86) + (147-86) + (147-91) + (177-91) + (177-94) + (150-94) + (150-102) + (175-102) + (175-130)

= 427 //

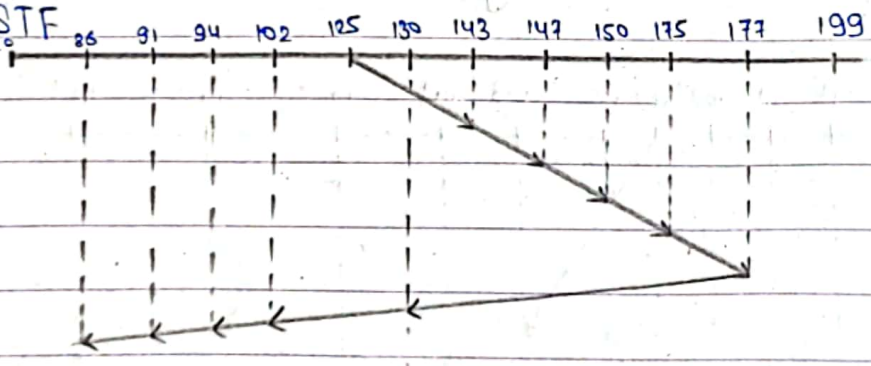
2) C-SCAN



Total head movement = (199-125) + (199-0) + (130-0)

= 403 //

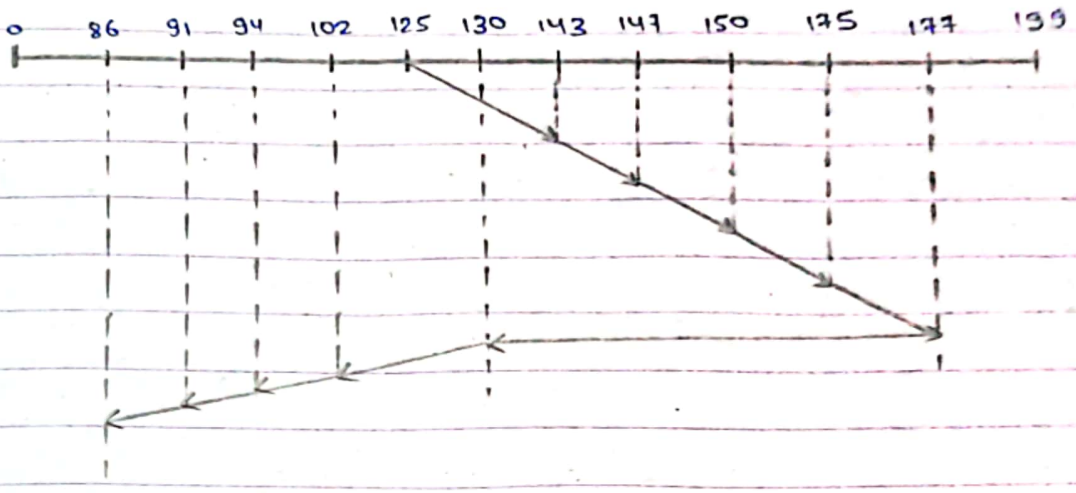
3) SSTF



Total head movement = (177-125) + (177-86)

= 149 //

4) LOOK



Total head movement = (177-125) + (177-86)

= 149