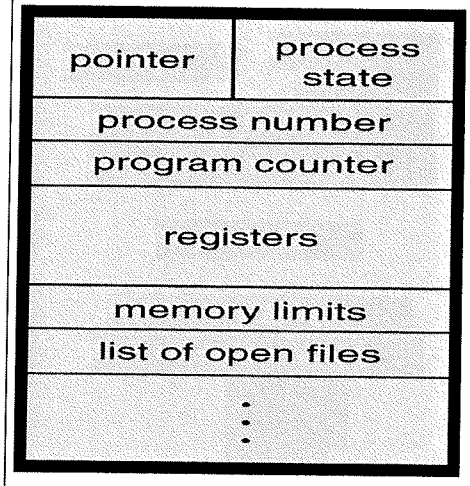
QID - 21092302894

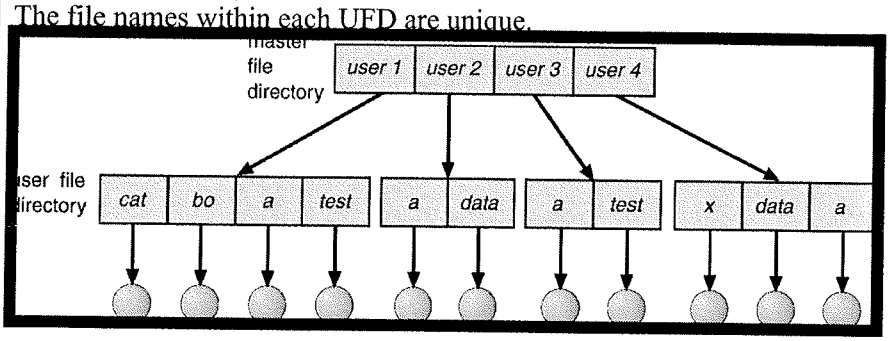
SCHEME OF EVALUATION
(Scoring Indicators)

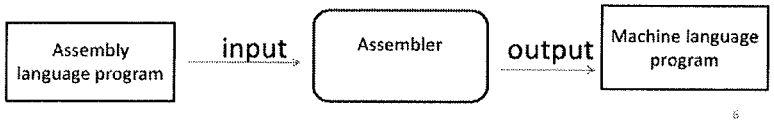
Revision: 21 **Course Code: 5132**
Course Title: OPERATING SYSTEM

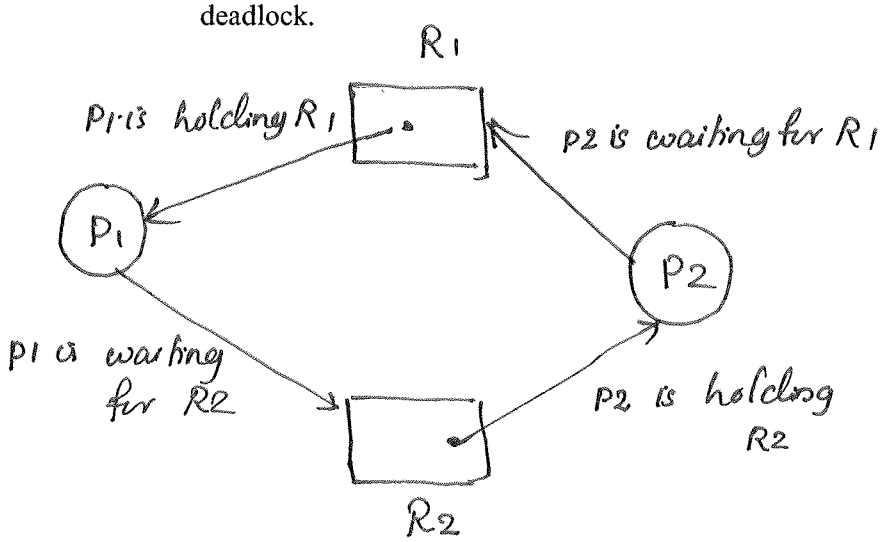
Qst. No.	Scoring Indicator	Split up score	Sub Total	Total
	PART A			
1	Set of programs to operate a computers and perform a specific task			1
2	Compiler, assembler, loader, interpreter	Any two ½ x 2 =1		1
3	The *ready queue* is a queue of all processes that are waiting to be scheduled on a CPU			1
4	I/O bound process is the one that spends more of its time doing I/O than it spends on doing computation. CPU bound process need very little I/O but require heavy computation.			1
5	CPU utilization, throughput, response time , waiting time etc.	Any two		1
6	*For some page-replacement algorithms, the page fault rate may increase as the number of allocated frames increases.*			1
7	Compaction and paging			1
8	User File Directory			1
9	Shortest seek time first			1
II 1.	Time-sharing systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems. In time sharing systems the prime focus is on minimizing the response time, while in multirogramming the prime focus is to maximize the CPU usage.	3		3
II 2.	1. File management 2. Memory management 3. Process management 4. Security 5. Device management 6. Input output managemen 7. User interface	Any three functions	3 x 1	3
II 3.	★ Each process is represented in the OS by a PCB ✪ **PROCESS STATE**: Any one of the five states ✪ **PROGRAM COUNTER**: Indicates address of the next instruction ✪ **CPU REGISTERS**: Status of CPU registers and flags ✪ **CPU-SCHEDULING INFORMATION**: Scheduling parameters ✪ **MEMORY-MANAGEMENT INFORMATION**: Value of base and limit registers, page tables, etc ✪ **ACCOUNTING INFORMATION**: Amount of CPU and real	Any three conditions	3 x 1	3

	time used, time limits, account numbers, process numbers etc. ✪ **I/O STATUS INFORMATION**: List of I/O devices allocated, list of open files, and so on. 			
II 4.	1. In preemptive scheduling, the CPU is allocated to the processes for a limited time whereas, in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to the waiting state. 2. The executing process in preemptive scheduling is interrupted in the middle of execution when a higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and waits till its execution. 3. In Preemptive Scheduling, there is the overhead of switching the process from the ready state to the running state, vise-verse, and maintaining the ready queue. Whereas in the case of non-preemptive scheduling has no overhead of switching the process from running state to ready state. 4. In preemptive scheduling, if a high-priorThe process The process non-preemptive low-priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. , in non-preemptive scheduling, if CPU is allocated to the process having a larger burst time then the processes with a small burst time may have to starve. 5. Preemptive scheduling attains flexibility by allowing the critical processes to access the CPU as they arrive in the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed. 6. Preemptive Scheduling has to maintain the integrity of shared data that's why it is cost associative which is not the case with Non-preemptive Scheduling.	Any three difference		3
II 5.	➢ **Virtual memory** – separation of user logical memory from physical memory. o Only part of the program needs to be in memory for execution o Logical address space can therefore be much larger than physical			3

	address space o Allows address spaces to be shared by several processes o Allows for more efficient process creation ➢ Virtual memory can be implemented via: o Demand paging o Demand segmentation			
II 6	• The **long-term scheduler** selects a balanced mix of I/O bound and CPU bound processes from the secondary memory (new state) and loads them into the main memory (ready state) for execution. Its primary objective is to maintain a good degree of multiprogramming [1] • The **short-term scheduler** decides which process to execute next from the ready queue. After it decides the process, Dispatcher assigns the decided process to the CPU for execution. Its primary objective is to increase the system performance [1]. • The **medium-term scheduler** swaps-out the processes from main memory to secondary memory to free up the main memory when required. Its primary objective is to perform swapping. It reduces the degree of multiprogramming [1].	1 mark for each		3
II 7.	→ ***Logical address*** – An address generated by the CPU is referred to as a logical address. → ***Physical address*** – An address seen by the memory unit—*that is, the one loaded into the memory-address register of the memory*—is referred to as a physical address. • Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme. • The set of all logical addresses generated by a program is a **logical address space.** • The set of all physical addresses corresponding to these logical addresses is a **physical address space.**	3		3
II.8	• Compile time • Load time • Run time	Each carries 1 mark		3
II.9	**1. Contiguous Allocation** **2. Linked Allocation** **3. Indexed Allocation**			

	• In a direct access storage device, data can be accessed directly, while in a sequential access storage device, data is accessed sequentially.			
II.9	**1. Contiguous Allocation** **2. Linked Allocation** **3. Indexed Allocation**			
II.10	Each user has her own **user file directory (UFD).** Each UFD has a similar structure, but lists only the files of a single user. Different users may have files with the same name. The file names within each UFD are unique. 	Explanation-1 mark Figure-2 marks		
III.1	**Multiprocessor system** A multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others. Following are some **advantages** of this type of system. • Enhanced performance • Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task. **Real time opeating system** If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks • It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling. • Well-defined fixed-time constraints • The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.** Eg: Air bag control in a vehicle • While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**. Eg : video conferencing	3 marks 4 marks		7

III.2	Assembler  • Translates the assembly language program in to machine code. • It Reserves space for data • Replaces he mnemonic codes by machine code • Replaces symbolic addresses by numeric addresses • Determines machine representation of constants Loader • A Loader is a system program that performs the loading function •It brings object program into memory and starts its execution Inteerpreter ❖ Interpreter takes single instruction as input ❖ No intermediate object code is generated ❖ Memory requirement is less ❖ Every time higher level program is converted into lower level program ❖ Errors are displayed for every instruction interpreted(if any) ❖ Eg: BASIC	3 marks 2 marks 2 marks		7
III.3	**Resource Allocation Graph** It is a directed graph that describes deadlock.It Consists of a set of vertices V and a set of edges E.V is partitioned into two different types of nodes ✪ P = {P1, P2,…,Pn} – All active processes in the system ✪ R = {R1, R2,…,Rn} - All resource types in the system ★ Two kinds of edges ✪ Request edge - Directed edge Pi ---> Rj ✪ Assignment edge - Directed edge Rj ----> Pi ★ If graph contains no cycles ✪ No deadlock	Explanation -5 marks		7

	★ If graph contains a cycle ✪ if only one instance per resource type, then deadlock ✪ if several instances per resource type, possibility of deadlock.  	Figure-2 marks		
III.4	**Deadlock** A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.A process is deadlocked if it is waiting for an event that will never occur.Deadlock situation can arise if the following four conditions hold simultaneously in a system: 1. **MUTUAL EXCLUSION**: Only one process at a time can use the resource 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 other processes. 3. **NO PREEMPTION:** Resource can be released only voluntarily by the process holding it, after that process has completed its task. 4. **CIRCULAR WAIT**: A circular chain of processes exists in which each process waits for one or more resources held by the next process in the chain.	Deadlock -2 marks Listing-1 mark Conditions-4 marks		7
III.5	Critical Section Problem Consider a system consisting of n processes {P0, P1, ...,Pn−1}. Each process has a segment of code, called a critical section in which the process may be accessing and updating data that is shared with at least one other process. When one process is executing in its critical section, no other process is allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.	Explanation- 5 marks		7

The critical-section problem is to design a protocol that the processes can use to synchronize their activity so as to cooperatively share data.

Each process must request permission to enter its critical section. The section of code implementing this request is the entry section.

The critical section may be followed by an exit section.

The remaining code is the remainder section.

The general structure of a typical process is

```
while (true) {
    Entry Section
        critical section
    Exit section
        Remainder section
}
```

Figure-2 marks

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual exclusion. If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. Progress. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. Bounded waiting. There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

III.6 FCFS

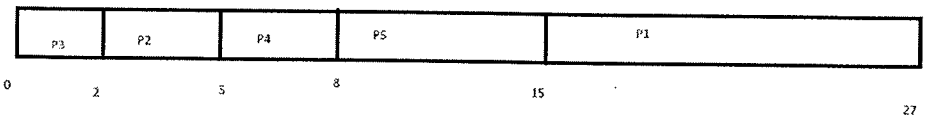


Chart- 2 marks

Average waiting time = (0+2+5+8+15)/5 = 6 milliseconds

Priority Scheduling

Calculation-1 mark

Chart-2

Average waiting time =(0+3+6+18+21)5= 9.6 milliseconds

Calculation-2 marks

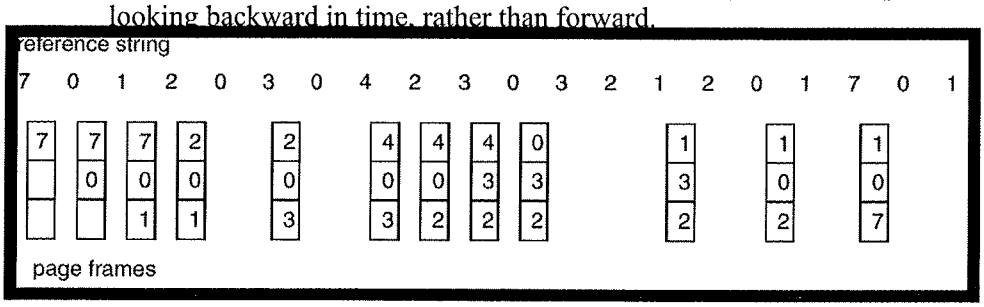
7

III.7

a) LRU replacement

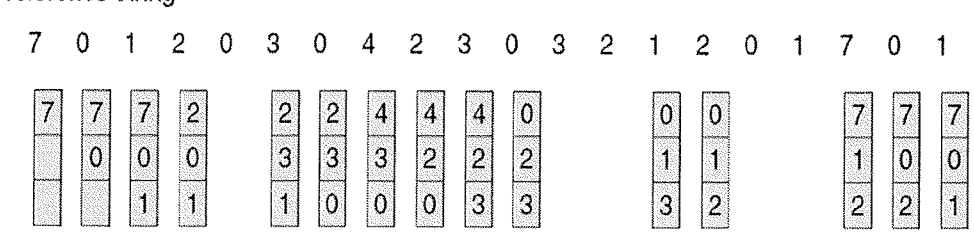
LRU Page Replacement

- The key distinction between the FIFO and OPT algorithms is that the FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be used.
- LRU replacement associates with each page the time of that page's last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.



b) FIFO replacement

- The simplest page-replacement algorithm is a ***first-in, first-out (FIFO)*** algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- Create a FIFO queue to hold all pages in memory.
- Replace the page at the head of the queue. When a page is brought into memory, insert it at the tail of the queue.
- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)



Disadvantage

Belady's anomaly: For some page-replacement algorithms, the page-fault rate may *increase* as the number of allocated frames increases

7

III.8

Segmentation

Memory management scheme that supports user view of memory. A logical-address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The mapping of logical address into physical address is affected by a segment table.

Each entry of the segment table has a segment base and a segment limit

A logical address consists of two parts: a segment number, s, and an offset into that segment, d. The segment number s is used as an index into the segment

Explanation Expls. 4

7

table. The offset d of the logical address must be between 0 and the segment limit

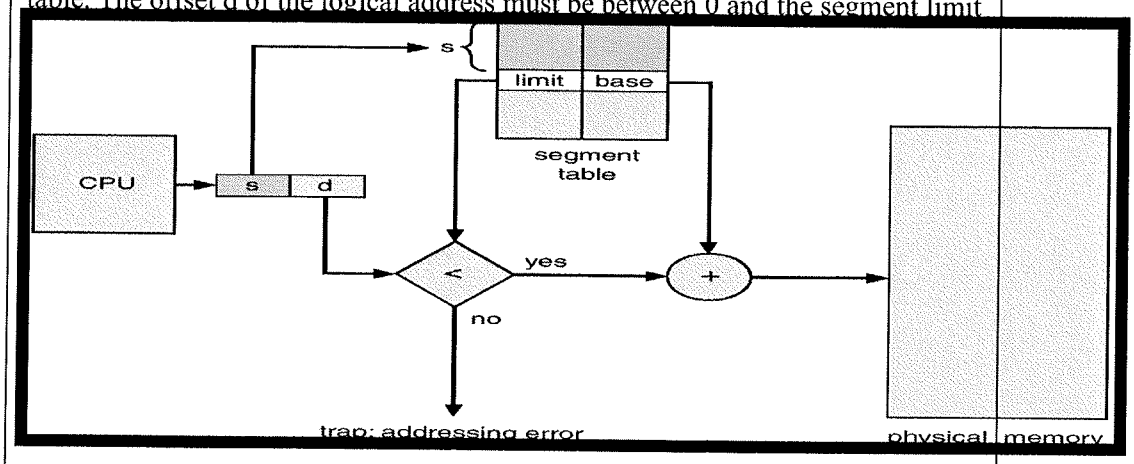


Figure 2

III.9

Page Fault

Access to a page marked invalid causes a page-fault.

Paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.

3 mark

Page Fault Handling

1. Check internal table for this process to determine whether the reference was a valid or invalid memory access
2. If the reference was invalid, we terminate the process
3. If it was valid, and is not brought in, now page in
   a) Find a free frame.
   b) Read the desired page into the newly allocated frame.
   c) Modify the internal tables

4 mark

7

III.10

Fragmentation

- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes. This fragmentation problem can be severe. In the worst case, we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

3 mark

- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous:

7

storage is fragmented into a large number of small holes.

- **External fragmentation** – we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes. Total memory space exists to satisfy a request, but it is not contiguous.
- **Internal fragmentation** – in fixed size partition, allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Compaction- external fragmentation can be reduced by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible only if relocation is dynamic, and is done at execution time.

Another possible solution to the external-fragmentation problem is: **Paging and Segmentation**

Types
4 mark

III.11 File Operations

2. CREATING A FILE:

   Step1: space in the file system must be found for the file.

   Step2: an entry for the new file must be made in the directory.

3. WRITING A FILE: system call is made with the name of the file and the information to be written to the file

4. READING A FILE: system call is made with the name of the file and where the next block of the file should be put

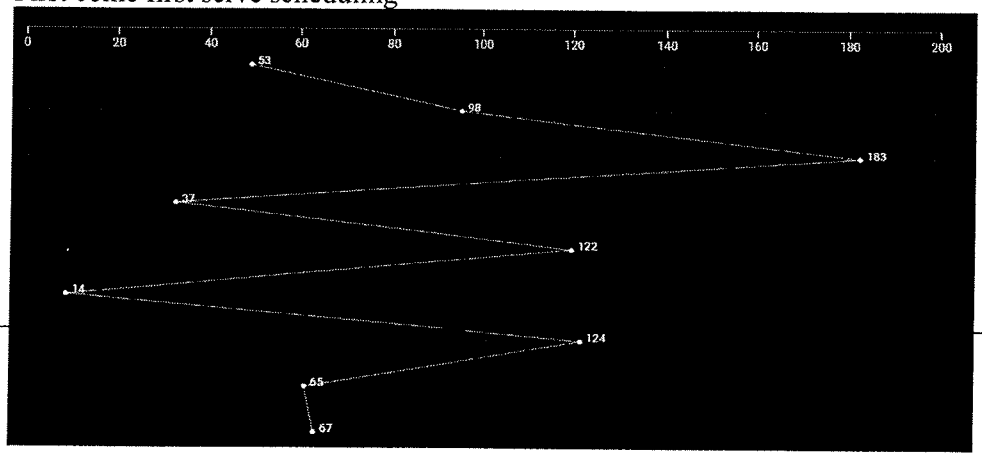
5. REPOSITIONING WITHIN A FILE: The directory is searched for the appropriate entry, and the current-file-position is set to a given value

6. DELETING A FILE: The directory is searched for the named file. Having found the associated directory entry, release all file space and the directory entry

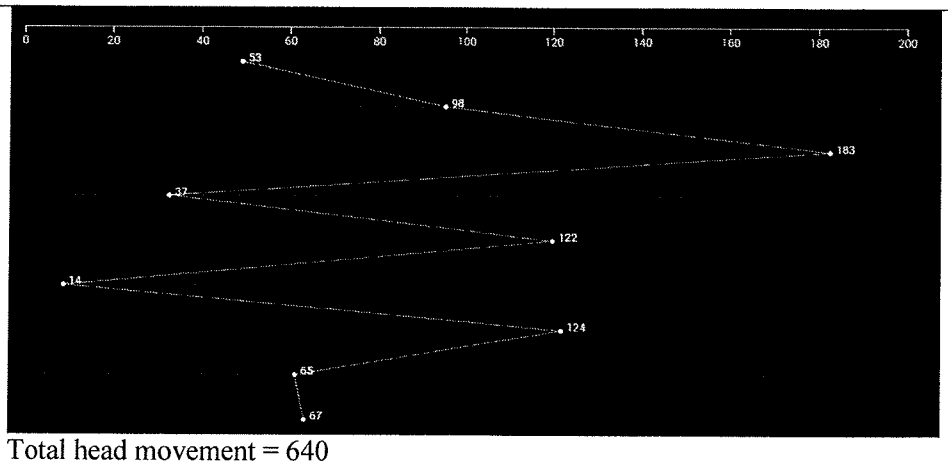
7. TRUNCATING A FILE: To erase the contents of a file but keep its attributes

7

III.12 First come first serve scheduling

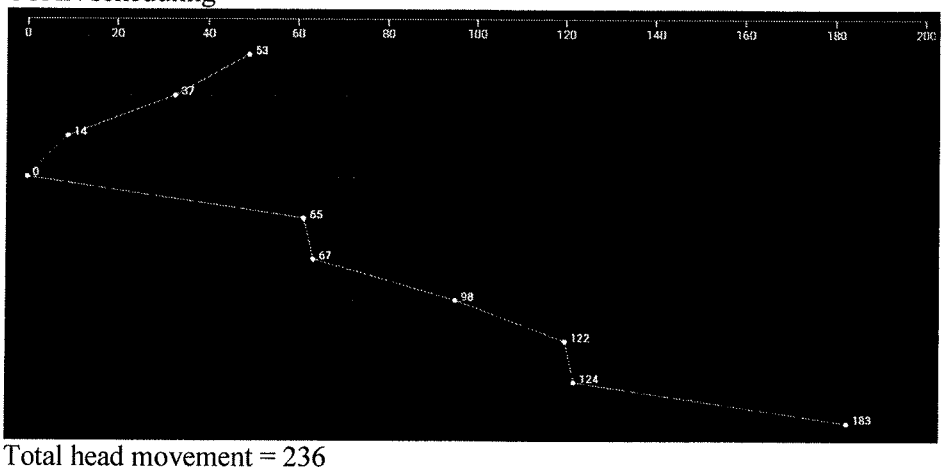


7



Total head movement = 640

SCAN scheduling



Total head movement = 236

		Figure-2 Calculation-1	3	
		Fig- 2 marks Cal - 2marks	4	