Simulation Of Disk Scheduling Algorithms

By

Major Waqar Ahmed

ABSTRACT

Speed, accuracy and bulk storage are the essence of a computer system. One of the tasks of the operating system is the optimum utilization of available hardware. Operating System designers have always strived for achieving highest standards of speed, accuracy and efficient handling of secondary storage devices. Disk drives are the major secondary storage Input / Output device on all computers. Requests for disk input / output are generated by the file system and by the virtual memory system. Each request specifies the address on the disk to be referenced, in the form of a logical block number.

In multiprogramming environment more processes need to access the disk for reading and writing of data at the same time. Therefore a queue of requests to be serviced, is always pending. It requires disk drives to have a fast access time and more bandwidth. We can improve both the access time and the bandwidth by scheduling the servicing of disk input / output requests in a better order.

Different scheduling algorithms available for this purpose are First Come First Serve (FCFS), Shortest Seek Time First (SSTF), LOOK, C- LOOK, SCAN and C-SCAN Scheduling.

This work is to design and develop a simulator for disk scheduling utilizing SCAN (both up ward and down ward direction), C-SCAN algorithms and provide performance results. Simulation results will be compared with the previous [1] experimental results (obtained through FCFS, SSTF, LOOK and C-LOOK) to ascertain its efficacy and usefulness.

ACKNOWLEDGEMENTS

All praises belong to Allah, His grace and mercy granted me the opportunity, strength and ability for acquiring knowledge, for my own I had nothing, as life, faculties, capabilities, means, relations, friends are all his gifts.

I would like to express my deepest gratitude to my thesis adviser Dr. Muhammad Riaz and Co-adviser Lt Col Dr Muhammad Younus Javed for their whole hearted guidance, wisdom, valuable time and motivation throughout my thesis research, despite their busy routine and commitments.

I am also thankful to Lab staff of Computer Engineering Department, College of EME for assisting me in completion of thesis work.

I am thankful to my Family for their encouragement, and cooperation throughout my research work.

Table of Contents

List of Figures

List of Tables

List of Graphs

Chapter 1
Introduction

1.1 What Is An Operating System?

An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware. Operating system should have [2] three basic functions:-

- ***Convenience*** An operating system makes a computer more convenient to use.
- ***Efficiency*** An operating system allows the computer system resources to be used in an efficient manner.
- ***Ability to evolve*** An operating system should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with services.

1.1.1 The Operating System as a User/Computer Interface

The hardware and software are used for providing applications to a user, which can be viewed in a layered, or hierarchical, fashion as depicted in Figure 1.1. The end user is generally not concerned with the computer's architecture. The operating system masks the details of the hardware from the user and provides the user with a convenient interface for using the system. It acts as mediator, making it easier for the user and for application programs to access and use facilities and services. Briefly, the operating system provides [4] following services:

- ***Program Creation*** The operating system provides a variety of facilities and services, such as editors and debuggers, to assist the programmer in creating programs. These services are in the form of utility programs that are not actually part of the operating system but are accessible through the operating system.

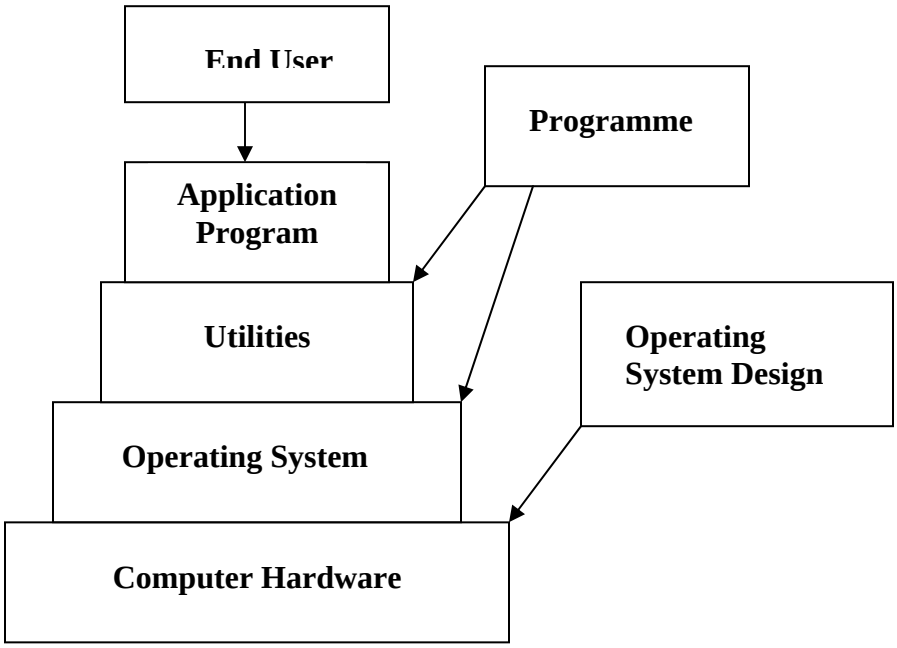


Figure 1.1 An Overview Of Computer System

- ***Program Execution*** Operating System has to perform number of tasks to execute a program. Such as loading of instructions and data into main memory, controlling of I/O devices, memory management, initialization / space management of files and management / control of other resources.

- ***Access To I/O Devices*** Each I/O device requires its own peculiar set of instructions, or control signals, for operation. The operating system takes care of these details so that the programmer can think in terms of simple read and write operations.

- ***Controlled Access To Files*** Each Operating System has its own file format for data storage on media. The embedded file system provides file level security.

- ***System Access*** In case of a shared or public system, the operating system controls access to the system as a whole and to specific system resources. The access function provides protection of resources and data from unauthorized users and resolves conflicts in the contention for resources.

- ***Error Detection And Response*** A variety of errors can occur while a computer system is running. These include internal and external hardware errors, such as a memory error, device failure or malfunction; and various software errors, such as arithmetic overflow, attempt to access forbidden memory location, and inability of the operating system to grant the request of an application. In each case, the operating system must make the response that clears the error condition with the least impact on running applications.

- ***Accounting****: A* good operating system collects usage statistics for various resources and monitors performance parameters such as response time. On any system, this information is useful in anticipating the need for future enhancements and in tuning the system to improve performance.

1.2 Types of Memory

Computer memory can be divided into two main types i.e. primary storage or semiconductor main memory and secondary storage. The main purpose of a computer system is to execute programs. These programs together with the data they access must be in main memory during execution. Ideally, we would want the programs and data to reside in main memory permanently, but it is not possible for two reasons. Firstly the main memory is usually too small to store all needed programs and data permanently. Secondly main memory is a volatile storage device that loses its contents when power is turned off or lost.

The wide variety of storage media in a computer can be organized in a hierarchy according to either speed or their cost (as shown in Figure 1.2). The higher levels are expensive, but fast. As we move down the hierarchy, the cost per bit decreases, whereas the access time increases, and the amount of storage at each level increases. The higher levels are fast, but expensive and lower levels of memory hierarchy are cheaper but slower.

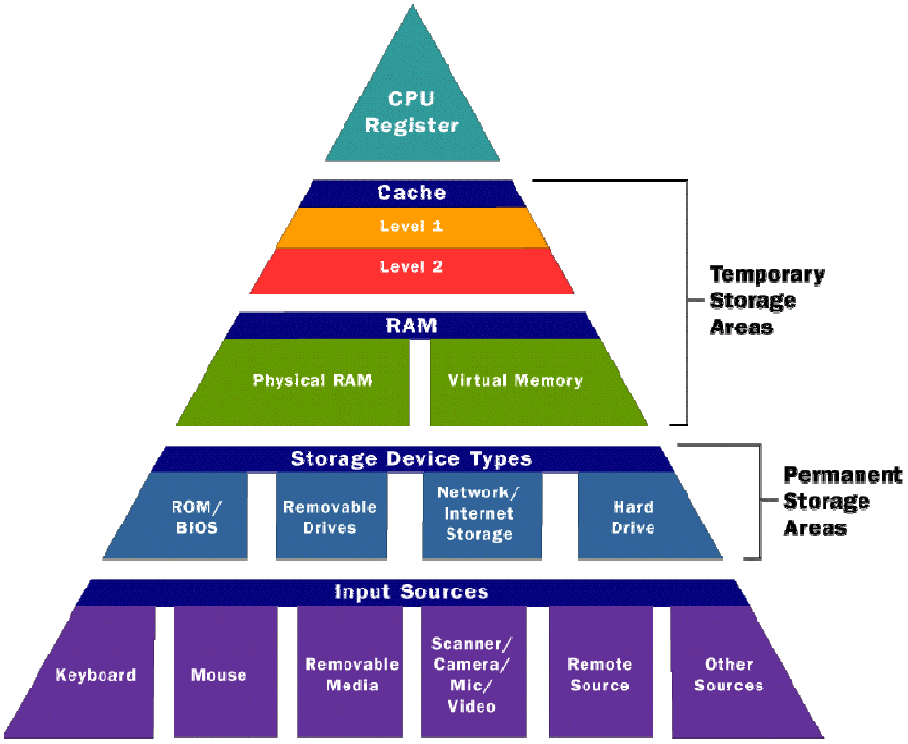


Figure 1.2 Hierarchy Of Memory

1.3 Magnetic Disk

Disks provide the bulk of secondary storage for modern computer systems. Disk is physically simple (Figure1.3). It is a circular platter constructed of metal or of plastic coated with a magnetizable material. Data is recorded and retrieved from the disk via a conducting coil, named as head. The write mechanism is based on the magnetic field produced by electricity flowing through a coil. Pulses are sent to the head, and magnetic patterns are recorded on the surface below, with different patterns for positive and negative currents. The read mechanism is based on the electric current in a coil produced by a magnetic field moving relative to the coil. When the surface of the disk passes under the head, it generates a current of the same polarity as the one already recorded.

1.3.1 Secondary Storage Management

In secondary storage, a file consists of a collection of blocks. In operating system the file-management system is responsible for allocating blocks to files. This raises two management issues. First, space on secondary storage must be allocated to files, and second, it is necessary to keep track of the space available for allocation[4]. Both of these tasks are interrelated; the approach taken for allocation of files may influence the approach taken for management of free space.

1.3.2 File Allocation Methods

There are three methods of file allocation

a. **Contiguous Allocation**. A single contiguous set of blocks is allocated to a file at the time of file creation. This is a pre allocation strategy that uses portions of variable size. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. Contiguous allocation is the best for the individual sequential file. Multiple blocks can be brought in one at a time to improve I/O performance for sequential processing. Moreover it is also easy to retrieve

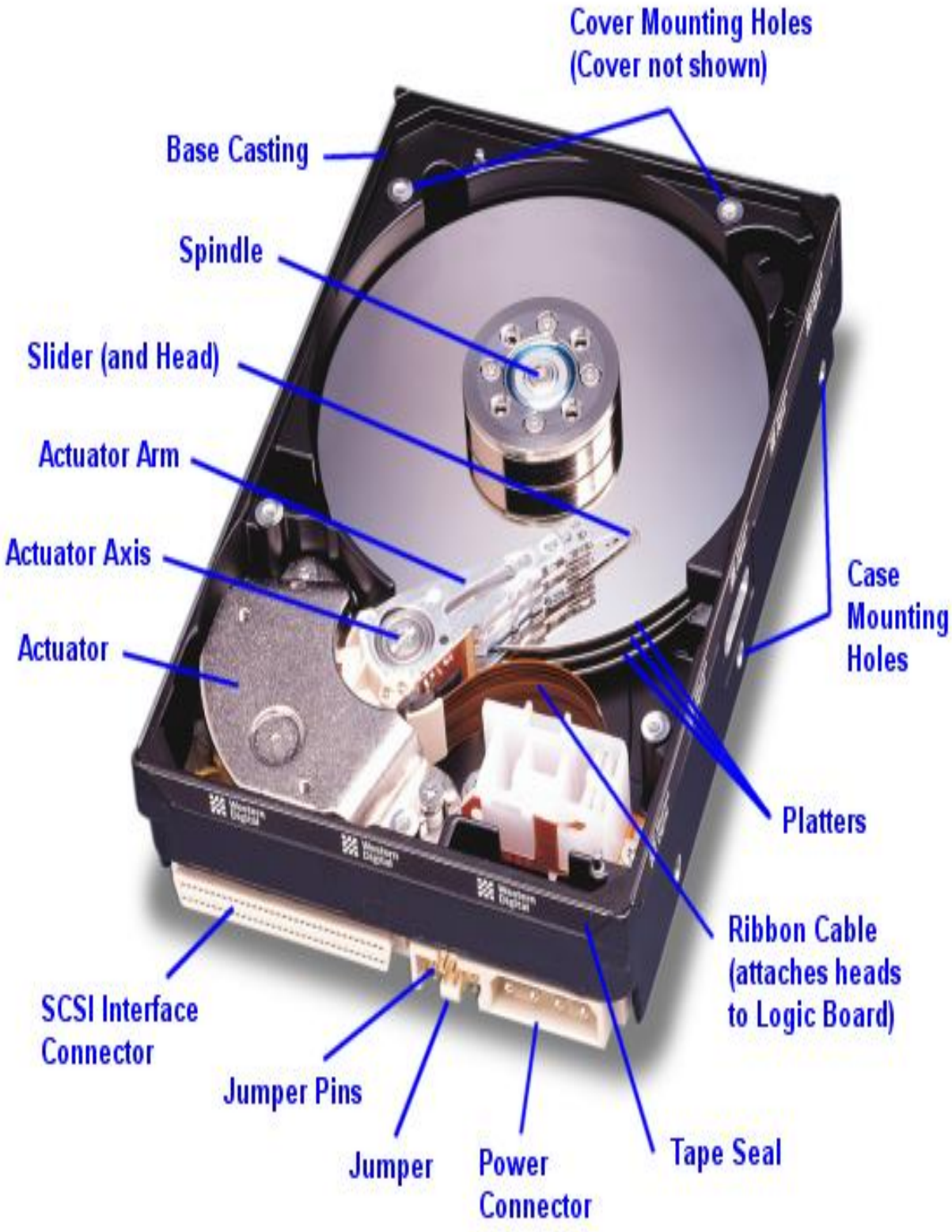


Figure 1.3 Overview of Hard Disk

a single block. Problem with contiguous allocation is of external fragmentation, making it difficult to find contiguous blocks of space of sufficient length. From time to time, it will be necessary to perform a compaction algorithm to free up additional space on the disk, with pre allocation, it is necessary to declare the size of the file at the time of creation.

b. **Chained Allocation.** This allocation is on the basis of an individual block. Each block contains a pointer to the next block in the chain. File allocation table needs just a single entry for each file to show the starting block and the length of the file. Although pre allocation is possible, but it is more common to simply allocate blocks as needed. The selection of blocks is a simple matter; any free block can be added to the chain. There is no external fragmentation because only one block at a time is needed. This type of physical organization is best suited to sequential files that are to be processed sequentially. To select an individual block of a file requires tracing through the chain to the desired block. One consequence of chaining, is that it does not accommodate the principle of locality.

c. **Indexed Allocation.** This type of allocation addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file; the index has one entry for each portion allocated to the file. Typically, the file indexes are not physically stored as part of the file allocation table. Rather, the file index for a file is kept in a separate block, and the entry for the file in the file allocation table points to that block. Allocation may be on the basis of either fixed size blocks or variable-size portions. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size portions improves locality. In either case, files may be consolidated from time to time. Consolidation reduces the size of the index in the case of variable-size portions but not in the case of block allocation. Indexed allocation supports both sequential and direct access to the file, and thus it is the most popular form of file allocation.

1.4 Free Space Management

As the space allocated to files must be managed, similarly the space that is not currently allocated to any file needs to be managed. To perform any of the file allocation techniques that have been described, it is necessary to know what blocks on the disk are available for allocation. Thus, we need a disk allocation table in addition to a file allocation table. Three techniques are in common use [4] bit tables, chained free portions, and indexing.

1.4.1 Bit Tables

The method of bit tables uses a vector containing 1 bit for each block on the disk. Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use. For example, for the disk layout of say 35 blocks, a vector of length 35 is needed, and would have the following value

00111000011111000011111111111011000

A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. A bit table works well with any of the file allocation methods we have discussed. Another advantage is that it is as small as possible and can be kept in main memory. This avoids the need to read the disk allocation table into memory every time an allocation is performed.

1.4.2 Chained Free Portions

Using a pointer and length value in each free portion may chain the free portions together. This method has negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain and the length of the first portion. This method is suited to all the file allocation methods. If allocation is made on the basis of a block at a time, simply choose the free block at the head of the chain and

adjust the first pointer or length value. If allocation is made by variable-length portion, a first-fit algorithm may be used: The headers from the portions are fetched one at a time to determine the next suitable free portion in the chain. Again, pointer and length values are adjusted.

1.4.3 Indexing

The indexing approach treats free space as a file and uses an index table as has been described in the subsection, 1.3.2 "File Allocation Methods." For efficiency, the index should be on the basis of variable-size portions rather than blocks. Thus, there is one entry in the table for every free portion on the disk. This approach provides efficient support for all the file allocation methods.

1.5 Hard Disk Errors

Some of the common hard disk errors are:-

a. Programming error (e.g request for nonexistent sector).
b. Transient checksum error (e.g caused by dust on the head).
c. Permanent checksum error (e.g disk block physically damaged).
d. Seek error (e.g the arm sent to cylinder 6 but it went to 7).
e. Controller error (e.g controller refuses to accept commands).

Programming errors occur when the driver tells the controller to seek to a nonexistent cylinder, read from a nonexistent sector, use a nonexistent head, or transfer to or from nonexistent memory. Most controllers check the parameters given to them and complain if they are invalid.

Transient checksum errors are caused by dust in the air that get between the head and the disk surface. Most of the time just repeating the operation a few times can

eliminate them. If the error persists, the block has to be marked as a *bad block* and avoided next time.

One way to avoid bad blocks is to write a very special program that takes a list of bad blocks as input and carefully put all these bad blocks in a file. Once this file has been made, the disk allocator will think these blocks are occupied and never allocate them. As long as no one ever tries to read the bad block file, no problem will occur.

Not reading the bad block file is easier said than done. Many disks are backed up by copying their contents a track at a time to a backup tape or disk drive. If this procedure is followed, the bad blocks will cause trouble. Backing up the disk one file at a time will-solve the problem, provided that the backup program knows the name of the bad block file and refrains from copying it.

Seek errors are caused by mechanical problems in the arm. The controller keeps track of the arm position internally. To perform a seek; it issues a series of pulses to the arm motor, one pulse per cylinder, to move the arm to the new cylinder. When the arm gets to its destination, the controller reads the actual cylinder number (written when the drive was formatted). If the arm is in the wrong place, a seek error has occurred.

Most hard disk controllers correct seek errors automatically, but many floppy controllers just set an error bit and leave the rest to the driver. The driver handles this error by issuing a *Recalibrate* command, to move the arm as far as it will go and reset the controller's internal idea of the current cylinder to 0. Usually this solves the problem, if it does not, the drive must be repaired.

The controller is a specialized little computer, complete with software, variables, buffers, and occasionally, bugs. Sometimes an unusual sequence of events such as an interrupt on one drive occurring simultaneously with a *Recalibrate* command for another drive will trigger a bug and cause the controller to go into a loop or lose track of what it was doing. Controller designers usually plan for the worst and provide a pin on the chip

which, when asserted, forces the controller to forget what it was doing and reset itself. If all else fails, the disk driver can set a bit to invoke this signal and reset the controller. If that does not help, all the driver can do is print a message and give up [3].

1.6 Improving The Performance Of Disk Systems

The performance of the disk system on a computer is often the most significant factor in determining the overall speed of an application. Disks are the slowest main computer component, thus they are the major bottleneck in system performance and reliability. In addition, a disk crash is one of the most catastrophic computer hardware failures. Which not only require the disk be replaced, but restoration of the disk's data is also required. This may take hours, as backup copies of data on tape are transferred to the disk. Under normal circumstances, these restored data are not a precise image of the disk when it crashed. Backups are usually performed daily or weekly, meaning that any changes to the data since the last backup are lost if a disk crashes. Improving disk speed and reliability is therefore an important topic in the current research [l]. A number of techniques, which are applied to this end, are described here. To summaries, these are:-

- Blocking
- Disk Cashing
- RAM disks
- File re-organization.

1.6.1 Blocking

In general, the actual mechanics of transferring data to and from a disk system are ‘transparent’ (invisible) to the application program; request is made to the operating system for, say, 100 bytes or for one record. (i.e one block).

In reality, data will always be transferred to and from a disk in units of the physical block or sector size of the disk-typically 512 bytes. To service a request for the first 100 bytes of a file, the operating system must read the whole of the first block into a

buffer in memory, and consequently extract the 100 bytes as requested. Requests for further data, e.g. the next l00 bytes, are met from the stored block. Further data transfer from the disk will only occur when a different block is requested. A similar process occurs on writing. If a series of sequential bytes are written from an application program, they are assembled in the buffer until a full block is available, which is then transferred to disk.

These processes of blocking (assembling a block for output) and deblocking (unpacking a block for output) minimize the actual number of disk transfers, at the expense of memory space and some processing complexity. The greatest benefit occurs when large amounts of sequential data are being processed; if data access is primarily random, then the blocking system will actually slow down the theoretical transfer rate. However, the block transfer is an integral part of the way disks operate and is therefore unavoidable [2].

1.6.2 Disk Caching

Disk caching is an extension of the buffering concept and is particularly applicable in multiprogramming machines or in disk file servers. A set (cache) of buffers is allocated to hold a number of disk blocks which have been recently accessed. If the data in a cache buffer is modified, only the local copy is updated. Thus, processing of the data takes place using the cached data avoiding the need to frequently access the disk itself. Caching is used in many systems.

Ultimately, the stored block must be transferred to disk to maintain the integrity of the file data. This will certainly happen when the file is closed, but could occur earlier for a number of reasons.

First, the cache buffer may be required for another block for which a read command was received by the system. The caching mechanism makes the assumption that the most recently accessed blocks are likely to be accessed again fairly soon. Thus, if

a new block is required to be shifted to the cache buffer, one buffer will be selected for 'flushing' to disk on the basis of being least recently used.

It should be noted that system using disk caching are risking loss of updated information in the event of machine failures such as loss of power. For this reason, the system may periodically flush the cache buffers in order to minimize the amount of data loss [2].

1.6.3 RAM Disk

A RAM disk (Figure 1.4) is a simulation of a conventional disk service using semiconductor random access memory. A RAM disk is simpler. It has the advantage of having instant access (no seek or rotational delay). There are two alternative implementations of this idea. One system simply uses a large semiconductor store as peripheral device providing a facility similar to a magnetic disk system but with very much faster disk times. The device controller would appear to the computer exactly like a conventional disk system. The merit of such a system is the faster speed, but it is correspondingly more expensive. Consequently, it would be of interest primarily in special applications where speed is of the essence. It is possible that such a device would provide only a small portion of the total disk capacity of the computer, holding the most-time critical data. Since most forms of RAM storage are volatile, the RAM disk generally requires an independent power supply [4].

The other form of RAM disk employs software to simulate a disk in the main memory of the computer. This is achieved by installing a device driver which responds to the operating system and to the user exactly like a disk device, to the extent that the RAM disk appears to have sectors and cylinders. In MS-DOS, the RAM disk would be assigned

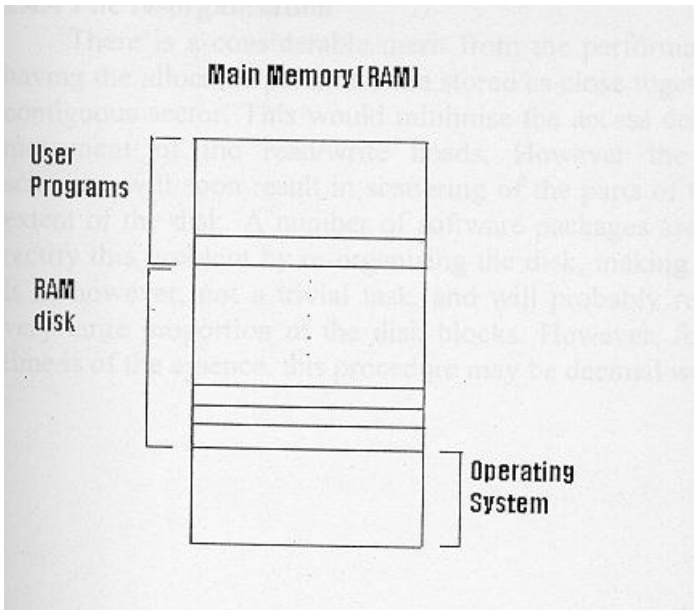


Figure 1.4 RAM disk.

to a volume driver letter such as D. All operations valid for disk will work equally well with the RAM disk, for both interactive and programmed access.

The advantage of the RAM disk idea is that it can be applied to any file (or even more than one file) with little programming involvement, so that it can be utilized easily in circumstances where the user requires extra speed in an application.

1.6.4 File Re-Organisation

There is a considerable merit from the performance point of view in having the allocated parts of a file stored as close together as possible, i.e. in contiguous sector. This would minimize the access delay caused by radical movement of the read/write heads. However the dynamic allocation schemes will soon result in scattering parts of the file across a wide extent of the disk. A number of software packages are available which can rectify this problem by re-organizing the disk, making each file contiguous. It is, however, not a trivial task, and will probably result in movement of very large proportion

of the disk blocks. However, for applications where time is of the essence, this procedure may be deemed appropriate [2].

Chapter 2
Disk Scheduling and Algorithms

2.1 Introduction

Since most jobs depend heavily on the disk for loading and input and output file, it is important that disk service be as fast as possible. The operating system can improve the disk service time by scheduling the requests for disk access.

2.2 Disk I/O Operations

Over the past 30 years, the increase in the speed of processors and main memory has far outstripped that of disk speed, This gap is expected to continue into the foreseeable future. Thus, the performance of disk storage subsystems is of vital concern, and much research has gone into schemes for improving the performance. In this section, some of the key issues will be highlight and the most important approaches will be discussed.

2.2.1 Disk Performance Parameters

The actual details of disk I/O operation depend on the computer system, the operating system, and the nature of the I/O channel and disk controller hardware. When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired cylinder/track and at the beginning of the desired sector on that track. Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system. On a movable-head system, the time it takes to position the head at the track is known as seek time. In either case, once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for the beginning of the sector to reach the head is known as rotational delay, or rotational latency. The sum of the seek time (if any) and the rotational delay is equal to the access time, the time it takes to get into position to read or

write. Once the head is in position, the Read or Write operation is then performed as the sector moves under the head.

2.2.2 Seek Time

Seek time is the time required to move the disk arm to the required track[4]. It turns out that this is a difficult quantity to pin down. The Seek time consists of two key components the initial startup time and the time taken to traverse the cylinders that have to be crossed once the access arm is up to speed. The traversal time is, not a linear function of the number of tracks. We can approximate Seek time with the linear formula:

$T_s = m \times n + s$

where

T_s = estimated seek time

n = number of tracks traversed

m = constant that depends on the disk drive

s = startup time

For example, an inexpensive Winchester disk on a personal computer might be approximated by m = 0.3 msec and s = 20 msec, whereas a larger, more expensive disk drive might have m = 0.1 msec and s = 3 msec.

2.2.3 Rotational Delay

Disks, other than floppy disks, typically rotate at 3600 rpm, which is one revolution per 16.7 msec. Thus, on the average, the rotational delay will be 8.3 mses. Floppy disks rotate much more slowly, typically between 300 and 600 rpm. Thus the average delay will be between 100 and 200 msec.

2.2.4 Transfer Time

The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion:

$T = b/rN$

Where

T = transfer time

B = number of bytes to be transferred

N = number of bytes on a track

r = rotation speed in revolutions per second

thus the total average access time can be expressed as $T_a = T_s + 1/2r + b/rN$

Where T_s is the average seek time.

2.2.5 Timing Comparison

Having defined the determining parameters, let us look at two l/O operations that illustrate the danger of relying on average values. Consider a typical disk [4] with an advertised average Seek time of 20 msec, a transfer rate of l MB/s, and 512-byte sectors with 32 sectors per track. Suppose that we wish to read a file consisting of 256 sectors for a total of 128KB. We would like to estimate the total time for the transfer. First, let us assume that the file is stored as compactly as possible on the disk. The file occupies all the sectors on eight adjacent tracks (8 tracks x 32 sectors/ track = 256 sectors). This is known as sequential organization. Now, the time to read the first track is as follows:

Average seek	20 msec
Rotational delay	8.3 msec
Read 32 sectors	16.7 msec
	45 msec

Suppose that the remaining tracks can now be read with essentially no Seek time. That is, the l/O operation can keep up with the flow from the disk. Then, at most, we need to deal with rotational delay for each succeeding track. Thus, each successive track is read in 8.3 + 16.7 = 25 msec. To read the entire file:

Total time = 45 + 7 x 25 = 220 msec = 0.22 sec

Now let us calculate the time required to read the same data by using random access rather than sequential access; that is, access to the sectors is distributed randomly over the disk. For each sector, we have:

Average seek	20 msec

Rotational delay	8.3 msec
Read 32 sectors	0.5 msec
	28.8 msec

Total time = 256 x 28.8 = 7373 msec = 7.37 sec.

It is clear that the order in which sectors are read from the disk has a tremendous effect on l/O performance.

2.3 Disk Scheduling Algorithms

On examining the example in the previous section, we see that the reason for the difference in performance can be pinned down to Seek time. If sector access requests involve selection of tracks at random, then the performance of the disk I/O system will be as poor as possible. To improve performance, we need to reduce the average time spent on Seeks. Consider the typical situation in a multiprogramming environment, in which the operating system maintains a queue of requests for each I/O device. Different algorithms are used for selecting request for servicing from the queue of requests. Following are the main types of algorithms used for this purpose:-

- FCFS Scheduling
- SSTF Scheduling
- SCAN Scheduling
- Circular-SCAN Scheduling
- Look Scheduling
- Circular-Look Scheduling

2.3.1 FCFS Scheduling

The simplest form of disk scheduling is, of course, *first-come, first-served* (FCFS) scheduling. This algorithm is easy to program and is intrinsically fair. However, it may not provide the best (average) service. Consider for example (Figure 2.1), [4] an ordered disk queue with requests involving tracks[2]:-

98, 183, 37,122, 4,124, 65 and 67

Listed first (98) to last (67). If the read-write head is initially at track 53, it will first move from 53 to 98, then to 183, 37, 122,14,124,65, and finally to 67 for a total head movement of 640 tracks.

98,37, 122,14,124,65,67,14

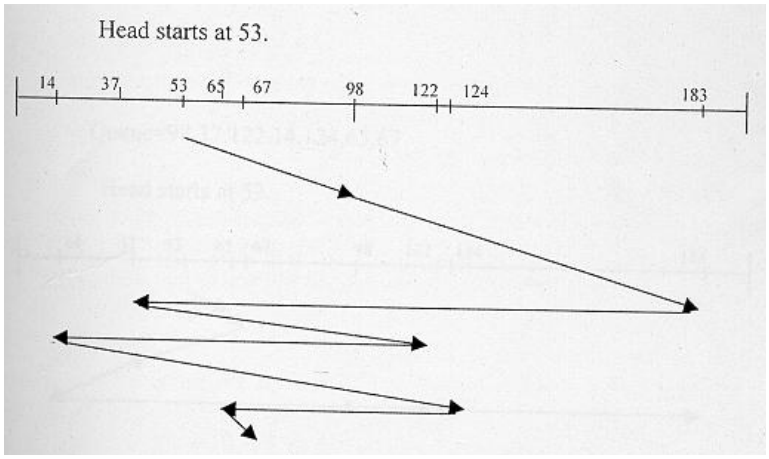


Figure 2.1 FCFS Disk Scheduling

The problem with this algorithm is illustrated by the wild swing from 122 to 14 and then back to 124. If the requests for tracks 37 and 14 could be serviced together, before or after the requests at 122 and 124, the total head movement can be decreased substantially and the average time to service each request would decrease, improving disk throughput [2].

2.3.2 SSTF Scheduling

It seems reasonable to service together all requests close to the current head position, before moving the head far away to service another request. This assumption is the basis for the shortest-seek time first (SSTF) disk-scheduling algorithm. The SSTF algorithm selects the request with the minimum seek time from the current head position. Since the seek time is generally proportional to the track difference between the requests,

we implement this approach by moving the head to the closest track in the request queue [1].

For our example [2] request queue (Figure 2.2), the closest queue to the initial head position (53) is at track 65. Once we are at track 65, the next closet request is at track 67. At this point, the distance to track 37 is 30, whereas the distance to track 98 is 31. Therefore, the request at track

65, 67, 37, 14, 98, 122, 124, 183

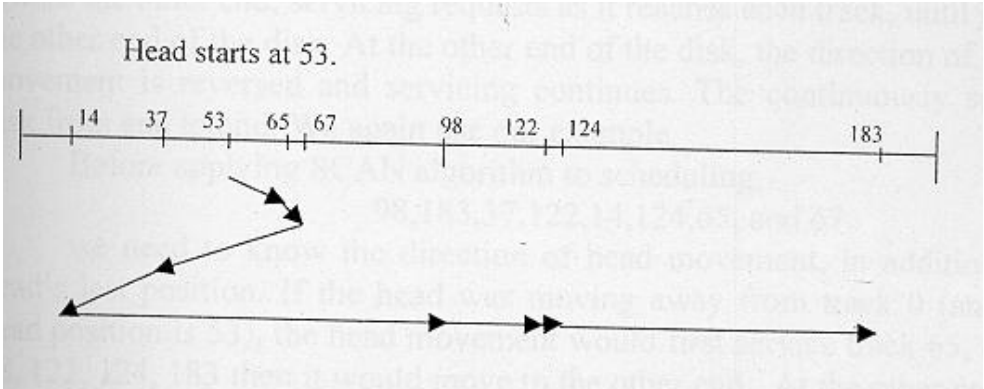


Figure 2.2 SSTF Disk Scheduling

37 is closer and is served next. Continuing, we service the request at track 14, then 98, 122,124 and finally at 183. This scheduling method results in a total head movement of only 236 tracks, little more than one-third of the distance-needed for FCFS scheduling. This algorithm would result in a substantial improvement in average disk service.

SSTF has the drawback that it may cause *starvation* of some requests. It is worth noting that, in a real system, requests may arrive at any time. Consider for example, we have two requests in the queue, for tracks 14 & 186. If a request near 14 arrives while we are servicing that request, it will be serviced next making the request at 186 wait. While this request is being serviced, another request close to 14 could arrive. In theory, a continual stream of requests near one another could arrive, causing the request for 186 to wait indefinitely.

The SSTF algorithm, although a substantial improvement over the FCFS algorithm, is not optimal. For example, if we move the head from 53 to 37, even though the latter is not closest, and then to 14, before turning around to service 65, 67, 98, 122, 124 and 183, we can reduce the total head movement to 98 tracks.

2.3.3 SCAN Scheduling

Recognition of the dynamic nature of the request queue leads to the SCAN algorithm [4]. The read-write head starts [2] at one end of the disk, and moves toward the other end, servicing requests as it reaches each track, until it gets to the other end of the disk. At the other end of the disk, the direction of the head movement is reversed and servicing continues. The head continuously scans the disk from end to end. We again use our example. Before applying SCAN algorithm to scheduling,

98, 183, 37, 122, 14, 124, 65, and 67

We need to know the direction of head movement, in addition to the head's last position. If the head was moving away from track 0 (and initial head position is 53), the head movement would first service (Figure 2.3a) track 65, then 67, 98,122,124, 183 then it would move to the other end. At the other end of the disk the head does not go toward the beginning of the disk and will first service request 37. After servicing track 37, it would then service track 14. For Scan/D it would first service track 53(Figure 2.3b).

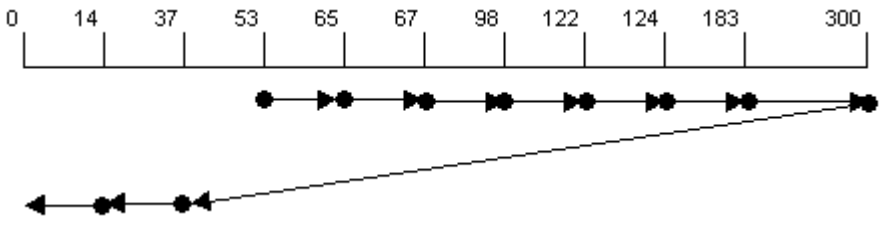


Figure 2.3a Scan/U Disk Scheduling

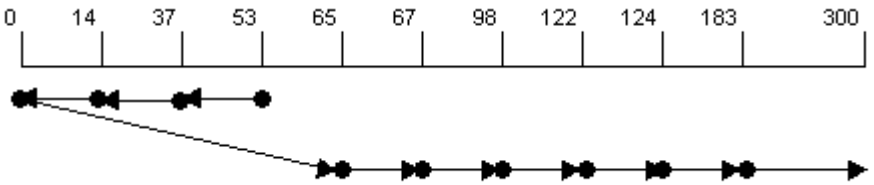


Figure 2.3b Scan/D Disk Scheduling

The SCAN algorithm is sometimes called elevator algorithm, since it is similar to the behavior of elevators as it service requests to move from floor to floor in a building. Another analogy is that of shoveling snow from a sidewalk during a snowstorm. Starting from one end, it removes snow and moves toward the other end. As it moves, new snow falls behind. At the far end, it reverses direction and removes the newly fallen snow behind [1].

2.3.4 C-SCAN Scheduling

A variant of SCAN scheduling that is designed to provide a more uniform wait time is C-SCAN (circular SCAN) scheduling [2]. As SCAN scheduling moves the head from one end of the disk to the other, servicing requests as it goes. When it reaches to the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. C-SCAN scheduling essentially treats the disk as circular, with the last track adjacent to the first one. The reordered queue for C-Scan/U (Figure 2.4 a) will be 65, 67, 98, 122, 124, 183, 300, 0, 37 ,14 and for Scan/D (Figure 2.4 b)will be 37, 14,0, 65, 67, 98, 122, 124 and 183

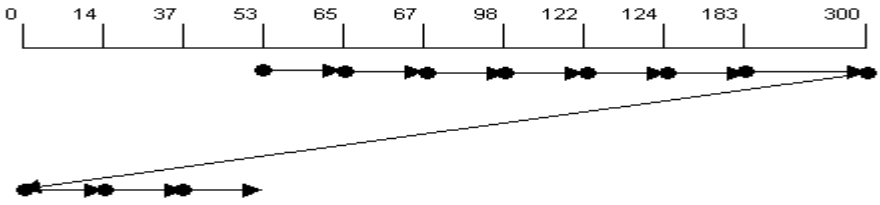


Figure 2.4a C-Scan/U Disk Scheduling

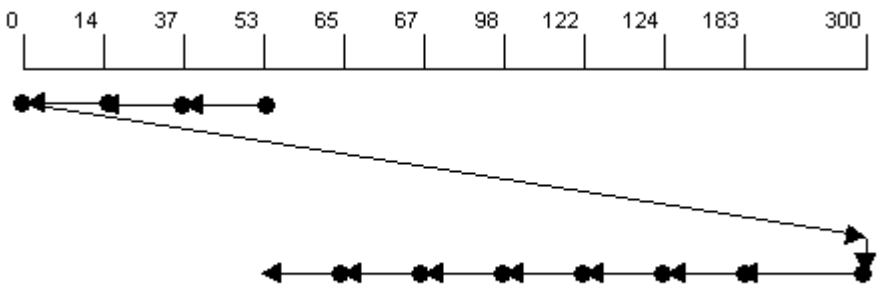


Figure 2.4b SCAN/U Disk Scheduling

2.3.5 Look Scheduling

Notice that, as we have described them, both SCAN and C-SCAN scheduling always move the head from one end of the disk to the other. In practice, neither algorithm is implemented in this way. In practical systems the head is moved as far as the last request in each direction. As soon as there are no requests in the current direction, the head movement is reversed. These variations [2] of SCAN and C-SCAN scheduling are called Look and C-look scheduling respectively (Figures 2.5 & 2.6).

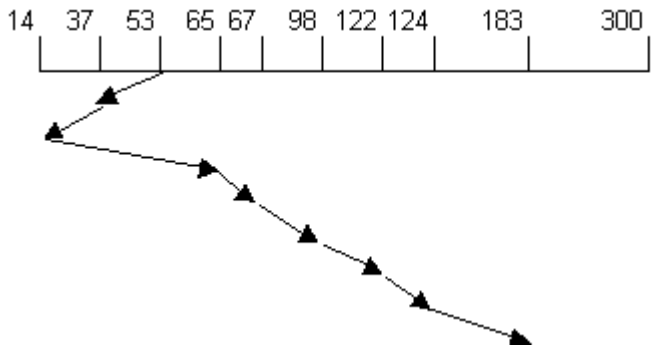


Figure 2.5 Look Disk Scheduling

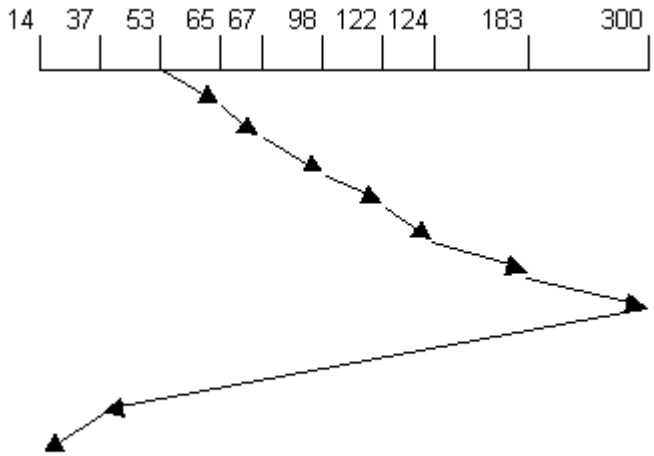


Figure 2.6 C-Look Disk Scheduling

Chapter 3
Conceptual Model of Software Development

3.1 General

A system for disk scheduling algorithms has been implemented in C language for scheduling the algorithms like Scan Upward (SCAN/U), Scan Downward (SCAN /D), Circular Scan Upward (C-SCAN/U) and Circular Scan Downward (C-SCAN/D). It consists of four modules namely SCAN/U, SCAN/D, C-SCAN/U, C-SCAN/D. When an algorithm is selected, it calls that module for execution. There are certain supporting functions/header files for different purposes. Header file comparat includes all functions, which are used for reordering the original list as per the condition of SCAN/U, SCAN/D, C-SCAN/U and C-SCAN/D algorithms. Header file graph includes all functions and programs for the display of different graphs. Header file display is used for the display of queue and related graphics. Header file show is used for the display of all other graphics in the program. The 'total Head movement' can judge the performance of each algorithm in a particular case.

3.2 Major Modules

When this program is executed, it presents the four options to the user, namely:

1- SCAN UPWARD
2- SCAN DOWNWARD
3- C-SCAN UPWARD
4- C-SCAN DOWNWARD
5- EXIT

Up/down arrow keys can select any algorithm for implementation. Selection of an algorithm calls another function which shows choices namely as

1-Run Default
2- Enter New values
3-Go to Main Menu
4-Exit

3.2.1 Procedure

When the program is executed, Selection of an algorithm for execution gives option of running default set of values (already fed set of requests), or option of entering new values. Selection of running default set & values which is as under:-

98, 183, 37, 122, 14, 124, 65 and 67

The default set of values is re-arranged according to selected algorithm for calculation of head movements. The re-arranged queue will be displayed and calculation is carried out. After calculation the number of head movements involved to process the queue is displayed. Graphical representation of the head movements is also shown. If the option of Entering new vales is selected the data can be written to the file by calling function wfile. Now the newly entered values are re-arranged according to selected algorithm for calculation of head movements. After calculation the number of head movements involved to process the queue is displayed. Graphical representation of the head movements is also shown. Selection of choice three takes you back to main menu screen.

3.2.2 Show File

This is header file which is used for drawing queue and other related graphics. This header file contains functions show queue, remove1, remove2, remove3, remove4, remove5, remove6 etc. All these functions are used for drawing queue. There are also certain other functions in this header file like showmain, clearmain, showchoices,

clearchoices etc. The showmain function is used for the graphical display of the options available in the simulator. Calling function clearmain clears the screen from these options. Show choices is used to display options available in fcfs, sstf, scan or c-scan algorithms while clear choices is called to clear the screen from these options.

3.2.3 Comparat File

This is also a developer's defined header file which consists of the functions compare 1, compare 2, compare 3. Function compare l in is used in SCAN to find the upward and downward tracks in the requests received. Functions compare 2 and compare 3 are used to rearranging the upward and downward requests respectively in case of SCAN/U, SCAN/D. Function compare 4 is used to find the upward and downward tracks in the requests received. Function compare5 is used to rearrange downward request in C-SCAN algorithm.

3.2.4 Graph File

This header file is used for drawing individuals as well as comparative graphs for all algorithms.

3.2.5 Summary

In this chapter we have briefly discussed the main modules and important header files of the developed software. It was noted that the major modules are scan/upward, scan/downward, c-scan/upward and c-scan/downward newname, and wfile. The header files discussed here are print, show, comparat and graph.

Chapter 4

Results and Evaluation Of Disk Scheduling Algorithms

4. I General

Disk drives are the major secondary storage I/O devices. Request for disks I/O are generated by the file system and by the virtual memory system[2]. Each request specifies the address on the disk to be referenced in the form of logical block number. Disk-scheduling algorithms can improve the effective bandwidth, the average response time and the variance in response time. Algorithms such as SSTF, SACN, C-SCAN, LOOK and C-LOOK make the improvements by reordering the disk queue to decrease the total seek time.

4.2 Performance Parameters of an Algorithm

Time taken by the head movement over the number of tracks is roughly the linear function of time i.e., if the head swings over more number of tracks the time taken will be more. So the data collected relating to head swings can be genuinely interpreted as the time taken in the head movement. Performance of an algorithm depends on following parameters:-

- Initial Head Position
- Direction of Head Movement
- Pattern of the Requested Queue

4.2.1 Initial Head Position

The Initial Head position will normally be the last request track of the preceding serviced queue. It is possible to control the position of head by positioning it to a preset position after servicing the last request queue. The preset initial head position will have it merits and demerits in case of patterns of different nature. The preset initial head position will give better performance in case of uniformly scattered request queue.

4.2.2 Direction of Head Movement

Direction of Head Movement is not applicable in all algorithms. It is applicable in case of LOOK, C-LOOK, SCAN and C-SCAN algorithms. The direction may be upward

or it may be towards downward direction. Selection of head movement direction has tremendously affect on the performance of algorithm but it is more influenced by the pattern of request queue.

4.2.3 Pattern of the Requested Queue

Pattern of the requested queue for disk service is greatly influenced by the file-allocation method [4]. A program reading a contiguously allocated file will generate several requests that are close together on the disk, resulting in limited head movement. A linked or indexed file, on the other hand, may include blocks that are widely scattered on the disk, resulting in better disk-space utilization at the expense of head movement.

The location of directories and index blocks is also important. As every file is to be opened before use, which requires searching the directory structure. For instance, if a directory entry is in the first sector and file's data in the last sector, then the disk head needs to move the entire width of the disk. If the directory entry is more toward the middle, the head will have to move at most one-half the width

In uniformly scattered request queue each request is likely to get fair deal and over all performance will be improved. Pattern of the request queue is unpredictable, but it can be made more uniform by selecting best file allocation method or by periodically de-fragmenting the disk by any de-fragmentation utility

As a result of these considerations, it is clear that the disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if required.

4.3 Test Samples

Few test samples have been selected in such manner that each sample consists of different type of pattern, number of tracks and initial head positions (IHP). Values of each sample are fed in through the option of ‘Enter New Values’ in the form of a separate

file. Each sample will run on the developed simulator to obtain 'total head movement' for each algorithm.

4.3.1 Sample-1 A uniformly scattered pattern of 24 tracks request have been tested for 100, 150, and 200 IHP.

67, 283, 37, 122, 224, 15, 167, 98, 212, 133, 254, 275, 5, 33, 66, 225, 56, 237, 288, 258, 128, 156, 144, 162

Scan/Up

Reordered Queue For IHP=100

122, 124, 128, 133, 144, 156, 162, 167, 212, 225, 237, 254, 258, 275, 283, 288, **300,** 98, 67, 66, 56, 37, 33, 15, 3

Head Movement Pattern for IHP=100

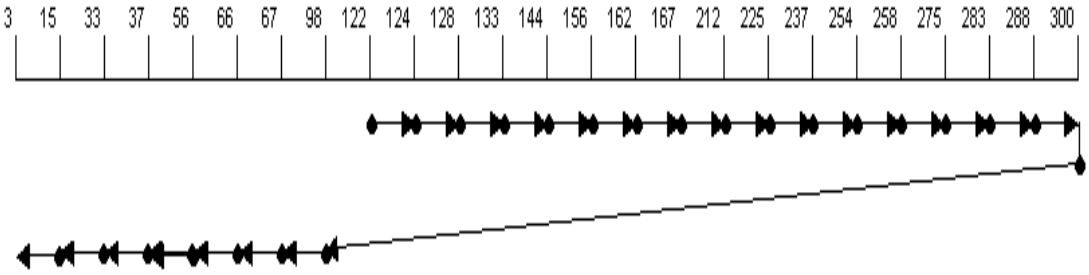


Figure 4.1 Total Head Movement=497

Reordered Queue For IHP=150

156, 162, 167, 212, 225, 237, 254, 258, 275, 283, 288, **300,** 144, 133, 128, 124, 122, 98, 67, 66, 56, 37, 33, 15, 3

Head Movement Pattern for IHP=150

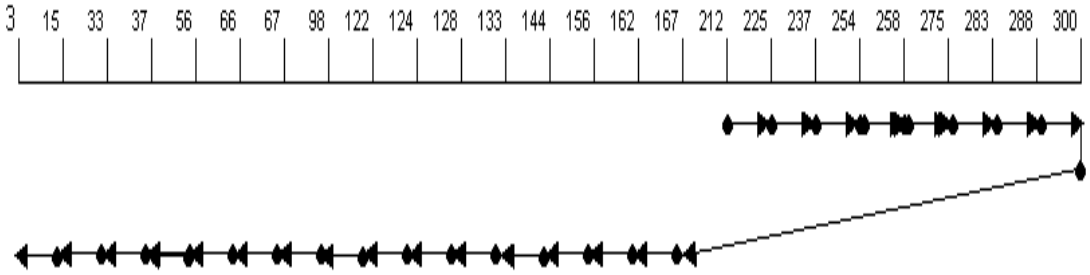


Figure 4.2 Total Head Movement=447

Reordered Queue For IHP=200

212, 225, 237, 254, 258, 275, 283, 288***, 300***,167, 162, ***156,144, 133, 128, 124, 122, 98, 67, 66, 56, 37, 33, 15, 3***

Head Movement Pattern for IHP=200

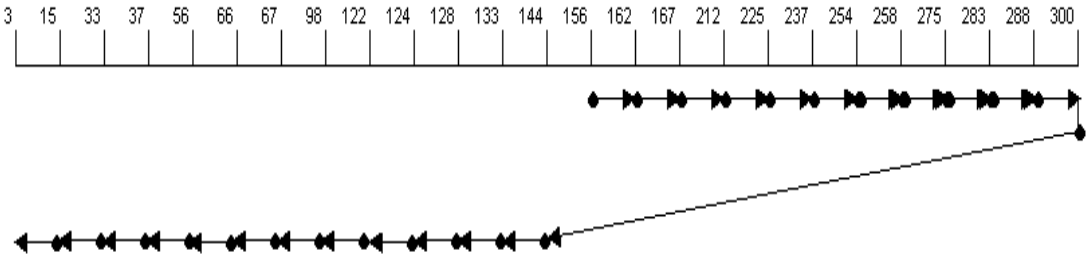


Figure 4.3 Total Head Movement=397

Scan/Down

Reordered Queue For IHP=100

98, 67, 66, 56, 37, 33, 15, 3, **0**, 122, 124, 128, 133, 144, 156, 162, 167, 212, 225, 237, 254, 258, 275, 283, 288

Head Movement Pattern for IHP=100

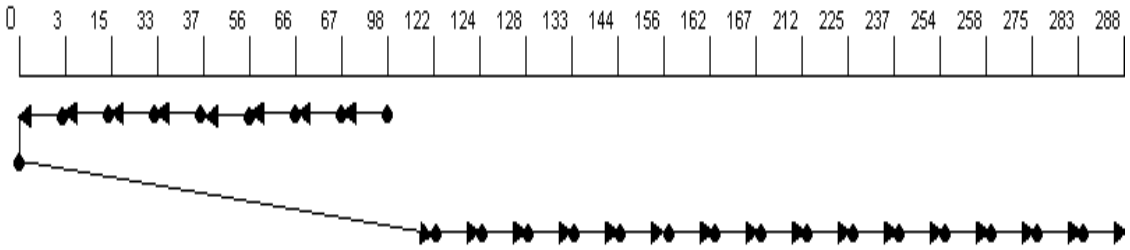


Figure 4.4 Total Head Movement=397

Reordered Queue For IHP=150

144, 133, 128, 124, 122, 98, 67, 66, 56, 37, 33, 15, 3,**0**, 156, 162, 167, 212, 225, 237, 254, 258, 275, 283, 288

Head Movement Pattern for IHP=150

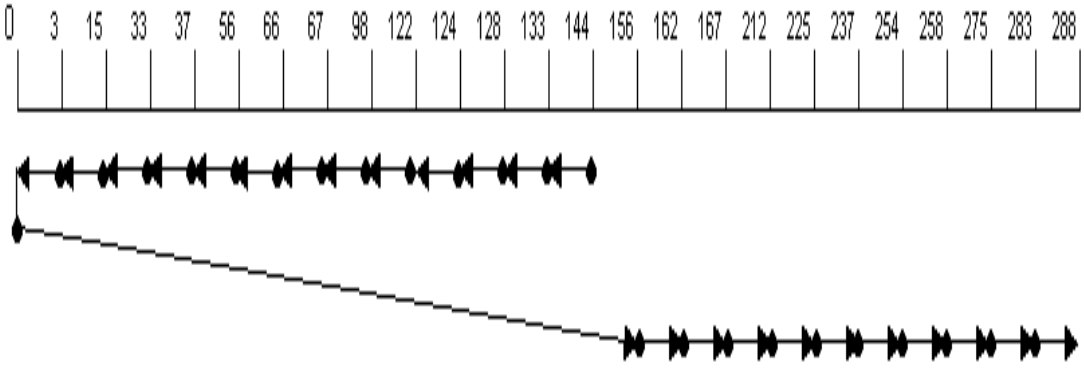


Figure 4.5 Total Head Movement=438

Reordered Queue For IHP=200

167, 162, 156,144, 133, 128, 124, 122, 98, 67, 66, 56, 37, 33, 15, 3***, 0,*** 212, 225, 237, 254, 258, 275, 283, 288

Head Movement Pattern for IHP=200

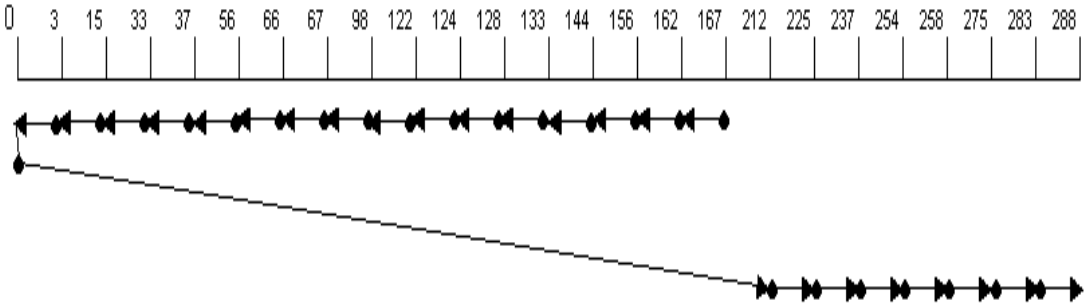


Figure 4.6 Total Head Movement=488

C-Scan/Up

Reordered Queue For IHP=100

122, 124, 128, 133, 144, 156, 162, 167, 212, 225, 237, 254,

258, 275, 283, 288***, 300, 0***, 3, 15, 33, 37, 56, 66, 67, 98

Head Movement Pattern for IHP=100

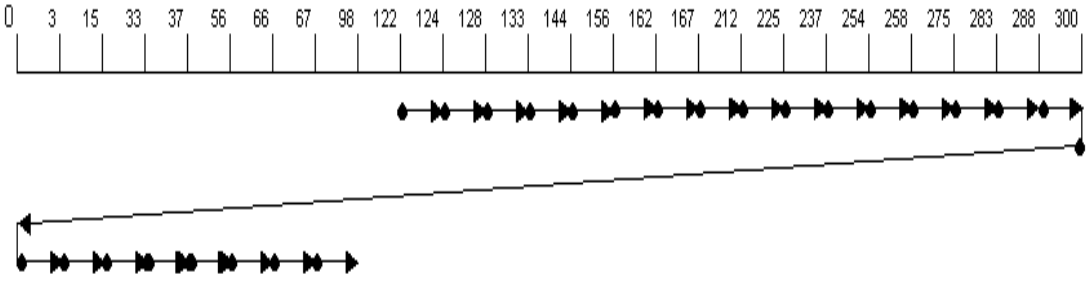


Figure 4.7 Total Head Movement=598

Reordered Queue For IHP=150

156, 162, 167, 212, 225, 237, 254, 258, 275, 283, 288, ***300, 0***, 3, 15, 33, 37, 56, 66, 67, 98, 122, 124, 128, 133, 144

Head Movement Pattern for IHP=150

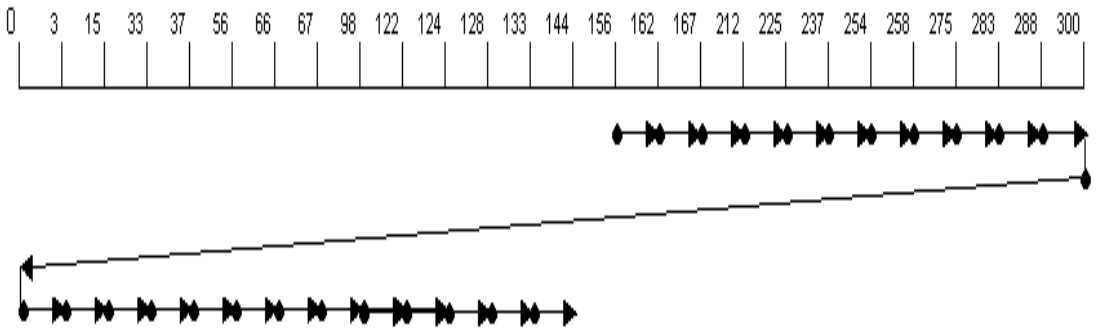


Figure 4.8 Total Head Movement=438

Reordered Queue For IHP=200

212, 225, 237, 254, 258, 275, 283, 288, **300*, 0,*** 3, 15, *33*, 37, 56, 66, 67, 98, 122, 124, 128, 133, 144, 156, 162, 167

Head Movement Pattern for IHP=200

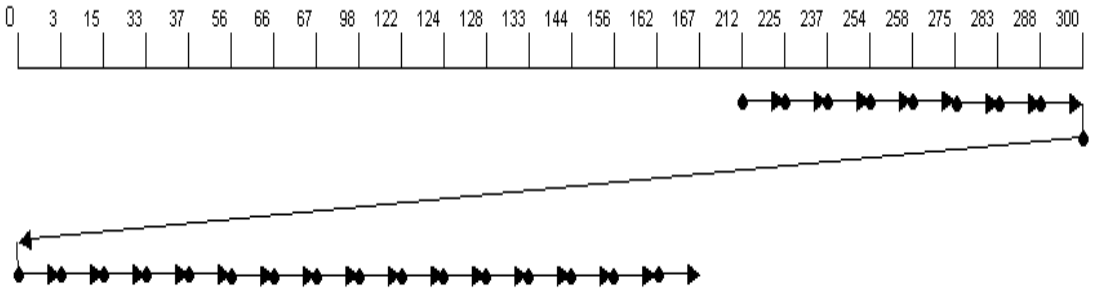


Figure 4.9 Total Head Movement=567

C-Scan/Down

Reordered Queue For IHP=100

98, 67, 66, 56, 37, 33, 15, 3, ***0, 300***, 288, 283, 275, 258, 254,237, 225, 212, 167, 162, 156, 144, 133, 128, 124, 122

Head Movement Pattern for IHP=100

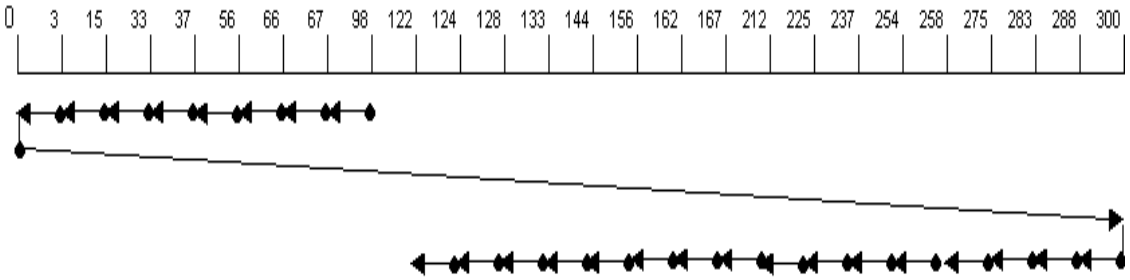


Figure 4.10 Total Head Movement=578

Reordered Queue For IHP=150

144, 133, 128, 124, 122, 98, 67, 66, 56, 37, 33, 15, 3, **0, 300,** 288, 283, 275, 258, 254,237, 225, 212, 167, 162, 156

Head Movement Pattern for IHP=150

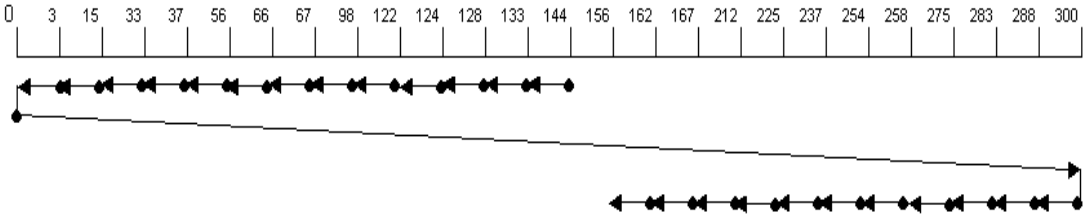


Figure 4.11 Total Head Movement=604

Reordered Queue For IHP=200

167, 162, 156, 144, 133, 128, 124, 122, 98, 67, 66, 56, 37, 33, 15, 3, **0, 300,** 288, 283, 275, 258, 254,237, 225, 212

Head Movement Pattern for IHP=200

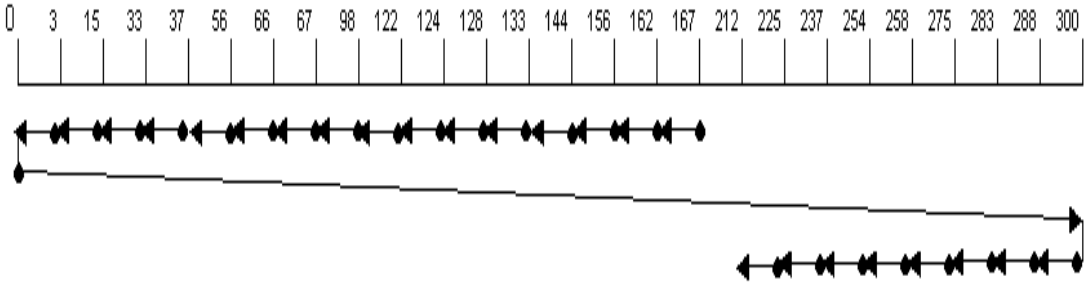


Figure 4.12 Total Head Movement=604

Summary of Results

Algorithm	*IHP100*	*IHP150*	*IHP200*
SCAN/U	*497*	*447*	***397***
SCAN/D	***388***	***438***	*488*
C-SCAN/U	*598*	*594*	*567*
C-SCAN/D	*578*	*604*	*588*

Table 4.1 Results of Sample-1

4.3.2 Sample-2 A uniformly scattered pattern of 24 tracks request will be for 100, 150, and 200 IHP.

211, 219, 222, 224, 242, 267, 288, 299, 88, 76, 111,

132, 145, 165, 160, 167, 188 198, 66, 56, 44,14, 36, 2

Scan/Up

Reordered Queue For IHP=100

111, 132, 145, 160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299**, 300,** 88, 76, 66, 56, 44, 36, 14, 2

Head Movement Pattern for IHP=100

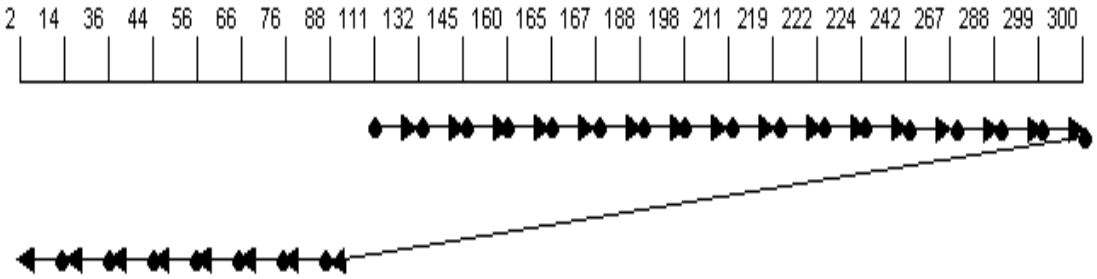


Figure 4.13 Total Head movement=500

Reordered Queue For IHP=150

160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299***, 300,*** 145, 132, 111, 88, 76, 66, 56, 44, 36, 14, 2

Head Movement Pattern for IHP=150

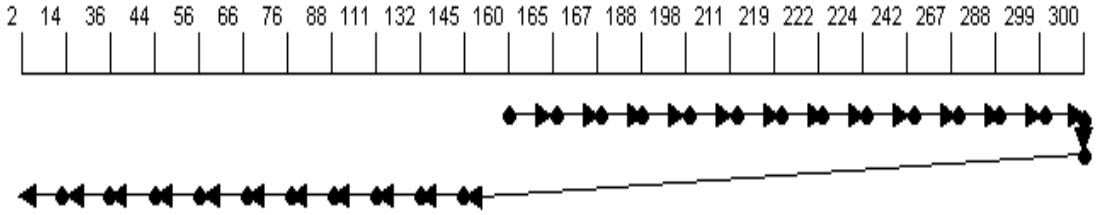


Figure 4.14 Total Head movement=448

Reordered Queue For IHP=200

211, 219, 222, 224, 242, 267, 288, 299, ***300***, 198, 188, 167, 165, 160, 145, 132, 111, 88, 76, 66, 56, 44, 36, 14, 2

Head Movement Pattern for IHP=200

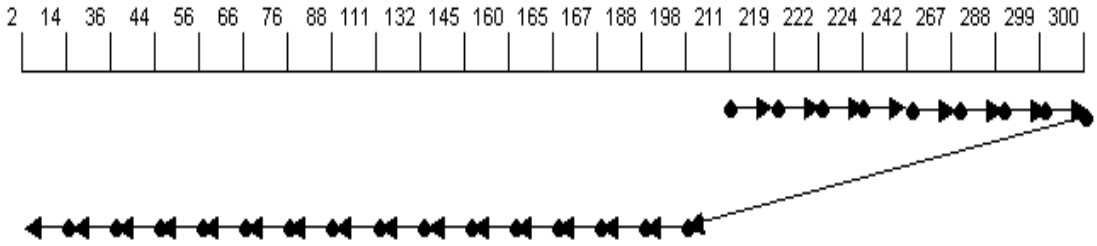


Figure 4.15 Total Head movement=398

Scan/Down

Reordered Queue For IHP=100

88, 76, 66, 56, 44, 36, 14, 2, **0,** 111, 132, 145, 160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299

Head Movement Pattern for IHP=100

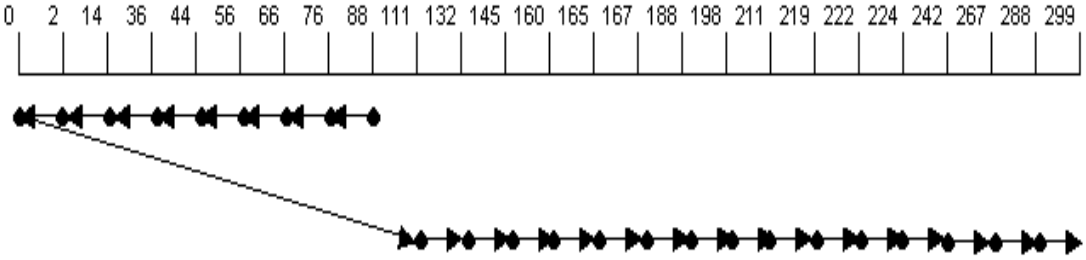


Figure 4.16 Total Head movement=399

Reordered Queue For IHP=150

145, 132, 111, 88, 76, 66, 56, 44, 36, 14, 2, **0,**160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299

Head Movement Pattern for IHP=150

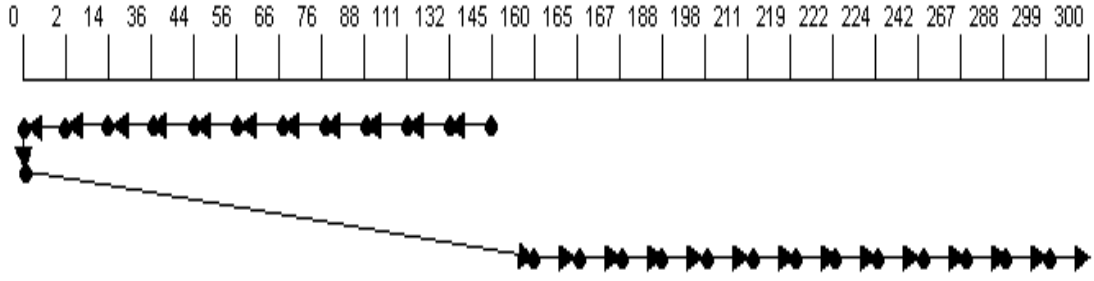


Figure 4.17 Total Head movement=449

Reordered Queue For IHP=200

198, 188, 167, 165, 160, 145, 132, 111, 88, 76, 66, 56, 44,36, 14, 2, **0**,211, 219, 222, 224, 242, 267, 288, 299

Head Movement Pattern for IHP=200

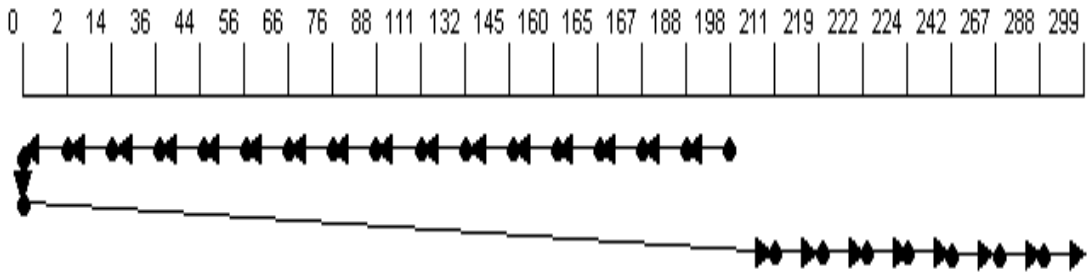


Figure 4.18 Total Head movement=499

C-Scan/Up

Reordered Queue For IHP=100

111, 132, 145, 160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299***, 300, 0,*** 88, 76, 66, 56, 44, 36, 14, 2

Head Movement Pattern for IHP=100

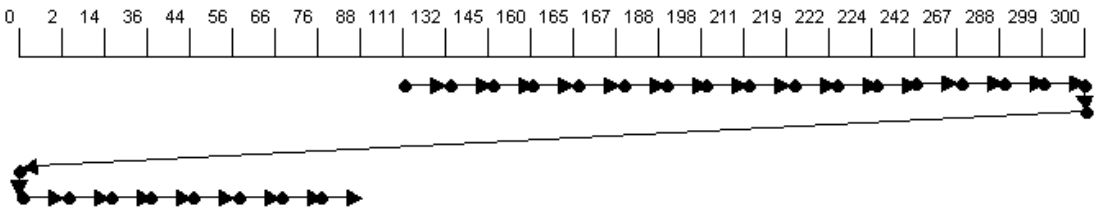


Figure 4.19 Total Head movement=588

Reordered Queue For IHP=150

160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299**, 300, 0,** 145, 132, 111, **,** 88, 76, 66, 56, 44, 36, 14, 2

Head Movement Pattern for IHP=150

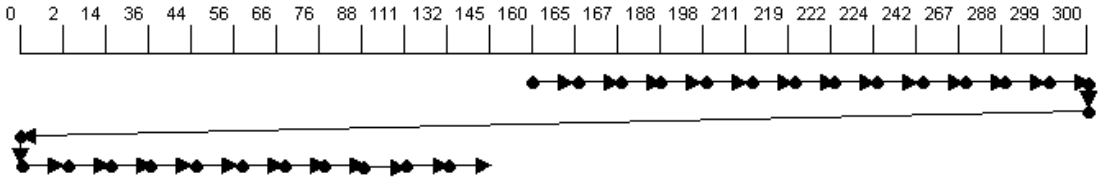


Figure 4.20 Total Head movement=595

Reordered Queue For IHP=200

211, 219, 222, 224, 242, 267, 288, 299**, 300**, **0**,198, 188, 167, 165, 160**,** 145, 132, 111, 88, 76, 66, 56, 44, 36, 14, 2

Head Movement Pattern for IHP=200

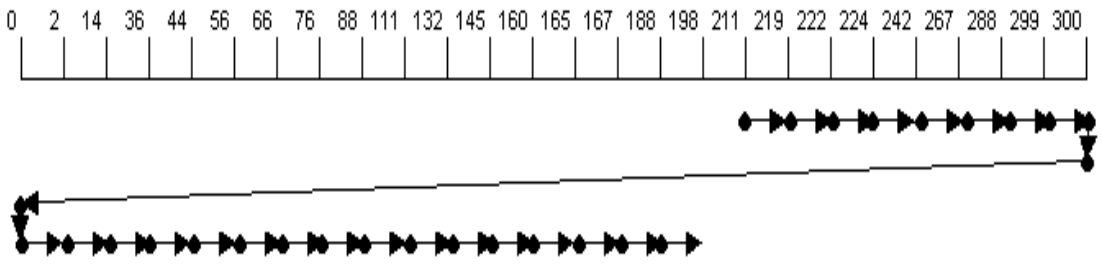


Figure 4.21 Total Head movement=612

C-Scan/Down

Reordered Queue For IHP=100

88, 76, 66, 56, 44, 36, 14, 2, **0, 300,** 111, 132, 145, 160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299

Head Movement Pattern for IHP=100

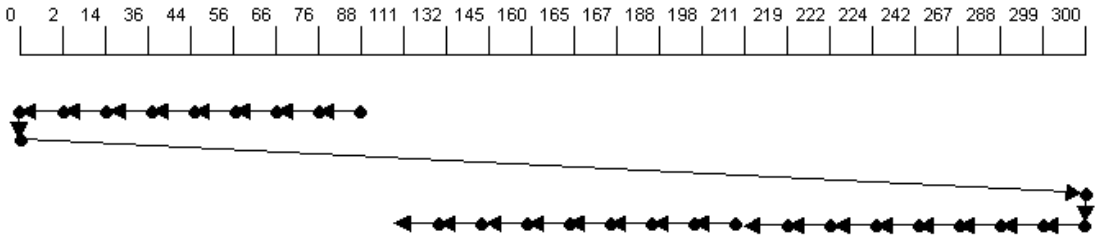


Figure 4.22 Total Head movement=589

Reordered Queue For IHP=150

145, 132, 111, 88, 76, 66, 56, 44, 36, 14, 2, **0, 300,** 160, 165, 167, 188, 198, 211, 219, 222, 224, 242, 267, 288, 299

Head Movement Pattern for IHP=150

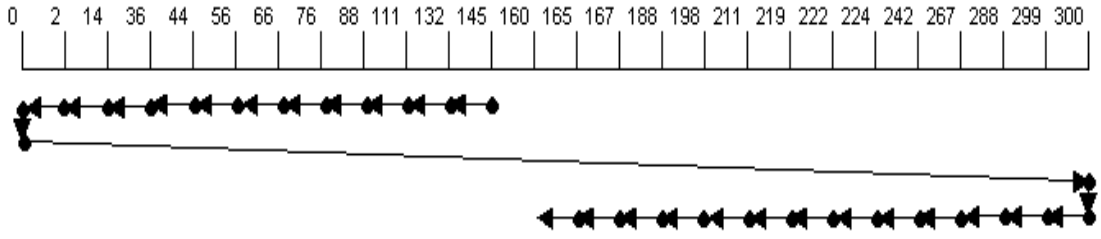


Figure 4.23 Total Head movement=599

Reordered Queue For IHP=200

198, 188, 167, 165, 160, 145, 132, 111, 88, 76, 66, 56, 44,36, 14, 2, **0**, **300,** 211, 219, 222, 224, 242, 267, 288, 299

Head Movement Pattern for IHP=200

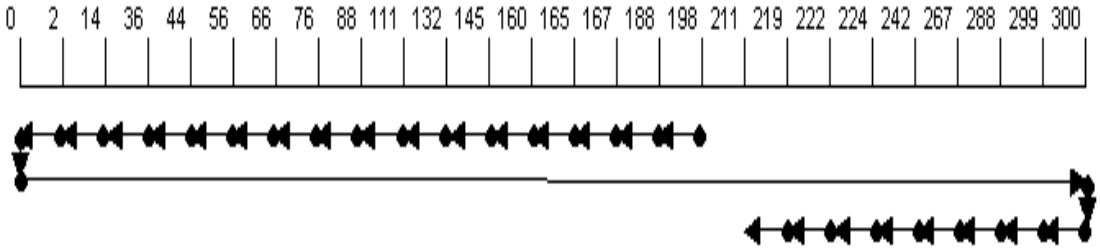


Figure 4.24 Total Head movement=623

Summary of Results Sample-2

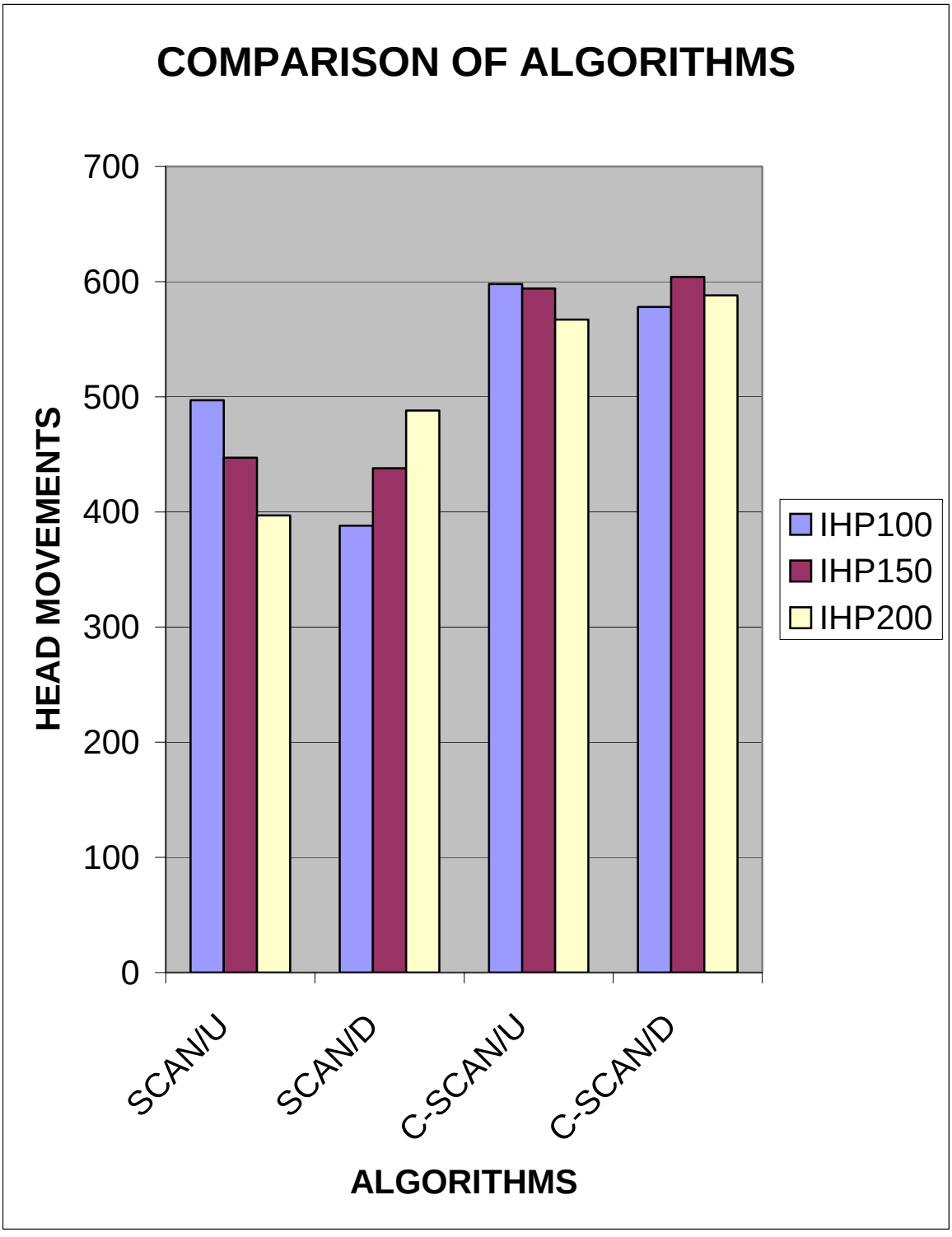
Algorithm	*IHP100*	*IHP150*	*IHP200*
SCAN/U	*500*	***448***	***398***
SCAN/D	***399***	*499*	*499*
C-SCAN/U	*588*	*595*	*612*
C-SCAN/D	*589*	*599*	*623*

Table 4.2 Results of Sample-2

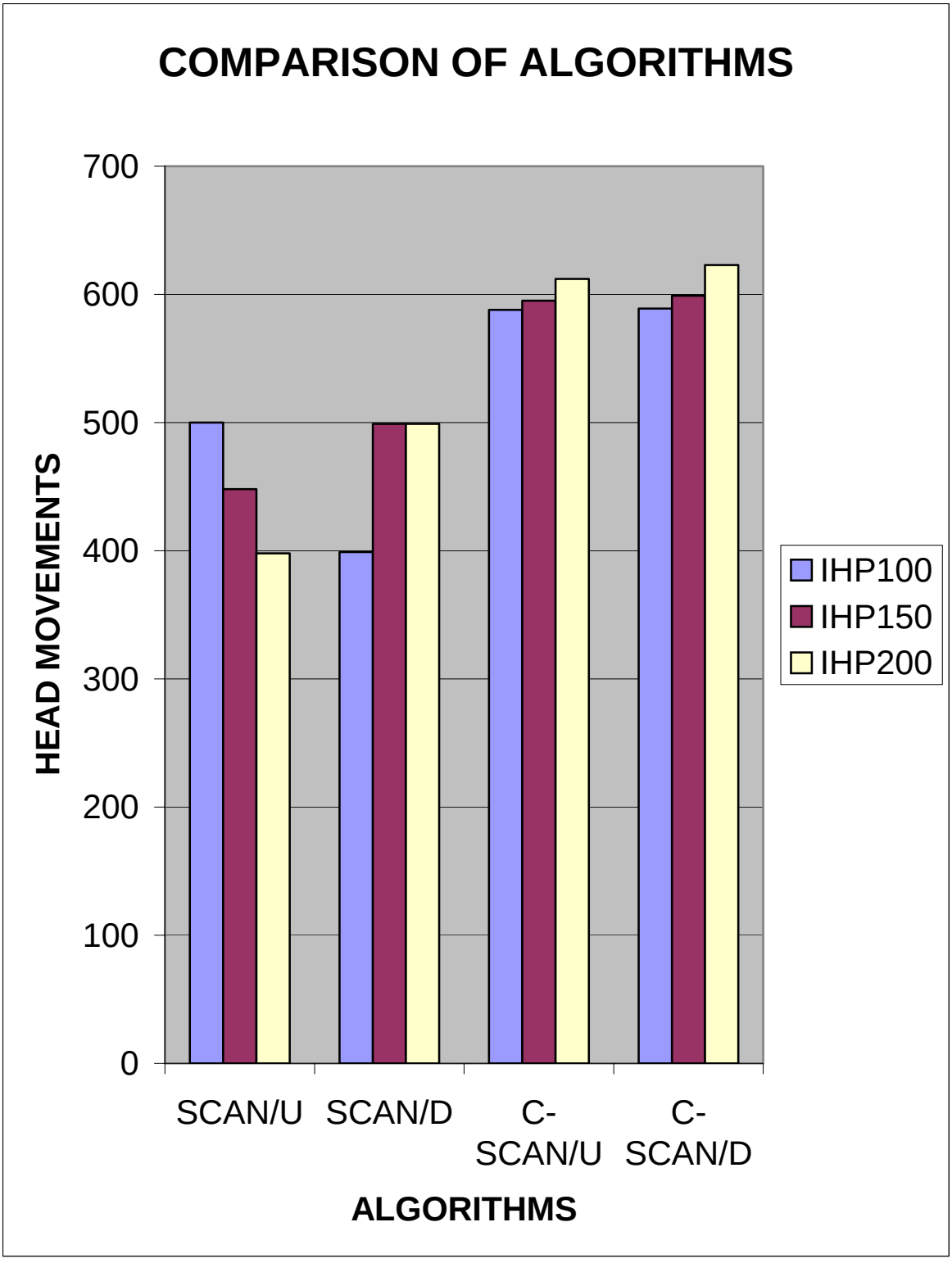
Summary of Results Sample-1 and Sample-2

Algorithm	***Sample-1***			***Sample-2***		
	IHP100	*IHP150*	*IHP200*	*IHP100*	*IHP150*	*IHP200*
SCAN/U	*497*	*447*	***397***	*500*	***448***	***398***
SCAN/D	***388***	***438***	*488*	***399***	*499*	*499*
C-SCAN/U	*598*	***594***	***567***	***588***	***595***	*612*
C-SCAN/D	***578***	*604*	*588*	***589***	*599*	*623*

Table 4.3 Results Sample-1 and Sample-2



Graph 4.1 Sample-1



Graph 4.2 Sample-2

Sample-1 and sapmle-2 are uniformly scattered request patterns, which have been tested for three different Initial Head Positions. The first IHP 100 is slightly towards lower side of the Disk, IHP 200 is slightly upper side of the disk and IHP 150 is centrally placed on the entire rang of track.

In case of IHP 150 the results shown in Table 4.1 and Graph 4.1 are 447 and 448 for SCAN/U, 438 and 499 for SCAN/D, 594 and 595 for C-SCAN/U, 604a and 599 for C-SCAN/D respectively for sample-1 and sample-2. Both the test samples irrespective of the direction of head movement have produced almost similar results for all the four algorithms.

With IHP 100 the results are 497 and 500 for SCAN/U, 338 and 399 for SCAN/D, 598 and 588 for C-SCAN/U, 578 and 589 for C-SCAN/D. It is clear that there is a remarkable decrease from 438 to 338 and from 499 to 399 in head movements for SCAN/D algorithm as IHP is more towards lower side of the hard disk.

It can be stated conclusively that SCAN has performed better then C-SCAN irrespective of the position of head movement and direction of head movement as C-SCAN has a additional head movement swing from last track to first track or from first to last track.

4.3.3 Sample-3 The sample is consisting of 12 tracks request with is uniformly scattered. The sample will be tested for two IHPs, which are 60 and 260.

23, 35, 75, 88, 112, 135, 187, 212, 235,267, 288, 295

Scan/Up

Reordered Queue For IHP=60

75, 88, 112, 135, 187, 212, 235,267, 288, 295, **300**, 35, 23

Head Movement Pattern for IHP=60

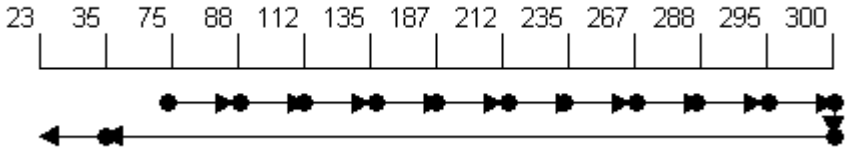


Figure 4.25 Total Head Movements=517

Reordered Queue For IHP=260

267, 288, 295, **300**, 235, 212, 187, 135, 112, 88, 75, 35, 23

Head Movement Pattern for IHP=260

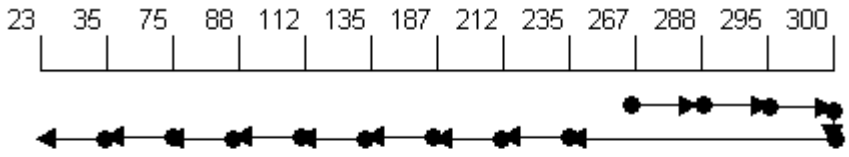


Figure 4.26 Total Head Movements=317

Scan/Down

Reordered Queue For IHP=60

35, 23**,0**, 75, 88, 112, 135, 187, 212, 235,267, 288, 295

Head Movement Pattern for IHP=60

Figure 4.27 Total Head Movements=355

Reordered Queue For IHP=260

235, 212, 187, 135, 112, 88, 75, 35, 23, **0,** 267, 288, 295

Head Movement Pattern for IHP=260

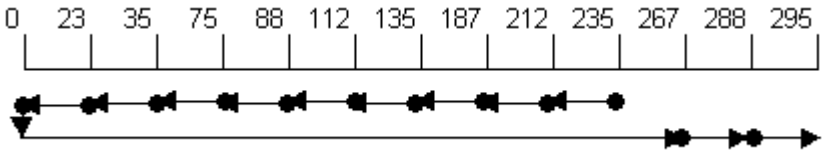


Figure 4.28 Total Head Movements=555

C-Scan/Up

Reordered Queue For IHP=60

75, 88, 112, 135, 187, 212, 235,267, 288, 295, **300**, **0**, 23, 35

Head Movement Pattern for IHP=60

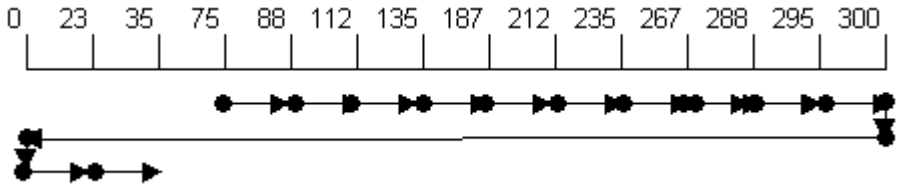


Figure 4.29 Total Head Movements=575

Reordered Queue For IHP=260

267, 288, 295, **300**, **0,** 235, 212, 187, 135, 112, 88, 75, 35, 23

Head Movement Pattern for IHP=260

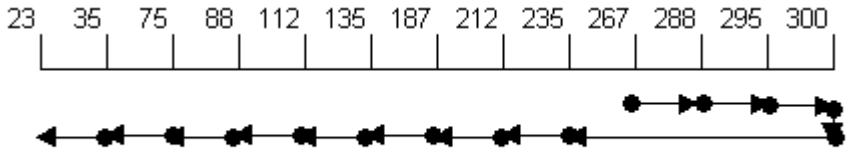


Figure 4.30 Total Head Movements=575

C-Scan/Down

Reordered Queue For IHP=60

35, 23**,0**, **300**, 75, 88, 112, 135, 187, 212, 235,267, 288, 295

Head Movement Pattern for IHP=60

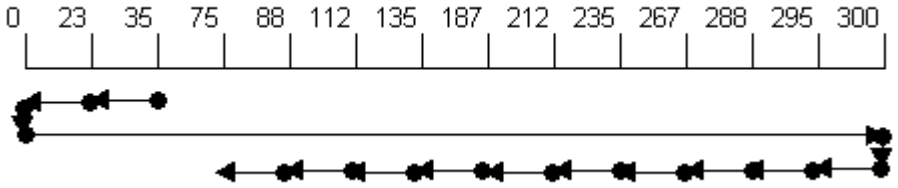


Figure 4.31 Total Head Movements=575

Reordered Queue For IHP=260

235, 212, 187, 135, 112, 88, 75, 35, 23, **0, 300,** 267, 288, 295

Head Movement Pattern for IHP=260

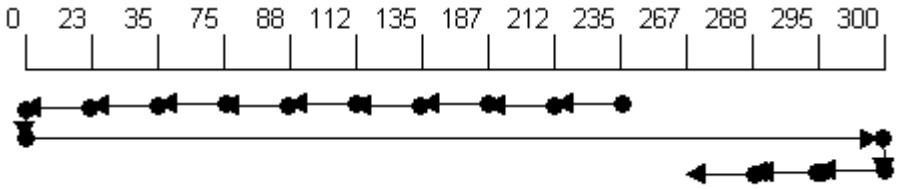


Figure 4.32 Total Head Movements=593

Summary of results

Algorithm	***IHP60***	***IHP260***
SCAN/U	517	***317***
SCAN/D	***355***	555
C-SCAN/U	575	575
C-SCAN/D	575	593

Table 4.4 Results of Sample-3

4.3.4 Sample-4 The sample is consisting of 12 tracks request with is uniformly scattered. The sample will be tested for two IHPs, which are 60 and 260.

12, 46, 66, 78, 108, 128, 167, 208, 224,254, 282, 290

Scan/Up

Reordered Queue For IHP=60

66, 78, 108, 128, 167, 208, 224,254, 282, 290,**300,** 46, 12

Head Movement Pattern for IHP=60

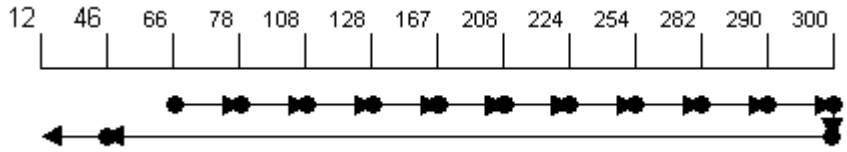


Figure 4.33 Total Head Movements=528

Reordered Queue For IHP=260

282, 290,**300,** 254, 224, 208, 167, 128, 108, 78, 66, 46, 12

Head Movement Pattern for IHP=260

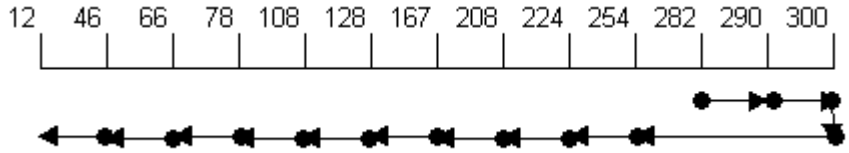


Figure 4.34 Total Head Movements=328

Scan/Down

Reordered Queue For IHP=60

46, 12, **0**, 66, 78, 108, 128, 167, 208, 224,254, 282, 290

Head Movement Pattern for IHP=60

Figure 4.35 Total Head Movements=350

Reordered Queue For IHP=260

254, 224, 208, 167, 128, 108, 78, 66, 46, 12, **0,** 282, 290

Head Movement Pattern for IHP=260

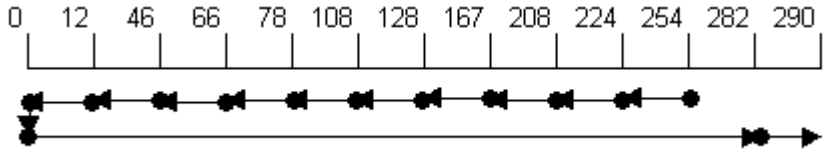


Figure 4.36 Total Head Movements=550

C-Scan/Up

Reordered Queue For IHP=60

66, 78, 108, 128, 167, 208, 224,254, 282, 290,**300, 0,** 12, 46

Head Movement Pattern for IHP=60

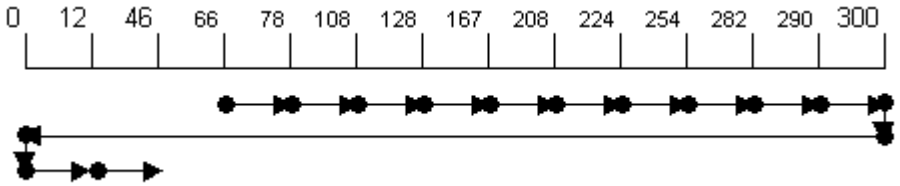


Figure 4.37 Total Head Movements=586

Reordered Queue For IHP=260

282, 290,**300, 0,** 12, 46, 66, 78, 108, 128, 167, 208, 224,254

Head Movement Pattern for IHP=260

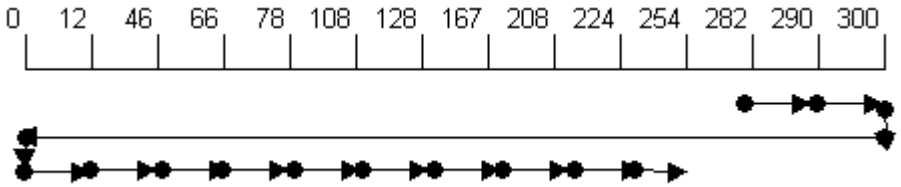


Figure 4.38 Total Head Movements=594

C-Scan/Down

Reordered Queue For IHP=60

46, 12, **0**, **300**, 66, 78, 108, 128, 167, 208, 224,254, 282, 290

Head Movement Pattern for IHP=60

Figure 4.39 Total Head Movements=594

Reordered Queue For IHP=260

254, 224, 208, 167, 128, 108, 78, 66, 46, 12, **0, 300,** 290, 282

Head Movement Pattern for IHP=260

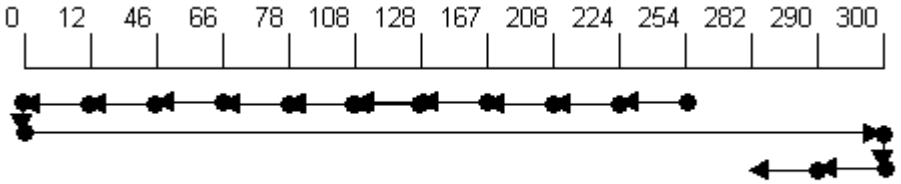


Figure 4.40 Total Head Movements=597

Different algorithms have been run using different values of IHP and their results have been obtained. For the sake of quick comparison they are summarized in table 4.5 for sample 4.

Summary of Results

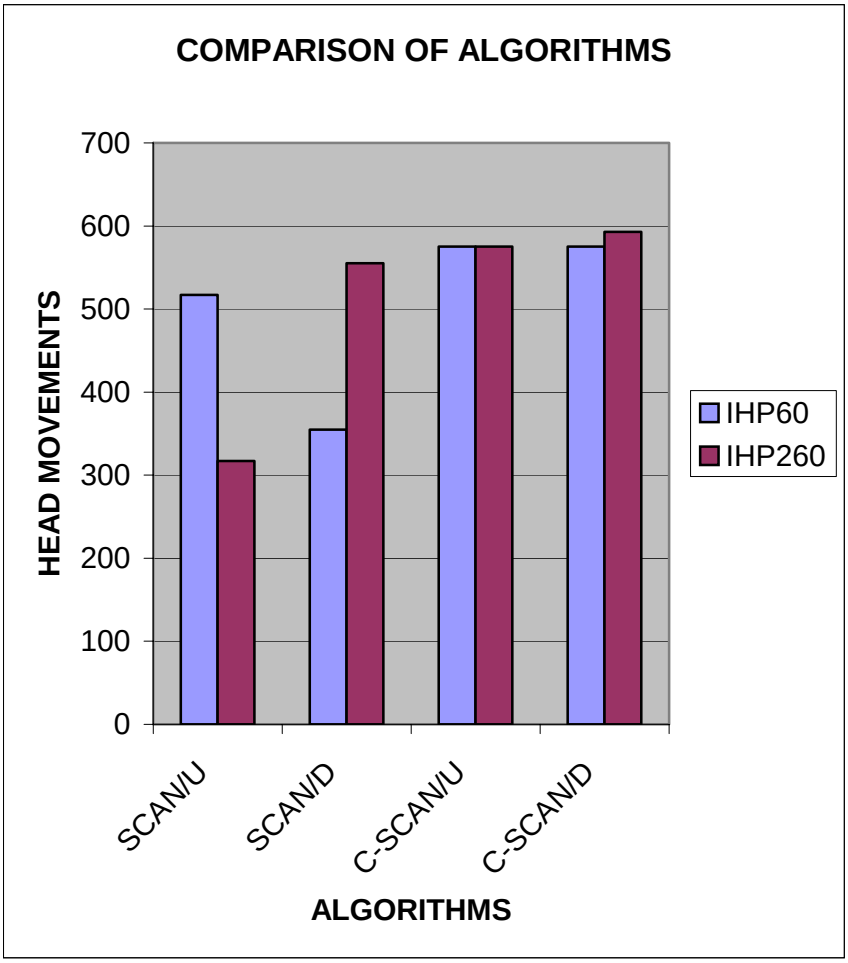
Algorithm	***IHP60***	***IHP260***
SCAN/U	528	***328***
SCAN/D	***350***	550
C-SCAN/U	586	594
C-SCAN/D	594	597

Table 4.5 Results of Sample-4

Summary of Results Sample-3 and Sample-4

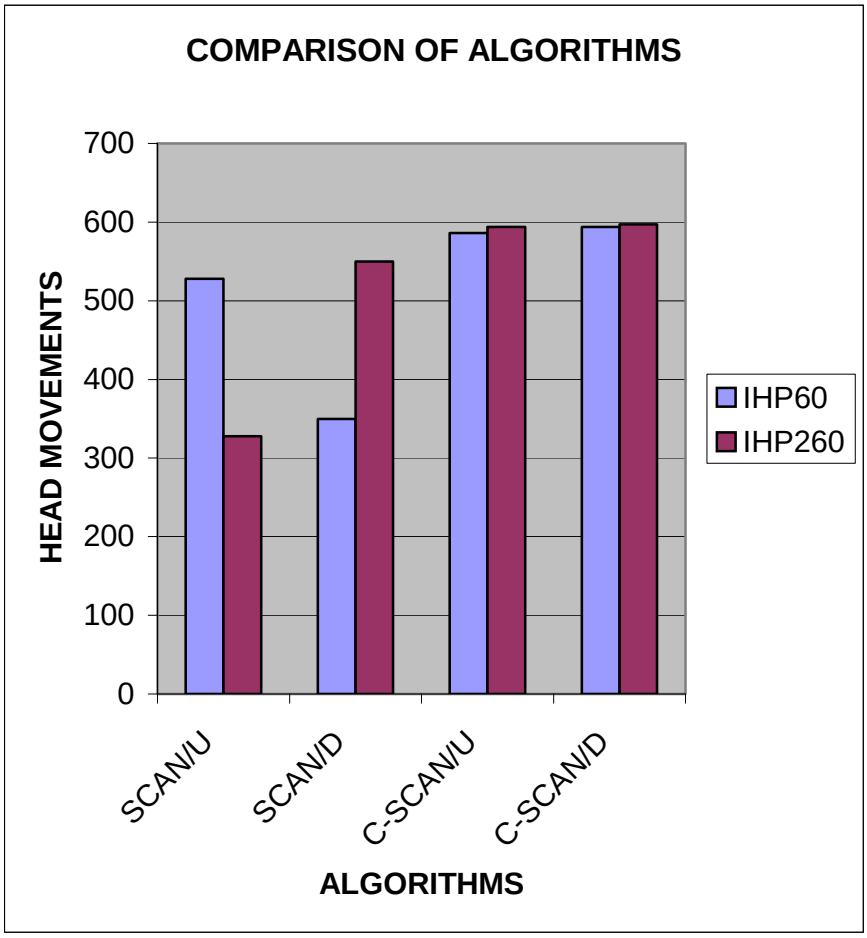
Algorithm	*Sample-3*		*Sample-4*	
	IHP60	*IHP260*	*IHP60*	*IHP260*
SCAN/U	517	***317***	528	***328***
SCAN/D	***355***	555	***350***	550
C-SCAN/U	**575**	**575**	***586***	594
C-SCAN/D	**575**	593	**594**	597

Table 4.6 Results of Sample-3&4



Graph 4.3 Sample-3

Sample-3 and sample-4 are uniformly scattered request sample, both have been tested for IHP 60 and IHP260.



Graph 4.4 Sample-4

In case of IHP 60 the results shown in Table 4.5 and Graph 4.3 & 4.4 are 517 and 528 for SCAN/U, 355 and 350 for SCAN/D, 575 and 586 for C-SCAN/U, 575 and 593 for C-SCAN/D respectively for sample-3 and sample-4. In case of sample-3 when the 10 out of 12 requested tracks are laying above the IHP but are spread over wider range of tracks such as from 75 to 295. The SCAN/U has resulted in 517 head movements as compared to 355 for SCAN/D because of a wild swing from 300 to 35(Figure 4.25 & Figure 4.27).

Similarly in case of sample-3 when the 10 out of 12 requested tracks are laying below the IHP 260 but are spread over wider range of tracks such as from 235 to 23. The SCAN/D has resulted in 555 head movements as compared to 317 for SCAN/U because of a wild swing from 0 to 267(Figure 4.26 & Figure 4.28).

The SCAN algorithm starts its head movement IHP goes to the end track according to preferred head movement direction and finishes its movement till the last requested track, therefore symmetry of requests before the end track (preferred end) and after the end track or number of requests before and after the end track determines the efficiency of the algorithm. If concentration of requested tracks is more after the end track (Figure 4.26 and Figure 4.27), the queue will be served with lesser number of head movements as compared to (Figure 4.25 and Figure 4.28), where concentration of requested tracks is more before the end track.

Both with IHP 60 and 260 the performance of SCAN/U, and SCAN/D, remained almost similar as C-SCAN deals the disk as circular and it does not serve any request during its wild swing from one end to the opposite end.

4.3.5 Sample-5 This sample has well scattered requests but only for lower half of the entire range of tracks. It will be tested for IHP 150.

5,12,19, 28, 37, 49, 61, 67, 69, 80,90,113,128, 137,145

Scan/Up

Reordered Queue For IHP=150

300, 145, 137, 128, 113, 90, 80, 69, 67, 61, 49, 37, 28, 12, 5

Head Movement Pattern for IHP=150

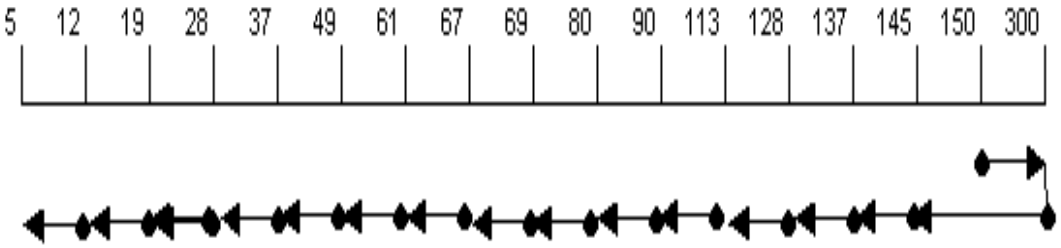


Figure 4.41 Total Head Movements=442

Scan/Down

Reordered Queue For IHP=150

145, 137, 128, 113, 90, 80, 69, 67, 61, 49, 37, 28, 12, 5,**0**

Head Movement Pattern for IHP=150

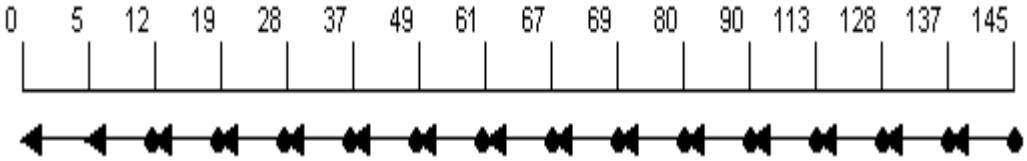


Figure 4.42 Total Head Movements=150

C-Scan/Up

Reordered Queue For IHP=150

300, 145, 137, 128, 113, 90, 80, 69, 67, 61, 49, 37, 28, 12, 5,**0**

Head Movement Pattern for IHP=150

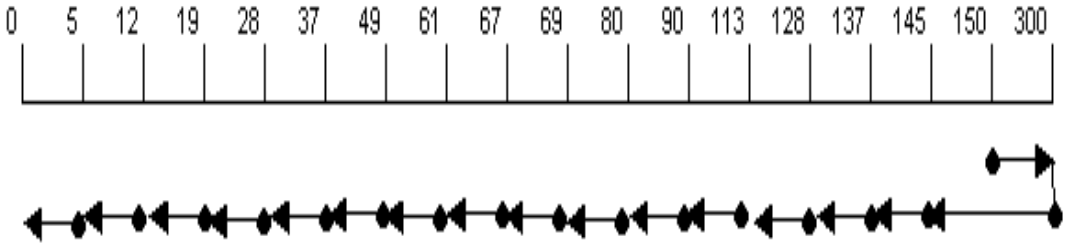


Figure 4.43 Total Head Movements=450

C-Scan/Down

Reordered Queue For IHP=150

145, 137, 128, 113, 90, 80, 69, 67, 61, 49, 37, 28, 12, 5, **0, 300**

Head Movement Pattern for IHP=150

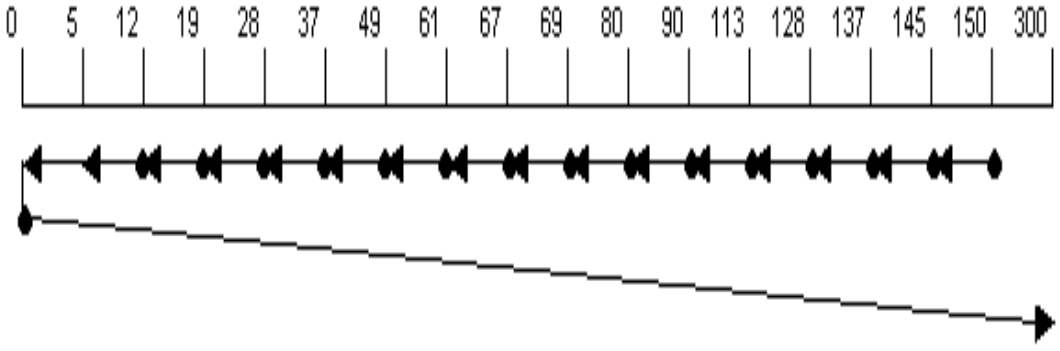
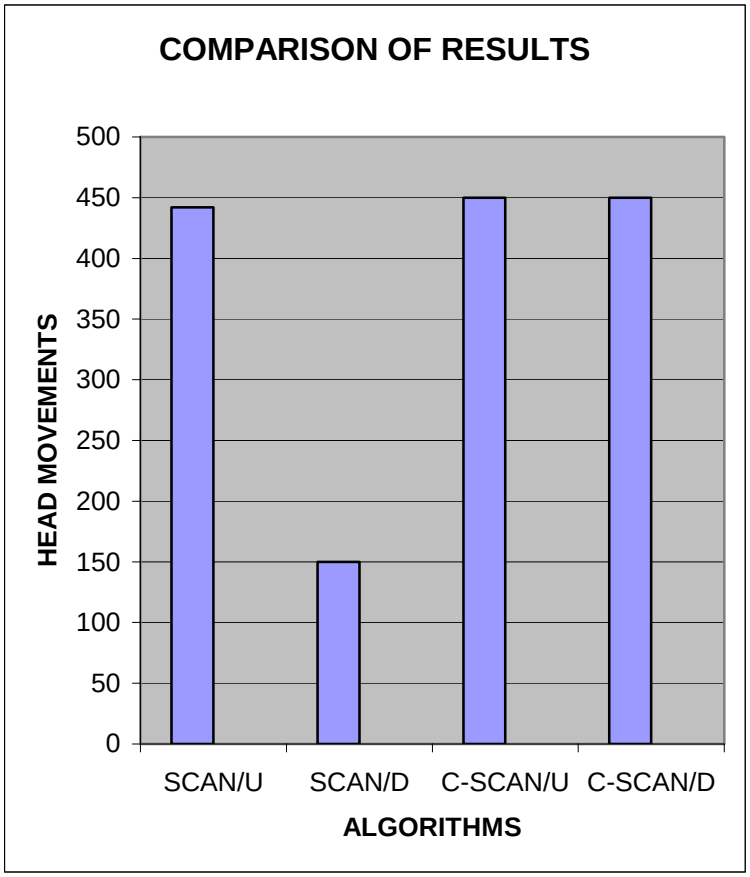


Figure 4.44 Total Head Movements=450

Summary of Results

Algorithm	*IHP150*
SCAN/U	442
SCAN/D	***150***
C-SCAN/U	450
C-SCAN/D	450

Table 4.7 Results of Sample-5



Graph4.5 Sample 5

Sample-5 has all the requests in the lower half of the disk and it has been tested for IHP 150 Table 4.7 and Graph 4.5 shows that SCAN/D has given the best results where as C-SCAN/U and C-SACN/D have resulted in equal number of head movements. It is clear that if the entire request is below the IHP then SCAN/D performs the best.

Chapter 5
Discussion and Comparison of Results

5.1 Introduction

In this chapter, we will compare Scan/Upward, Scan/Downward, C-Scan/Upward and C-Scan/Downward with FCFS (First Come First serve), SSTF (Shortest Seek Time First), Look/Upward, Look/Downward C-Look/Upward and C-Look/Downward algorithms. The simulated results of four test samples of FCFS, SSTF, Look/Upward, Look/Downward and C-look Obtained by [1] have been compared with the currently simulated results of Scan/Upward, Scan/Downward, C-Scan/Upward and C-Scan/Downward. Initially each algorithm is discussed in the light of results obtained from the four examples with the results of [1] and then a comparison of all algorithms is made.

5.2 Scan Upward

Scan Upward algorithm operates like SSTF except that it chooses the requests that results in the shortest seek distance in a *upward direction.* The read-write head starts movement in the upward direction, and moves till the last track of the disk servicing requests as it reaches each track., After reaching at the last track the direction of head movement is reversed and servicing continues till the availability of requests in this direction. The head continuously scans the disk from end to end.

Sample-1

Let us start with example-1, which consists of the following 8 tracks:

98, 183, 37, 122, 14, 124, 65, 67

Where track 98 is the first and track 67 is the last request received. If the initial head position is at track 53 and the head movement is upward then the reordered queue on the basis of Scan Upward algorithm becomes

65, 67, 98, 122, 124, 183, 300, 37, 14

Where track 65 will be first serviced and then track 67, 98, 122, 124, 183, 37 and finally track 14 will be serviced for a total head movement of 333 tracks. (Figure 5.1).

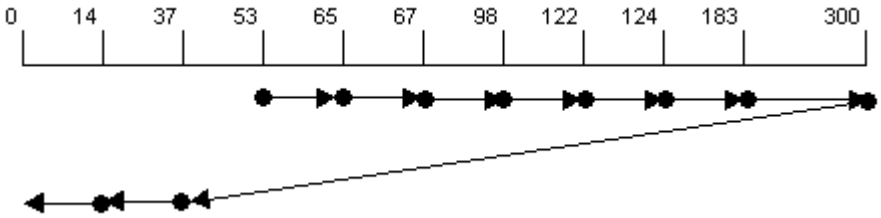


Figure 5.1 Sample-1 Scan Upward disk scheduling.

Sample-2

Let us consider example 2 which consists of the following 11 tracks:-

31, 35 23, 26, 2, 5, 7, 10, 17, 15, 19

Where track 31 is the first and track 19 is the last request received. If the initial head position is at track 53 and the head movement is upward then the reordered queue on the basis of SCAN UPWARD algorithm becomes

35, 31, 26, 23, 19, 17, 15, 10, 7, 5, 2

Where track 65 will be first serviced and then track 67, 98, 122, 124, 183, 37 and finally track 14 will be serviced for a total head movement of 300 tracks. (Figure 5.2)

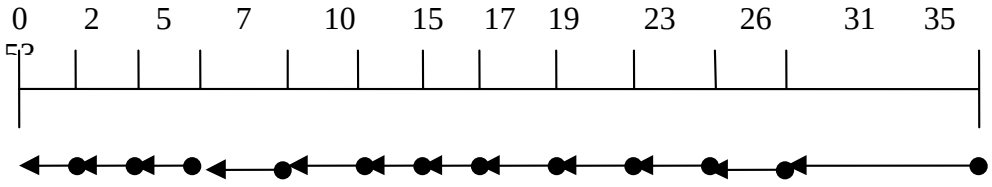


Figure 5.2 Sample-2 Scan Upward disk scheduling.

Sample-3

Let us consider third example which consists of the following 24 tracks:-

47, 35, 180, 132, 156, 23, 34, 5, 210, 83, 96, 103,
88, 76, 113, 128, 73, 120, 169, 123, 111, 179, 1, 6, 40

Where 47 is the first request and 40 is the last request. Before applying scan downward to this example we need to know the direction of head movement, in addition to the head's last position. So if the head was moving upward from the initial head position 53 then the reordered queue becomes

73, 76, 83, 88, 96, 103, 111, 113, 120, 123, 128,
132, 136,156, 169, 179, 180, 210, 47, 40, 35,34, 23, 5

Where track 73 will be serviced first and track 5 will be serviced last, thus resulting in a total head movement of 542 tracks. (Figure 5.3)

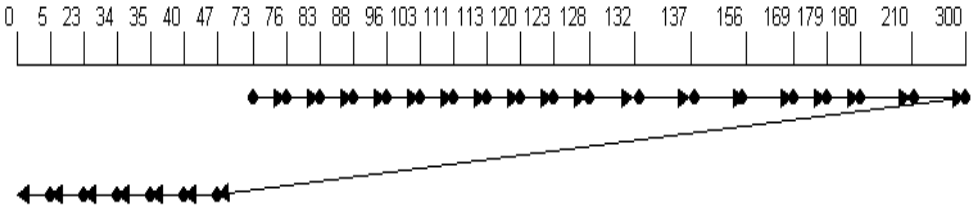


Figure 5.3 Sample-3 Scan Upward disk scheduling.

Sample-4

Now let us apply algorithm to third example which consists of the following 24 tracks:-

77, 1, 132, 149, 31, 211, 250, 285,
48, 19, 38, 37, 34, 175, 25, 20, 42,17, 15, 11 6,5,2, 125

Where track 77 is the first and track 125 is the last request received. If the initial head position is at track 53 and the head movement is upward then the reordered queue on the basis of Scan Upward algorithm becomes

77, 125, 132, 149, 175, 211, 250, 285, 300,
48, 42, 38, 37, 34, 31, 25, 20, 19,17, 15, 11 6,5,2, 1,

Where track 77 will be serviced first and finally track 1 will be serviced for a total head movement of 546 tracks (Figure 5.4)

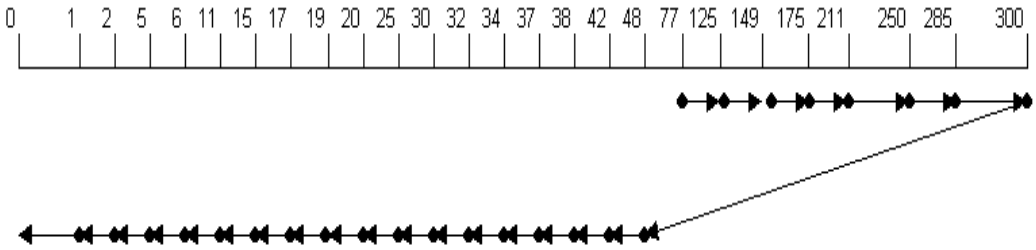


Figure 5.4 Sample-4 Scan Upward disk scheduling

5.3 Scan Downward

Scan Downward algorithm also operates like SSTF except that it chooses the requests that results in the shortest seek distance in a *downward direction.* The read-write head starts movement in the downward direction, and moves till the last track of the disk servicing requests as it reaches each track., After reaching at the last track the direction of head movement is reversed and servicing continues till the availability of requests in this direction. The head continuously scans the disk from end to end.

Sample-1

Let us start with example, which consists of the following 8 tracks:

98, 183, 37, 122, 14, 124, 65, 67

where track 98 is the first and track 67 is the last request received. If the initial head position is at track 53 and the head movement is upward then the reordered queue on the basis of Scan Downward algorithm becomes

37, 14, 0, 65, 67, 98, 122, 124, 183

Where track 37 will be first serviced and finally track 183 will be serviced for a total head movement of 236 tracks. (Figure 5.5)

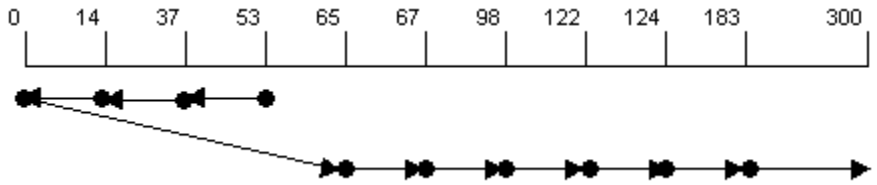


Figure 5.5 Sample-1 Scan Downward disk scheduling

Sample-2

Now we consider second example, which consists of the following 11 tacks:

2,5,7, 10, 15, 17, 19,23,26,29,31,35

Where track 2 is the first and track 35 is the last request. Here all requests are located below the initial head position 53. By applying Scan Down algorithm to this example the reordered queue becomes:

35, 31, 29, 26, 23, 19, 17, 15, 10, 7, 5, 2

Where track 35 wi11 be serviced first then 31, 29, 26, 23, 19, 17, 15, 10, 7. 5 and finally track 2 will be serviced in a total head movement of 51 tracks (Figure 5.6).

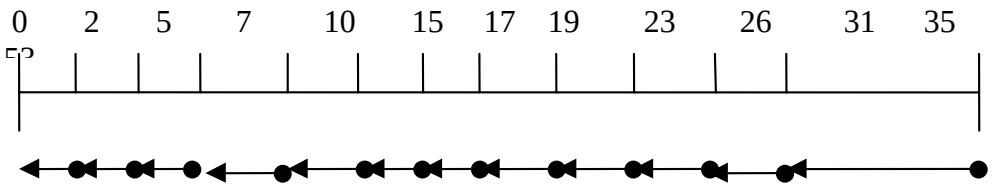


Figure 5.6 Sample-2 Scan Downward disk scheduling.

Sample-3

Now we consider second example, which consists of the following 24 tacks.

47, 35, 180, 132, 156, 23, 34, 5, 210, 83, 96, 103,
88, 76, 113, 128, 73, 120, 169, 123, 111, 179, 1, 6, 40

If the head was moving downward from the initial head position 53 then the reordered queue becomes

47, 40, 35, 34, 23, 5, 0, 73, 76, 83, 88, 96, 103, 111,
113, 120, 123, 128, 132, 136, 156, 169, 179, 180,210,

Where the head will first move to track 47 and finally to track 210 (Figure 5.7), thus resulting in a total head movement of 263 tracks.

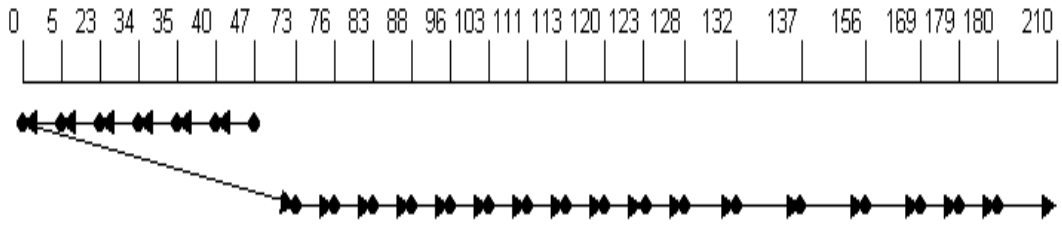


Figure 5.7 Sample-3 Scan Downward disk scheduling.

Sample-4

Let us consider fourth example, which consists of the following 24 tracks:

77, 5, 11, 175, 25, 38, 1, 90, 250, 17, 211, 15,
37, 285, 149, 42, 19, 132, 2, 31, 6, 34, 48, 125

Where track 77 is the first request and track 125 is the last request. If the initial head position is 53 and the head movement is downward, then the reordered queue on the basis of Scan algorithm becomes

48, 42, 38, 37, 34, 31, 25, 20, 19, 17, 15, 11, 6,
5, 2, 1, **0,** 77, 125, 132, 149, 175, 211, 250, 285

Where track 48 will be serviced first then 42, 38, 37, 34, 31, 25, 20, 19, 17, 15, 11, 6, 5, 2, 1, 0, 77, 125, 132, 149, 175, 211, 250, and finally track 285 will be serviced (Figure 5.8) thus resulting in a total head movement of 337 tracks.

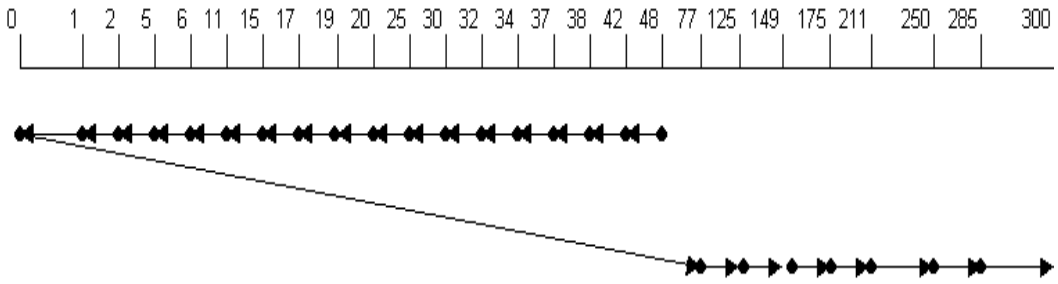


Figure 5.8 Sample-4 Scan Downward disk Scheduling.

5.4 Circular-Scan/Upward

In this type of scheduling, the head is moved upward servicing requests as it goes to the end track. After reaching the last track the head direction is reversed and it then returns to the opposite end track of the disk, without servicing any requests on the return trip. Now from this end it again starts servicing the requests and goes till last request.

Sample-1

Let us start with example, which consists of the following 8 tracks:-

98, 183,37, 122, 14, 124,65,67,

Where track 98 is the first and track 67 is the last request received. If the initial head position is at track 53 then the reordered queue on the basis of C-Scan / Upward algorithm becomes:

65, 67, 98, 122, 194, 183, 14, 37

Where track 65 will be serviced first then it will service 67, 98, 122, 124 and 183. After servicing 183, the head will move to the end track of the disk (which is 300 in this case) from there it will reverse the direction will not service request 37 on the return trip rather it goes straight to track o then to track 14 which is the pending request nearest to start of the disk and then to 37. So the path will be

65, 67, 98, 122, 194, 183, **300**, **0**, 14,37,

The total head movement for this example is 322 tracks (Figure 5.9).

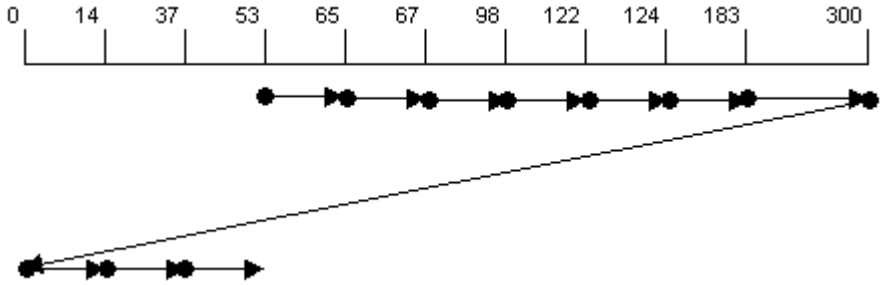


Figure 5.9 Sample-1 C-Scan/Upward disk scheduling.

Sample-2

Let us consider second example, which consists of the following 11 track requests:

2, 35, 10, 7, 6, 17, 19, 31, 15, 9, 5

Where track 2 is the first request and track 35 is the last request received. If the initial head position is at track 53, then after application of C-Scan / Upward algorithm the reordered queue becomes

35, 31, 26, 23, 19, 17, 15, 10, 7, 5, 2

Where track 35 will be serviced first and track 2 will be serviced last. It is pertinent to note here that the head first moves from 53 to track 35, then to 31, 26, 23, 19, 17, 15, 10, 7, 5 and finally to track 2 for a total head movement of 51 tracks (Figure 5.10).

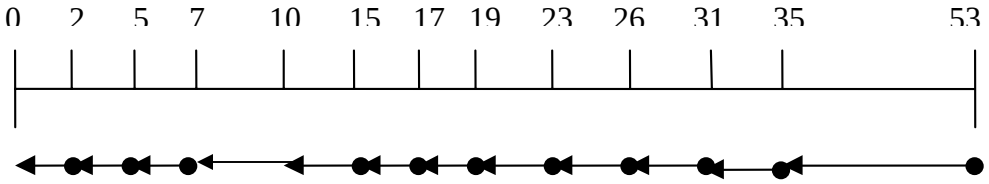


Figure 5.10 C-Scan/Upward disk Scheduling.

Sample-3

Let us consider third example, which consists of the following 24 tracks requests:

47, 35, 180, 132, 156, 23, 34, 5, 210, 83, 96, 103,
88, 76, 113, 198, 73, 120, 169, 193, 111, 179, 136,40,

Where track 47 is the first request and track 40 is the last request as received. If the initial head position is at track 53, then after application of C-Scan / Upward the reordered queue becomes as:-

73, 76, 83, 88, 96, 103, 111, 113, 123, 128,
132, 136, 156, 169, 179, 180,210, **300**, **0**, 5,23,34,35,40,47,

Where track 73 will be served first and track 47 will be served last for a total head movement of 404 tracks (Figure 5.11).

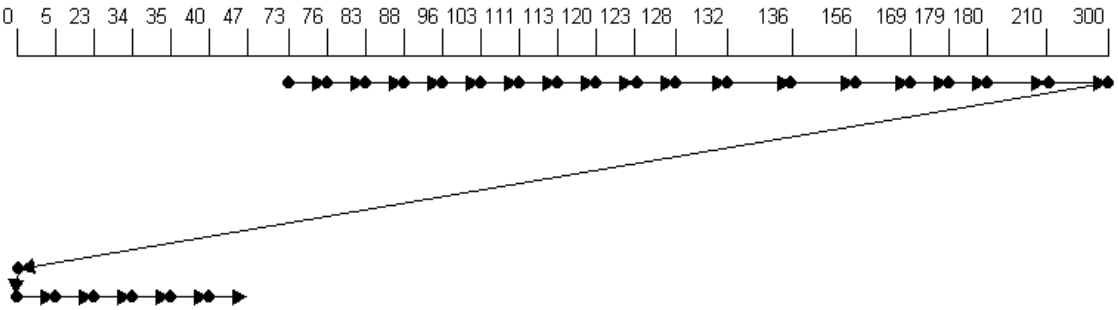


Figure 5.11 C-Scan/Upward disk Scheduling.

Sample-4

Considering fourth example which consists of the following 24 tracks:

77, 5, 11, 175, 25, 38, 1, 20, 250, 17, 211,
15, 37, 285, 149, 42, 19, 132,2,31,6,34,48, 125,

Where track 77 is the first and track 125 is the last request as received. If the initial head position is at track 53, then the reordered queue after the application of C-Scan/Upward becomes

77, 125, 132, 149, 175, 211, 250, 285, **300**, **0**,
1, 2, 5, 6, 11, 15, 17, 19, 20, 25,31,34,37,38,42,48,

Where track 77 will be serviced first and finally track 48 will be serviced, for a total head movement of 563 tracks (Figure 5.12).

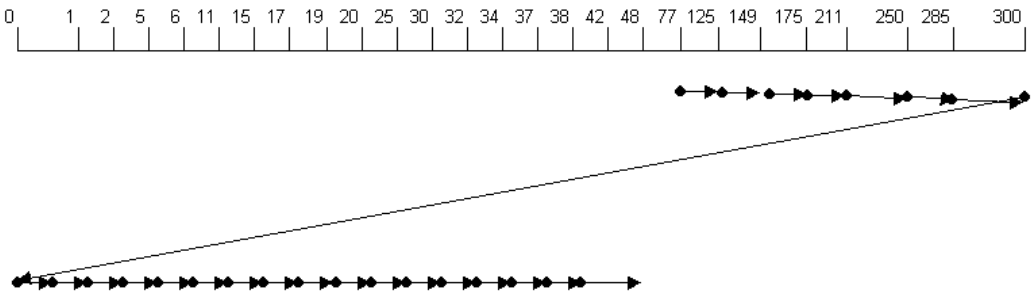


Figure 5.12 C-Scan/Upward disk Scheduling.

5.5 Circular-Scan/Downward

In this type of scheduling, the head is moved downward servicing requests as it goes to the end track. After reaching the last track the head direction is reversed and it then returns to the opposite end track of the disk, without servicing any requests on the return trip. Now from this end it again starts servicing the requests and goes till last request.

Sample-1

Let us start with example 1 which consists of the following 8 tracks:

98, 183,37, 122, 14, 124,65,67,

Where track 98 is the first and track 67 is the last request received. If the initial head position is at track 53 then the reordered queue on the basis of C-Scan/Downward algorithm becomes:

37, 14, 0, 300, 183, 124, 122, **300**, **0**, 98, 67, 65

Where track 37 will be serviced first then it will service 14, 0, 300, 183, 124, 122, 98, 67, and then 65. After servicing 14, as there is no request pending on this side so the

head will go to track o reverse its movement and will not service request 65 on the return trip rather it goes straight to end track on the opposite direction (300 to track). It again reverses it movement direction and starts servicing requests 183,124,122, 98, 67 and finally 65 will be served with total head movements of 322 tracks (Figure 5.13).

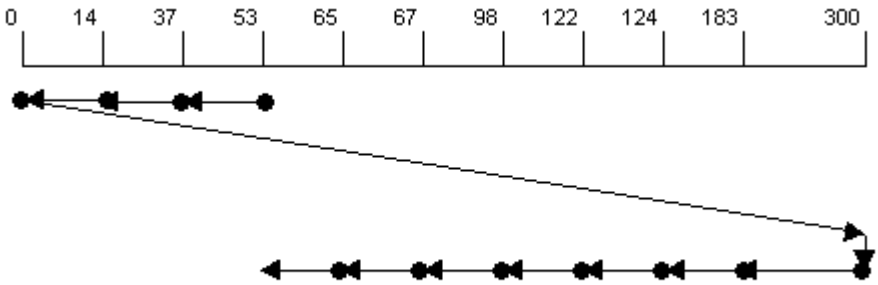


Figure 5.13 Sample-1 C-Scan/Downward disk Scheduling

Sample-2

Let us consider second example, which consists of the following 12 track requests:

2,5,7, 10, 15, 17, 19,'3,'6 '9.35,

Where track 2 is the first request and track 35 is the last request received. If the initial head position is supposed at track 53, then after application of C-Scan/Downward algorithm the reordered queue becomes:

35, 31, 26, 23, 19, 17, 15, 10, 7, 5, 2

Where track 35 will be serviced first and track 2 will be serviced last. It is pertinent to note here that the head first moves from 53 to track 35, then to 31, 26, 23, 19, 17, 15, 10, 7, 5 and finally to track 2 for a total head movement of 51 tracks (Figure 5.14).

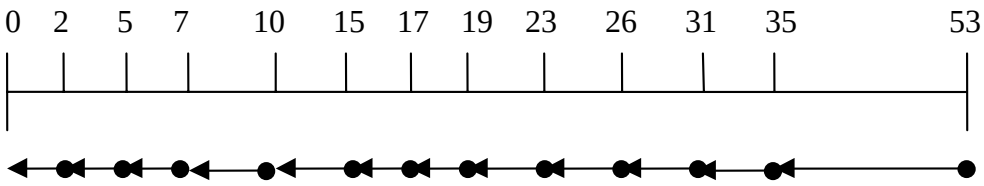


Figure 5.14 Sample-2 C-Scan/Downward disk Scheduling.

Sample-3

Let us consider third example, which consists of the following 24 tracks requests:

47, 35, 180, 132, 156, 23, 34, 5, 210, 83, 96, 103,
88, 76, 113, 198, 73, 120, 169, 193, 111, 179, 136,40,

Where track 47 is the first request and track 40 is the last request as received. If the initial head position is at track 53, then after application of C-Scan /Downward the reordered queue becomes as:

47, 40, 35, 34, 23, 5, **0**, **300**, 210, 180, 179, 169, 156,
136, 132, 128, 123, 120,113,111, 103, 96, 88, 83, 76, 73

Where the head will first move to track 47 and finally to track 210 (Figure 5.15), thus resulting in a total head movement of 263 tracks.

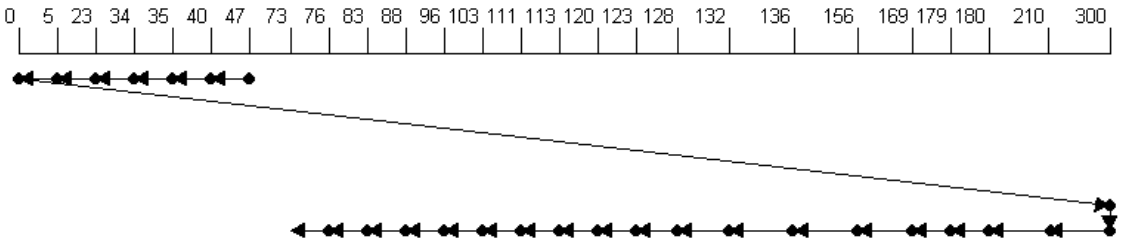


Figure 5.15 Sample-3 C-Scan/Downward disk Scheduling.

Sample-4

Considering fourth example which consists of the following 24 tracks:

77, 5, 11, 175, 25, 38, 1, 20, 250, 17, 211,
15, 37, 285, 149, 42, 19, 132,2,31,6,34,48, 125,

Where track 77 is the first and track 125 is the last request as received. If the initial head position is at track 53, then the reordered queue after the application of C-Scan/Downward becomes

48, 42, 38, 37, 34, 31, 25, 20, 19, 17, 15, 11, 6,
5, 2, 1, **0**, **300**, 285, 250, 211, 175, 149, 132, 125, 77

Where track 48 will be serviced first then 42, 38, 37, 34, 31, 25, 20, 19, 17, 15, 11, 6, 5, 2, 1, **0**, **300**, 285, 250, 211, 175, 149, 132, 125 and finally track 77 will be serviced (Figure 5.16) thus resulting in a total head movement of 337 tracks.

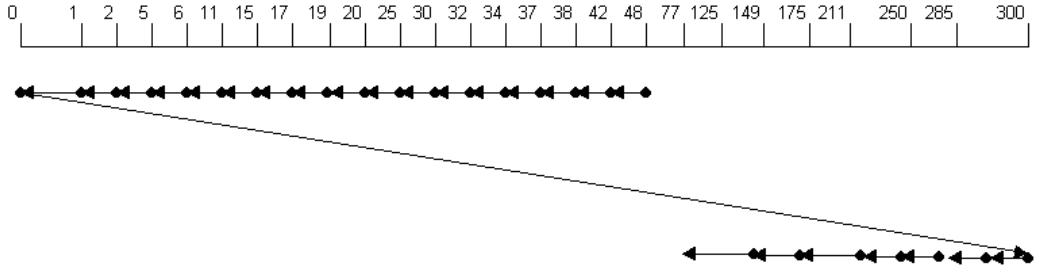


Figure 5.16 C-Scan/Downward disk Scheduling.

5.6 Comparison of algorithms

The four test samples for which we already have results [1] of FCFS, SSTF, Look and C-Look algorithms will be compared with results of Scan/upward, Scan/downward, C- Scan/Upward and C-Scan /Downward.

When we look at table 5.1, it transpires that FCFS algorithm has the largest head movement for all four examples. FCFS can be termed as the least efficient algorithm. However, this algorithm is easy to program and is intrinsically fair as it does not change the position of a request in the queue.

SSTF algorithm can be termed as the best algorithm as for as the criteria of total head movement is concerned. Let us consider sample 1 for which head movement in SSTF is 236 tracks while for LOOK /UP is 299 and LOOK/DOWN is 208 tracks SCAN/UP is 433, SCAN /DOWN is 236, C-SCAN/UP is 584 and for C-SCAN/ DOWN is 588(Table 5.1). Now if we compare results of three algorithms with least head movements which are SSTF, LOOK and SACN/Down it becomes clear that result in LOOK /DOWN is better than that of SSTF and SACN/DOWN. But we must bear in mind the fact that total head movement in LOOK and SCAN algorithm depends on the preferred head movement direction and if the actual head movement is different than the supposed one then it will change the result.

The drawback of SSTF algorithm is that, in a real system, SSTF algorithm may cause *starvation* of some requests. To clear this point further let us assume that we have two requests in the queue, for 14 and 186. If a request near 14 arrives while we are servicing that request, it will be serviced next, making the request at 186 to wait. While this request is being serviced, another request close to 14 could arrive. In theory, a continual stream of requests near one another could arrive causing the request for track 186 to wait indefinitely. Such a situation is called starvation.

From table 5.1 we note that for all examples the total head movement in case of LOOK is lesser than FCFS and C-LOOK SCAN and C-SCAN algorithms. However when we compare LOOK with SSTF, it transpires that in some cases LOOK and SCAN results in lesser head movement than SSTF while in some cases the results of both are same. Like in example I - the head movement for LOOK/DOWN is 208 tracks which is lesser than 236 tracks for SSTF algorithm but the head movement in LOOK/UP is 299 tracks which is more than the total head movement for SSTF (236).

S/No	Total Tracks	Algorithms	Initial Head Position	Head Movement Direction	Total Head Movement (In Tracks)
1.	8	FCFS	53	N.A	640
2.	8	SSTF	53	N A	236
3.	8	LOOK	53	U/WARD	299
4.	8	LOOK	53	D/WARD	208
5.	8	C-LOOK	53	N.A	322
6.	8	SCAN	53	U/WARD	433
7.	8	SCAN	53	D/WARD	236
8.	8	C-SCAN	53	U/WARD	584
9.	8	C-SCAN	53	D/WARD	588
10	12	FCFS	53	N.A	84
11	12	SSTF	53	N A	51
12	12	LOOK	53	D/WARD	51
13	12	C-LOOK	53	N.A	84
14	12	SCAN	53	U/WARD	298
15	12	SCAN	53	D/WARD	53
16	12	C-SCAN	53	U/WARD	300
17	12	C-SCAN	53	D/WARD	353
18	24	FCFS	53	N.A	1255
19	24	SSTF	53	N A	253
20	24	LOOK	53	U/WARD	362
21	24	LOOK	53	D/WARD	253
22	24	C-LOOK	53	N.A	404
23	24	SCAN	53	U/WARD	542
24	24	SCAN	53	D/WARD	263
25	24	C-SCAN	53	U/WARD	679
26	24	C-SCAN	53	D/WARD	570
27	24	FCFS	53	N.A	2290
28	24	SSTF	53	N A	336
29	24	LOOK	53	U/WARD	516
30	24	LOOK	53	D/WARD	336
31	24	C-LOOK	53	N.A	563
32	24	SCAN	53	U/WARD	546
33	24	SCAN	53	D/WARD	337
34	24	C-SCAN	53	U/WARD	595
35	24	C-SCAN	53	D/WARD	637

Table 5.1 Head Movement Comparison of Different Algorithms

If we consider example 3 we note that for SSTF the total head movement is 253 tracks whereas for LOOK (U/WARD) the total head movement is 362 tracks and for LOOK (D/WARD) the total head movement is 253 tracks which is similar to SSTF.

Now considering example 4, it comes to light that the total head movement for SSTF and LOOK/DOWN is 336 tracks whereas for LOOK /UP is 516 tracks which is more than SSTF.

From the above discussion we came to the conclusion that at times LOOK behaves like or better than SSTF algorithm but the drawback of LOOK is that, result depends on the direction of head movement and nature of the pattern. Furthermore, if a request arrives in the queue just in front of the head, it will be serviced almost immediately, whereas a request arriving just behind the head will have to wait until the head moves to the end of the disk, reverses direction, and returns, before being serviced.

Now let us consider C-LOOK algorithm. From table 5.1 we note that total head movement in C-LOOK is lesser than FCFS and more than SSTF algorithm. For example 1 the total head movement in C-LOOK is 322 tracks which is more than total head movement for LOOK/UP that is 299 tracks and LOOK/UP that is 208 tracks. The difference in total head movement of LOOK and C-LOOK is substantial in case of LOOK/DOWN but in case of LOOK/UP this difference is little. Almost similar reasoning holds for the results obtained for examples 2, 3, and 4 .

From table 5.2 it is clear that SCAN/UP and SCAN/DOWN algorithms have an overhead of going to end track irrespective of presence of a request. This peculiar nature of SCAN algorithm makes it inefficient as compared to SSTF(Shortest Seek Time First), Look/Upward, Look/Downward algorithm. The table 5.2 shows the overhead as compared to Look for all the four samples considered. In a real-time system, if a request arrives just in front of the head, it will be serviced almost immediately, whereas a request arriving just behind the head will have to wait until the head moves to the end of the disk, reverses direction and returns to service the leftover requests.

	Scan/Up	**Scan/Down**
Sample-1	134	26
Sample-2	247	2
Sample-3	180	10
Sample-4	30	1

Table 5.2 Head Movement Overheads

From table 5.3 it is clear that C-SCAN algorithm has an overhead of going to end track irrespective of presence of a request plus the total number of tracks as it does not serve any request in its way back, but it goes to opposite end track and then starts servicing the requests. This peculiar nature of C-SCAN algorithm makes it inefficient in terms of total head movement as compared to SCAN/UP and SACN/DOWN algorithms. C-Scan algorithm focuses on equalizing the waiting time for each request among the queue. The table 5.2 shows the overhead of head movements as compared to look for all the four samples considered.

	C-Scan/Up	**C-Scan/Down**
Sample-1	151	252
Sample-2	2	300*
Sample-3	137	307
Sample-4	49	300

Table 5.3 Head Movement Overheads

5.7 Analysis

In this chapter we have discussed the results of four test samples obtained for FCFS, SSTF, LOOK/DOWN, LOOK/UP, SCAN/UP, SCAN/ DOWN, C-SCAN/UP and C-SCAN/DOWN algorithms. FCFS is found to be the least efficient and SSTF is the most efficient in terms of head movement. All the eight algorithms have been studied and analyzed. They can be prioritized for their performance in following order.

SSTF (Shortest Seek Time First) algorithm has performed the best for all kinds of the request queue patterns, but may causes starvation for some of the request.

LOOK algorithm has produced results, which are comparable with SSTF, but performance of LOOK algorithm is dependent on the head movement and nature of the request pattern.

C-LOOK algorithm has an overhead of large swing (serving the last request on one end and then turning to the first request on the other end) as compared with LOOK. Therefore its performance is degraded as compared to LOOK.

SCAN algorithm has an overhead (touching the last track after serving the last request on one end and then reversing to the immediate request) as compared to C-LOOK, which makes its less efficient.

C-SCAN has additional overhead (after touching the last track returning to opposite end of the disk) as compared to SCAN. Therefore it becomes less efficient than SCAN.

FCFS (First Come First Serve) is last in the order of performance as practically it processes the requests as they are received.

Chapter 6

Conclusions

6.1 General

Multiprogramming systems task the storage media with multiple requests as number of processes need to access the disk for reading and writing of data at same time. Therefore a queue of request is always pending to be served. It requires disk drives to have a fast access time and more bandwidth. We can improve both the access time and the bandwidth by scheduling the servicing of disk input / output requests in a good order.

6.2 Objectives

a. Study of data storage / retrieval techniques from Hard Disk.
b. Study of Disk Scheduling Algorithms.
c. Development of Algorithms in C language for simulating Scan/Down Ward, Scan/Upward, C-Scan/Upward and C-Scan/Downward.
d. Testing of different request samples with different head positions for simulated algorithms.
e. Comparison of results for SSTF, FCFS, Look and C-Look [1] with Scan/Downward, Scan/Upward, C-Scan/Upward and C-Scan/Downward.
f. Prioritizing the algorithms on bases of their performance for future reference.

6.3 Achievements

a. Detailed study /discussion on data storage /retrieval techniques has been carried out and is presented in chapter-1 and 2.

b. A simulator for Scan/Upward, Scan/Downward, C-Scan/Upward and C-Scan/Downward has been developed in C language.

c. Study of Scan/Upward, Scan/Downward, C-Scan/Upward and C-Scan/Downward on the bases of results of ten samples is presented in Chapter –4.

g. Study and comparison of SSTF, FCFS, Look and C-Look [1] with Scan/Downward, Scan/Upward, C-Scan/Upward and C-Scan/Downward. on the bases of four samples is presented in Chapter-5.

d. On the bases of performance results the algorithms have been prioritized.

6.4 Evaluation and Comparison of Results

a. ***SSTF (Shortest Seek Time First)*** algorithm has performed the best for all kinds of the request queue patterns.

b. ***LOOK*** algorithm has produced results, which are comparable with SSTF, but performance of LOOK algorithm is dependent on the head movement.

c. ***C -LOOK*** algorithm has an overhead of large swing (serving the last request on one end and then turning to the first request on the other end) as compared with LOOK. Therefore its performance is degraded as compared to LOOK.

d. ***SCAN*** algorithm has an overhead (touching the last track after serving the last request on one end and then reversing to the immediate request) as compared to C-LOOK, which makes its less efficient.

e. ***C-SCAN*** has additional overhead (after touching the last track returning to opposite end of the disk) as compared to SCAN. Therefore it becomes less efficient then SCAN algorithm.

f. ***FCFS (First Come First Serve)*** is last in the order of performance as practically it follows no algorithm, but processes the requests as they are received.

6.5 Who Can Use It?

This simulator can be used by those students of BSc or MSc Computer Engineering who are taking a course in Operating System. It will help to clear most of their concepts regarding the working of disk scheduling algorithms. I hope that it will also be beneficial for the teachers of Operating Systems as well as those researchers who intend to work further in this field.

6.6 Recommendation for Future Work

Study of the entire system reveals that pattern of request queue has unpredictable nature moreover analysis of the results clearly indicates that every algorithm has it peculiar nature. It may work efficiently in one type of pattern but may behave differently in other type of pattern There is no algorithm which can be declared best in all type of requested queues. Therefore future work is recommended on following lines:-

a. Development of new algorithm, which can cater for all possible eventualities of request queue.

b. Development of system which can find out the nature of request queue and activate the algorithm respectively.

c. Development of system, which can use combination of different algorithms depending on the nature of the queue.

d. Implement priority algorithm for comparison with currently obtained results of FCFS, SSTF, LOOK/U, LOOK/D, C-LOOK/U, C-LOOK/D, SCAN/U, SACN/D, C-SCAN/U AND C-SCAN/D.

REFERENCES

[1] Dr. Muhammad Younus Javed "**Simulation of Disk Scheduling Algorithms**" TENCON International Conference 2000, Kualalumpur, Malaysia, September 2000.

[2] Abraham Silberschatz, James L Peterson & Peter B Galvin **"Operating System Concepts (3rd Edition)"**

[3] Colin Ritchi "**Operating System (2nd Edition)"**

[4] William Stallings **" operating Systems (Second Edition)"**

[5] Harvey M. Deitel **"An Introduction to Operating System (2nd Edition)"**

[6] Robert Lafore **"Programming for the PC and Turbo C++"**

[7] Yashwant Kanetkhar **"Pointer in C"**

[8] M. Morris Mano **"Computer System Architecture (3rd Edition)"**