Operating System (5th semester)

Prepared by SANJIT KUMAR BARIK (ASST PROF, CSE)

MODULE-IV

TEXT BOOK:

1. Operating System Concepts – Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, 8th edition, Wiley-India, 2009.
2. Modern Operating Systems – Andrew S. Tanenbaum, 3rd Edition, PHI
3. Operating Systems: A Spiral Approach – Elmasri, Carrick, Levine, TMH Edition.

DISCLAIMER:

UNIT-IV
FILE SYSTEM

File concept:

A file is a collection of related information that is stored on secondary storage. Information stored in files must be persistent i.e. not affected by power failures & system reboots. Files represent both programs as well as data.

Part of the OS dealing with the files is known as ***file system***. The File system takes care of the following issues

- **File Structure**

  We have seen various data structures in which the file can be stored. The task of the file system is to maintain an optimal file structure.

- **Recovering Free space**

  Whenever a file gets deleted from the hard disk, there is a free space created in the disk. There can be many such spaces which need to be recovered in order to reallocate them to other files.

- **disk space assignment to the files**

  The major concern about the file is deciding where to store the files on the hard disk. There are various disks scheduling algorithm which will be covered later.

- **tracking data location**

  A File may or may not be stored within only one block. It can be stored in the non contiguous blocks on the disk. We need to keep track of all the blocks on which the part of the files reside.

The important file concepts include:

1. **File attributes:** A file has certain attributes which vary from one operating system to another.
   - **Name:** Every file has a name by which it is referred.
   - **Identifier:** It is unique number that identifies the file within the file system.
   - **Type:** This information is needed for those systems that support different types of files.
   - **Location:** It is a pointer to a device & to the location of the file on that device
   - **Size:** It is the current size of a file in bytes, words or blocks.
   - **Protection:** It is the access control information that determines who can read, write & execute a file.

     EX: The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.
   - **Time, date & user identification:** It gives information about time of creation or last modification & last use.
2. **File operations:** The operating system can provide system calls to create, read, write, reposition, delete and truncate files.
   - **Creating files:** Two steps are necessary to create a file. First, space must be found for the file in the file system. Secondly, an entry must be made in the directory for the new file. Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.
   - **Reading a file:** Data & read from the file at the current position. The system must keep a read pointer to know the location in the file from where the next read is to take place. Once the read has been taken place, the read pointer is updated. Every file is opened in three different modes: Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.

- **Writing a file:** Data are written to the file at the current position. The system must keep a write pointer to know the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Repositioning within a file (seek):** The directory is searched for the appropriate entry & the current file position is set to a given value. After repositioning data can be read from or written into that position. So, Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as seeking.
- **Deleting a file:** To delete a file, we search the directory for the required file. After deletion, the space is released so that it can be reused by other files. Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.

- **Truncating a file:** Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

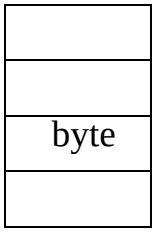
3. **File types:** The file name is spilt into 2 parts, Name & extension. Usually these two parts are separated by a period. The user & the OS can know the type of the file from the extension itself. Listed below are some file types along with their extension:

File Type	Extension
Executable File	exe, bin, com
Object File	obj, o (compiled)
Source Code file	C, C++, Java, .pas(Pascal)
Batch File	bat, sh (commands to command interpreter)
Text File	txt, doc (textual data documents)

Archieve File	arc, zip, rar (related files grouped together into file compressed for storage)
Multimedia File	mpeg (Binary file containing audio or A/V information)

4. **File structure:** Files can be structured in several ways. Three common possible are:

- **Byte sequence:** The figure shows an unstructured sequence of bytes. The OS doesn‘t care about the content of file. It only sees the bytes. This structure provides maximum flexibility. Users can write anything into their files & name them according to their convenience. Both UNIX & windows use this approach.



- **Record sequence:** In this structure, a file is a sequence of fixed length records. Here the read operation returns one records & the write operation overwrites or append

- **Tree:** In this organization, a file consists of a tree of records of varying lengths. Each record consists of a key field. The tree is stored on the key field to allow first searching for a particular key.

Access methods: Basically, access method is divided into 2 types:

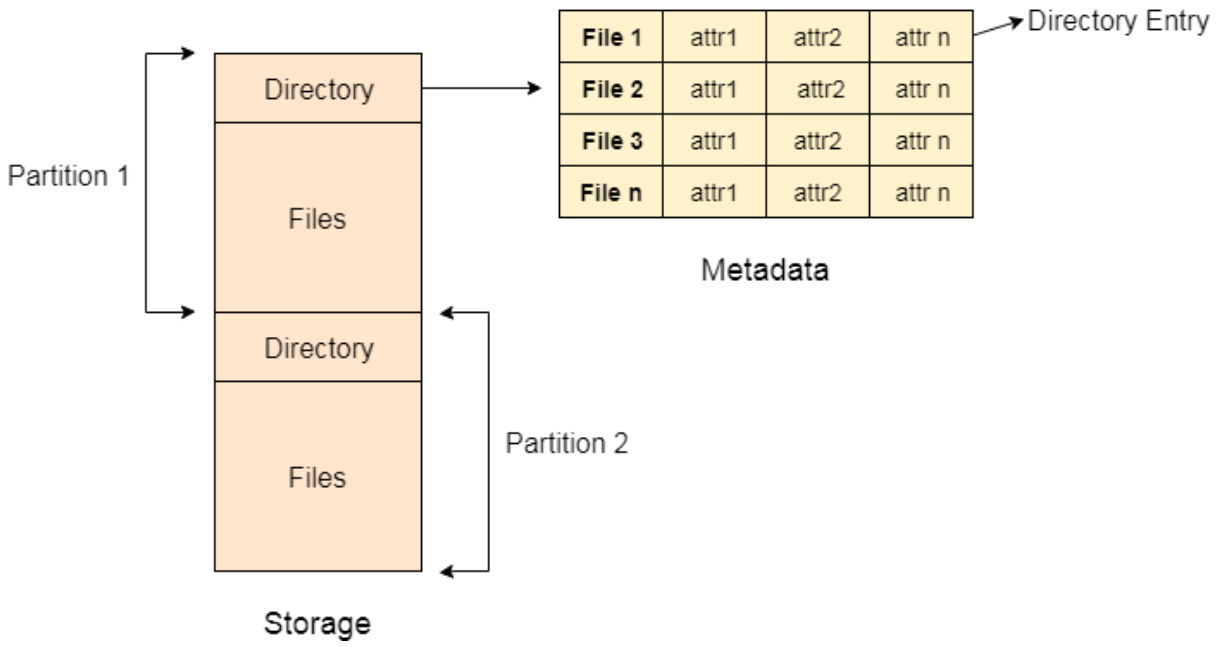
- **Sequential access:** It is the simplest access method. Information in the file is processed in order i.e. one record after another. A process can read all the data in a file in order starting from beginning but can‘t skip & read arbitrarily from any location. Sequential files can be rewound. It is convenient when storage medium was magnetic tape rather than disk.

- **Direct access:** A file is made up of fixed length-logical records that allow programs to read & write records rapidly in no particular order. This method can be used when disk are used for storing files. This method is used in many applications e.g. database systems.

  Ex: If an airline customer wants to reserve a seat on a particular flight, the reservation program must be able to access the record for that flight directly without reading the records before it. In a direct access file, there is no restriction in the order of reading or writing. For example, we can read block 14, then read block 50 & then write block 7 etc. Direct access files are very useful for immediate access to large amount of information.

Directory structure: The file system of computers can be extensive. Some systems store thousands of file on disk. To manage all these data, we need to organize them. The organization is done in 2 steps. The file system is broken into partitions. Each partition contains information about file within it.

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.



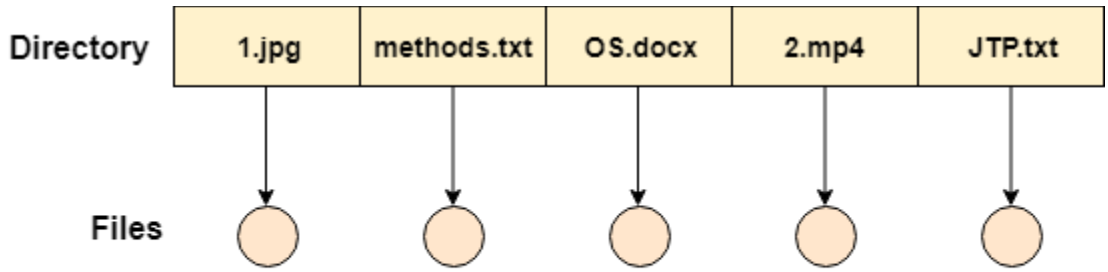
A directory can be viewed as a file which contains the Meta data of the bunch of files.

Operation on a directory:

- **Search for a file:** We need to be able to search a directory for a particular file.
- **Create a file:** New files are created & added to the directory.
- **Delete a file:** When a file is no longer needed, we may remove it from the directory.
- **List a directory:** We should be able to list the files of the directory.
- **Rename a file:** The name of a file is changed when the contents of the file changes.
- **Traverse the file system:** It is useful to be able to access every directory & every file within a directory.

Structure of a directory: The most common schemes for defining the structure of the directory are:

1. **Single level directory:** It is the simplest directory structure. All files are present in the same directory. So it is easy to manage & understand.



Single Level Directory

Limitation: A single level directory is difficult to manage when the no. of files increases or when there is more than one user. Since all files are in same directory, they must have unique names. So, there is confusion of file names between different users.

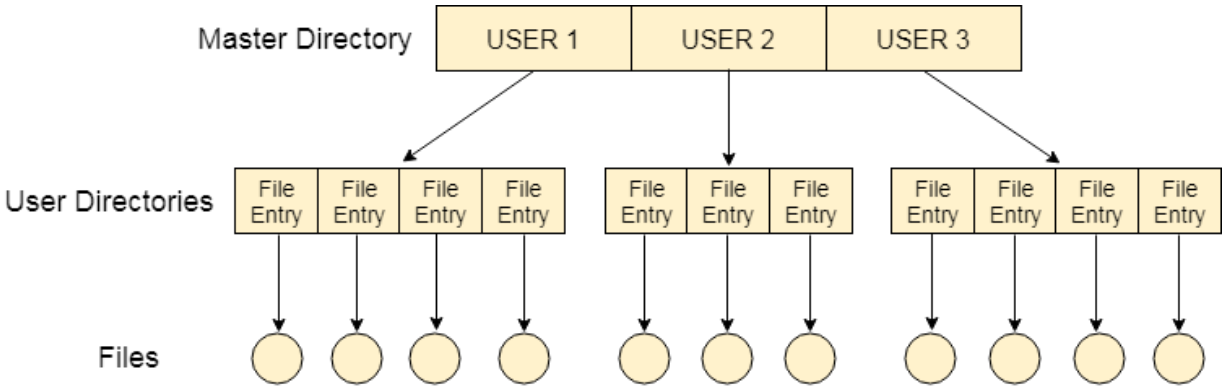
Advantages

1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

1. We cannot have two files with the same name.
2. The directory may be very big therefore searching for a file may take so much time.
3. Protection cannot be implemented for multiple users.
4. There are no ways to group same kind of files.
5. Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

2. **Two level directories:** The solution to the name collision problem in single level directory is to create a separate directory for each user. In a two level directory structure, each user has its own user file directory. When a user logs in, then master file directory is searched. It is indexed by user name & each entry points to the UFD of that user.

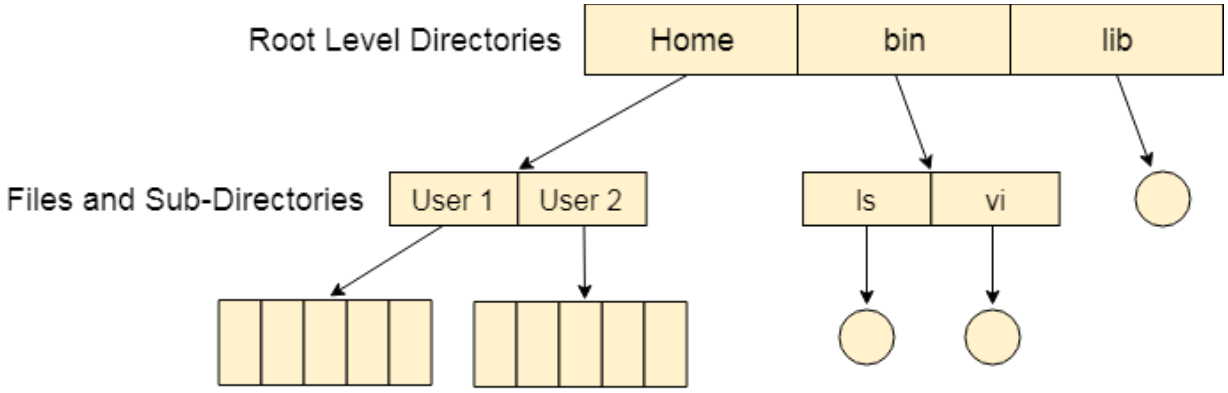


Two Level Directory

Limitation: It solves name collision problem. But it isolates one user from another. It is an advantage when users are completely independent. But it is a disadvantage when the users need to access each other‘s files & co-operate among themselves on a particular task.

3. **Tree structured directories:** It is the most common directory structure. A two level directory is a two level tree. So, the generalization is to extend the directory structure to a tree of arbitrary height. It allows users to create their own subdirectories & organize their files. Every file in the system has a unique path name. It is the path from the root through all the sub-directories to a specified file. Each user has a current directory. It contains most of the files that are of current interest to the user. Path names can be of two types: An absolute path name begins from the root directory & follows the path down to the specified files. A relative path name defines the path from the current directory.

   EX: If the current directory is *root/spell/mail*, then the relative path name is *prt/first* & the absolute path name is *root/ spell/ mail/ prt/ first*. Here users can access the files of other users also by specifying their path names.



The Structured Directory System

Advantage: Searching is more efficient in this directory structure.

Permissions on the file and directory

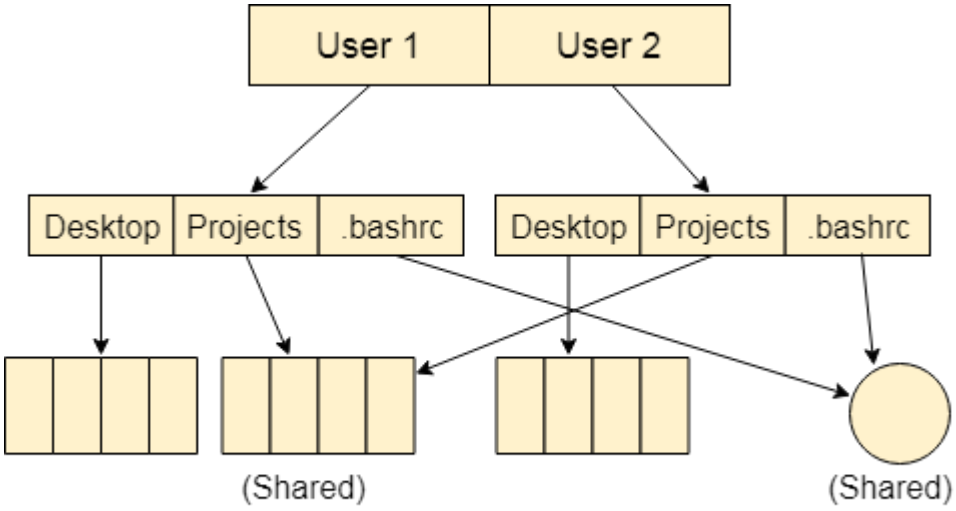
A tree structured directory system may consist of various levels therefore there is a set of permissions assigned to each file and directory.

The permissions are **R W X** which are regarding reading, writing and the execution of the files or directory. The permissions are assigned to three types of users: owner, group and others.

There is a identification bit which differentiate between directory and file. For a directory, it is **d** and for a file, it is dot **(.)**

4. **A cyclic graph directory:** It is a generalization of tree structured directory scheme. An a cyclic graph allows directories to have shared sub-directories & files. A shared directory or file is not the same as two copies of a file. Here a programmer can view the copy but the changes made in the file by one programmer are not reflected in the other‘s copy. But in a shared file, there is only one actual file. So many changes made by a person would be immediately visible to others. This scheme is useful in a situation where several people are

working as a team. So, here all the files that are to be shared are put together in one directory. Shared files and sub-directories can be implemented in several ways. A common way used in UNIX systems is to create a new directory entry called link. It is a pointer to another file or sub-directory. The other approach is to duplicate all information in both sharing directories. A cyclic graph structure is more flexible then a tree structure but it is also more complex.



Acyclic-Graph Structured Directory System

Limitation: Now a file may have multiple absolute path names. So, distinct file names may refer to the same file. Another problem occurs during deletion of a shared file. When a file is removed by any one user. It may leave dangling pointer to the non existing file. One serious problem in a cyclic graph structure is ensuring that there are no cycles. To avoid these problems, some systems do not allow shared directories or files. E.g. MS-DOS uses a tree structure rather than a cyclic to avoid the problems associated with deletion. One approach for deletion is to preserve the file until all references to it are deleted.

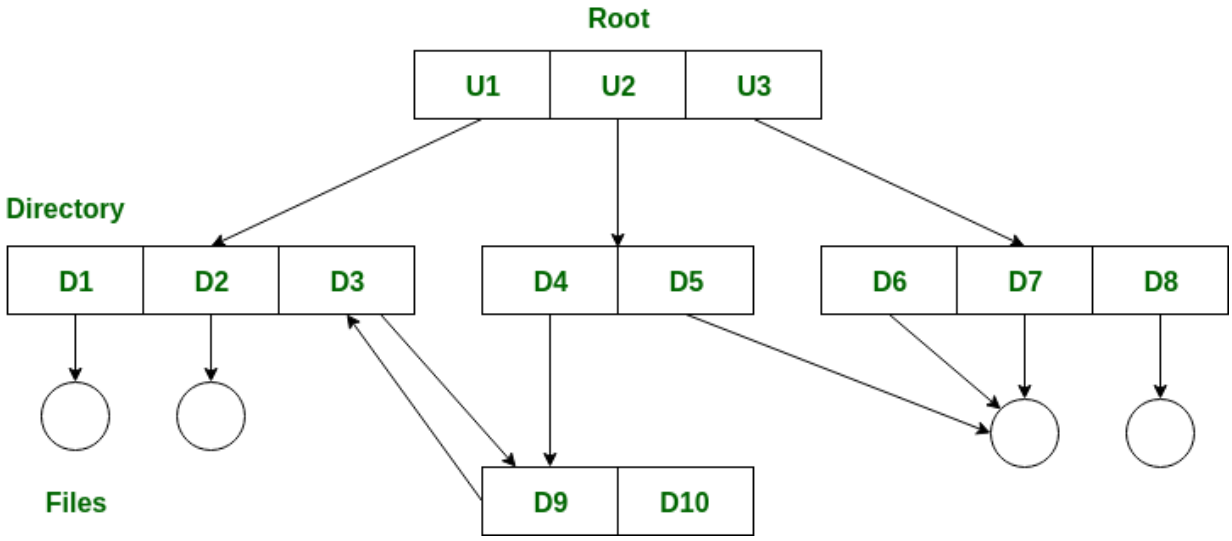
5. **General graph directory:** When links are added to an existing tree structured

directory, the tree structure is destroyed, resulting in a simple graph structure. Linking is a technique that allows a file to appear in more than one directory. The advantage is the simplicity of algorithm to transverse the graph & determines when there are no more references to a file. But a similar problem exists when we are trying to determine when a file can be deleted. Here also a value zero in the reference count means that there are no more references to the file or directory & the file can be deleted. But when cycle exists, the reference count may be non-zero even when there are no references to the directory or file. This occurs due to the possibility of self referencing (cycle) in the structure. So, here we have to use garbage collection scheme to determine when the last references to a file has been deleted & the space can be reallocated. It involves two steps:

- Transverse the entire file system & mark everything that can be accessed.
- Everything that isn‘t marked is added to the list of free space.

  But this process is extremely time consuming. It is only necessary due to presence of cycles in the graph. So, a cyclic graph structure is easier to work than this.

  In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.


Root
U1
U2
U3
Directory
D1
D2
D3
D4
D5
D6
D7
D8
Files
D9
D10

Protection

When information is kept in a computer system, a major concern is its protection from physical damage (reliability) as well as improper access.

Types of access: In case of systems that don‘t permit access to the files of other users. Protection is not needed. So, one extreme is to provide protection by prohibiting access. The other extreme is to provide free access with no protection. Both these approaches are too extreme for general use. So, we need controlled access. It is provided by limiting the types of file access. Access is permitted depending on several factors. One major factor is type of access requested. The different types of operations that can be controlled are:

- **Read**
- **Write**
- **Execute**
- **Append**
- **Delete**
- **List**

Access lists and groups:

Various users may need different types of access to a file or directory. So, we can associate an access lists with each file and directory to implement identity dependent access. When a user access requests access to a particular file, the OS checks the access list associated with that file. If that user is granted the requested access, then the access is allowed. Otherwise, a protection violation occurs & the user is denied access to the file. But the main problem with access lists is their length. It is very tedious to construct such a list. So, we use a condensed version of the access list by classifying the users into 3 categories:

- **Owners:** The user who created the file.

- **Group:** A set of users who are sharing the files.
- **Others:** All other users in the system.

Here only 3 fields are required to define protection. Each field is a collection of bits each of which either allows or prevents the access. E.g. The UNIX file system defines 3 fields of 3 bits each: r w x

- r(read access)
- w(write access)
- x(execute access)

Separate fields are kept for file owners, group & other users. So, a bit is needed to record protection information for each file.

Allocation methods:

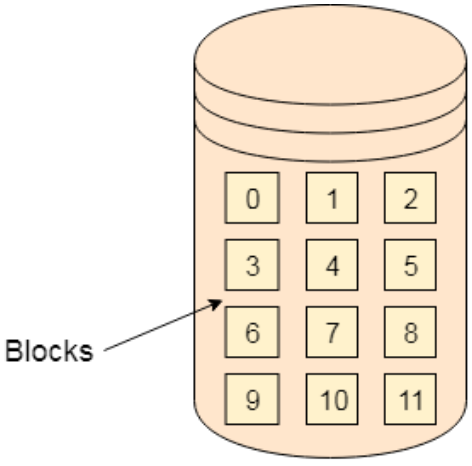
There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

We have mainly discussed 3 methods of allocating disk space which are widely used.

1. **Contiguous allocation:**

   If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

   In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



Hard Disk

File Name	Start	Length	Allocated Blocks
abc.text	0	3	0,1,2
video.mp4	4	2	4,5
jtp.docx	9	3	9,10,11

Directory

Contiguous Allocation

a. It requires each file to occupy a set of contiguous blocks on the disk.

b. Number of disk seeks required for accessing contiguously allocated file is minimum.

c. The IBM VM/CMS OS uses contiguous allocation. Contiguous allocation of a file is defined by the disk address and length (in terms of block units).

d. If the file is ‘n‘ blocks long and starts at location ‘b‘, then it occupies blocks b, b+1, b+2,----,b+ n-1.

e. The directory for each file indicates the address of the starting block and the length of the area allocated for each file.

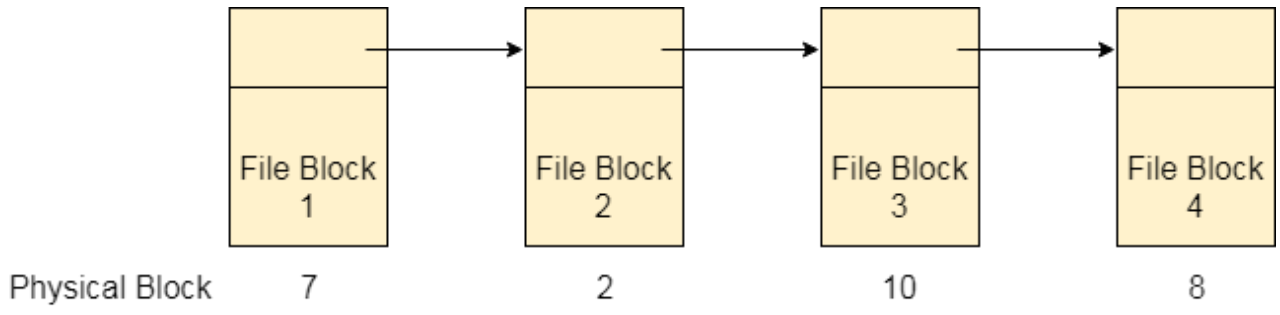
f. Contiguous allocation supports both sequential and direct access. For sequential access, the file system remembers the disk address of the last block referenced and reads the next block when necessary.

g. For direct access to block i of a file that starts at block b we can immediately access block b+i

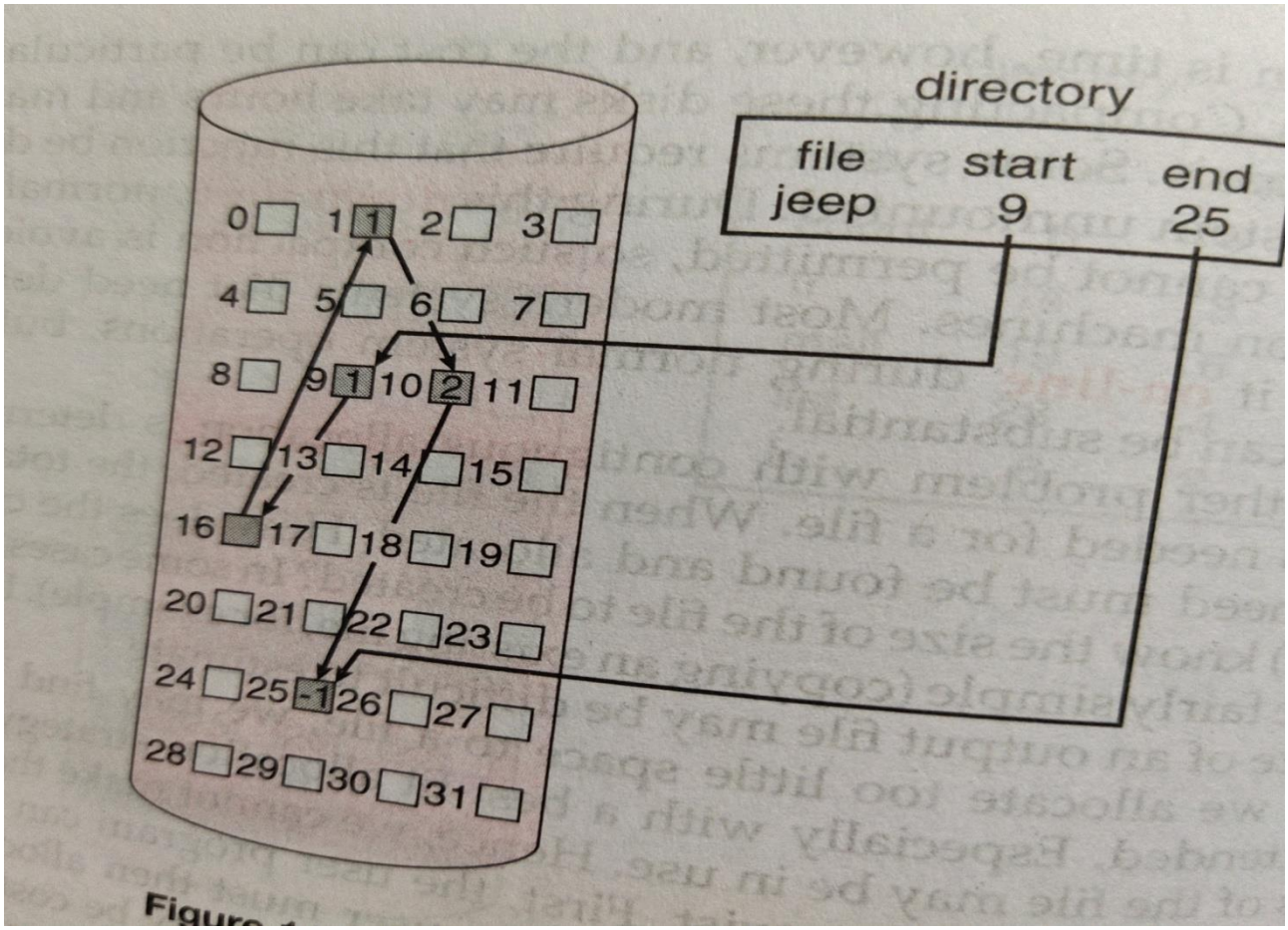
Problems: One difficulty with contiguous allocation is finding space for a new file. It also suffers from the problem of external fragmentation. As files are deleted and allocated, the free disk space is broken into small pieces. A major problem in contiguous allocation is how much space is needed for a file. When a file is created, the total amount of space it will need must be found and allocated. Even if the total amount of space needed for a file is known in advances, pre-allocation is inefficient. Because a file that grows very slowly must be allocated enough space for its final size even though most of that space is left unused for a long period time. Therefore, the file has a large amount of internal fragmentation.

2. **Linked List Allocation**

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



Linked List Allocation



a. Linked allocation solves all problems of contiguous allocation.
b. In linked allocation, each file is linked list of disk blocks, which are scattered throughout the disk.
c. The directory contains a pointer to the first and last blocks of the file.
d. Each block contains a pointer to the next block.
e. These pointers are not accessible to the user. To create a new file, we simply create a new entry in the directory.
f. For writing to the file, a free block is found by the free space management system and this new block is written to & linked to the end of the file.
g. To read a file, we read blocks by following the pointers from block to block.
h. There is no external fragmentation with linked allocation & any

free block can be used to satisfy a request.

i. Also there is no need to declare the size of a file when that file is created. A file can continue to grow as long as there are free blocks.

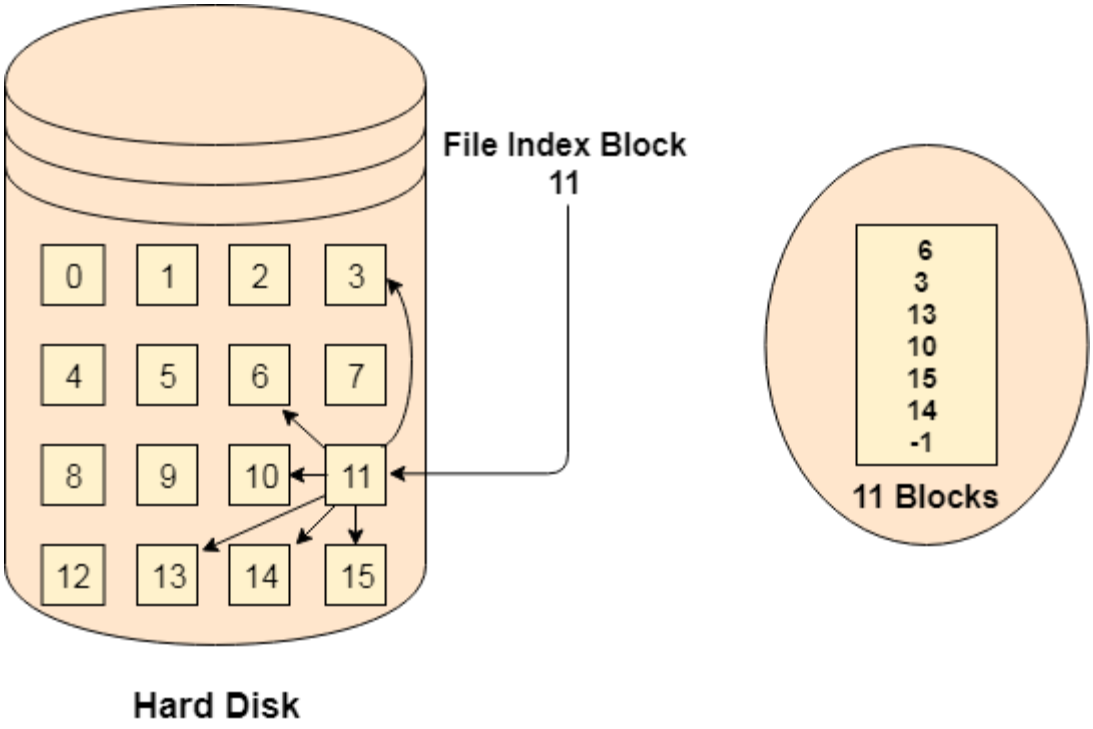
Limitations: It can be used effectively only for sequential access files. To find the i th block of the file, we must start at the beginning of that file and follow the pointers until we get the ith block. So it is inefficient to support direct access files. Due to the presence of pointers each file requires slightly more space than before. Another problem is reliability. Since the files are linked together by pointers scattered throughout the disk. What would happen if a pointer were lost or damaged?

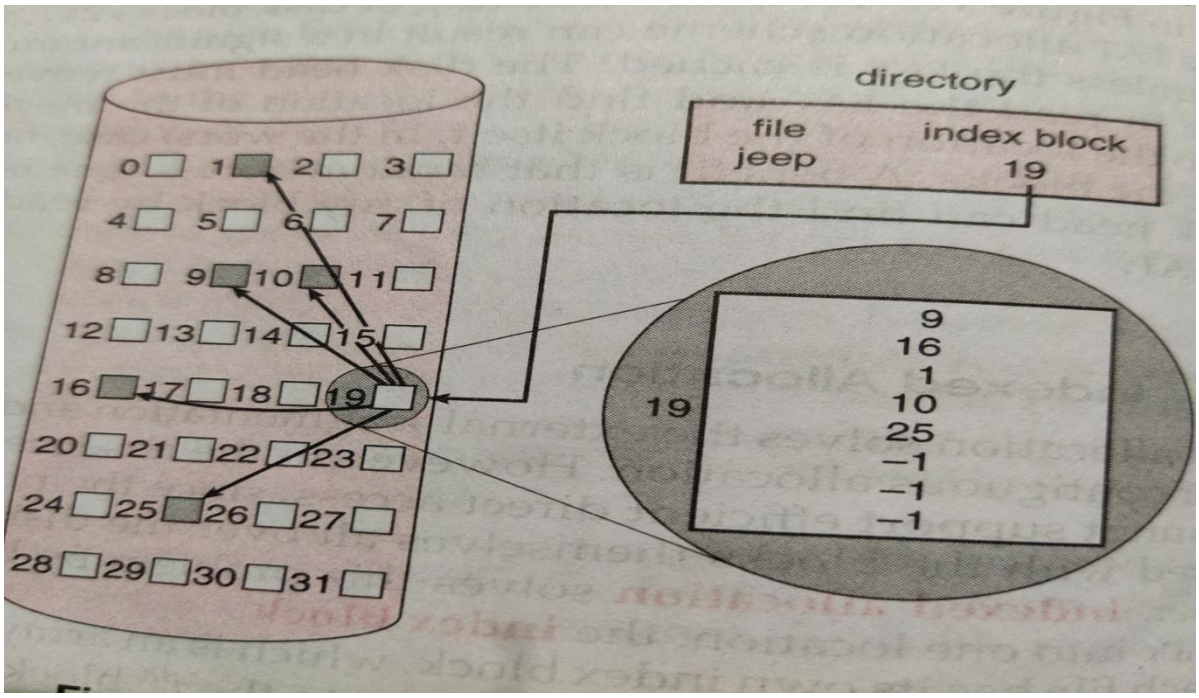
Disadvantages

1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block.

3. Indexed Allocation:

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.





a. Indexed allocation solves the problem of linked allocation by bringing all the pointers together to one location known as the index block.

b. Each file has its own index block which is an array of disk block addresses. The i^{th} entry in the index block points to the i^{th} block of the file.

c. The directory contains the address of the index block. The read the i^{th} block,

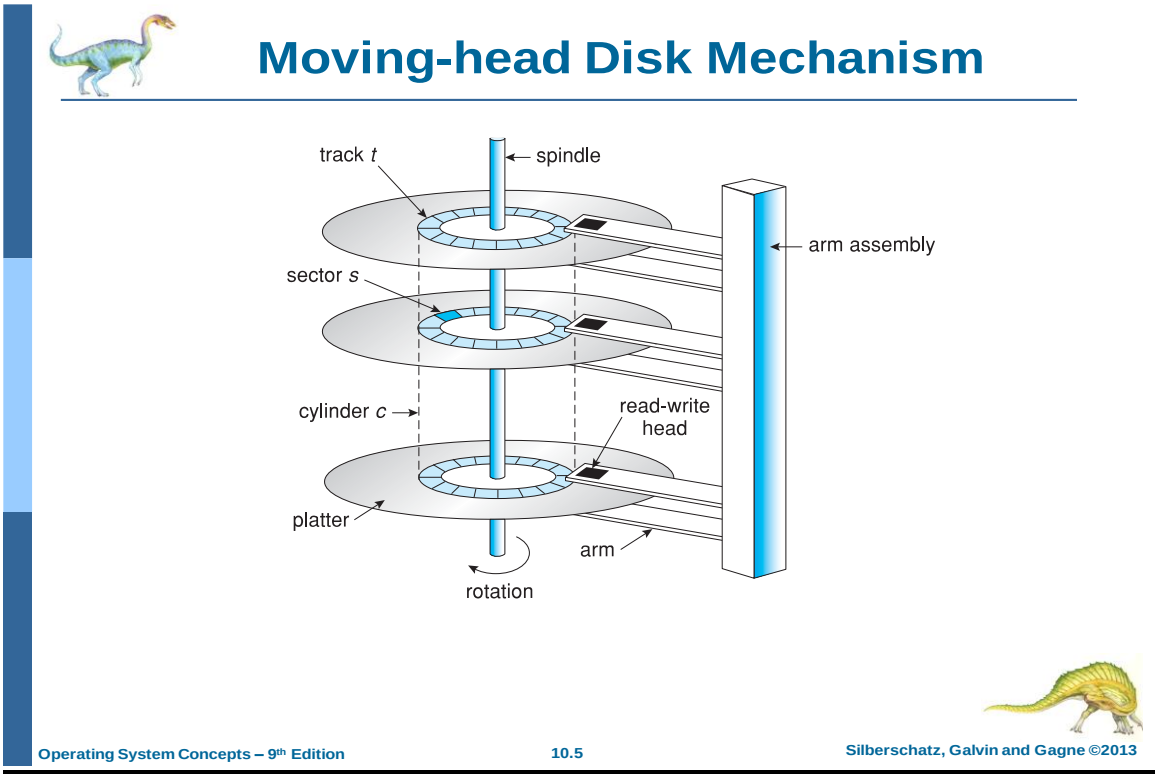
we use the pointer in the i^{th} index block entry and read the desired block.

d. To write into the i^{th} block, a free block is obtained from the free space manager and its address is put in the i^{th} index block entry.

e. Indexed allocation supports direct access without suffering external fragmentation.

Limitations: The pointer overhead of index block is greater than the pointer overhead of linked allocation. So here more space is wasted than linked allocation. In indexed allocation, an entire index block must be allocated, even if most of the pointers are nil.

Disk Scheduling



As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Important terms related to disk scheduling.

Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

Transfer Time

It is the time taken to transfer the data.

Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughout
- Minimal traveling head time
- To minimize the seek time

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

- **FCFS scheduling algorithm**
- **SSTF (shortest seek time first) algorithm**
- **SCAN scheduling**
- **C-SCAN scheduling**
- **LOOK Scheduling**
- **C-LOOK scheduling**

FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

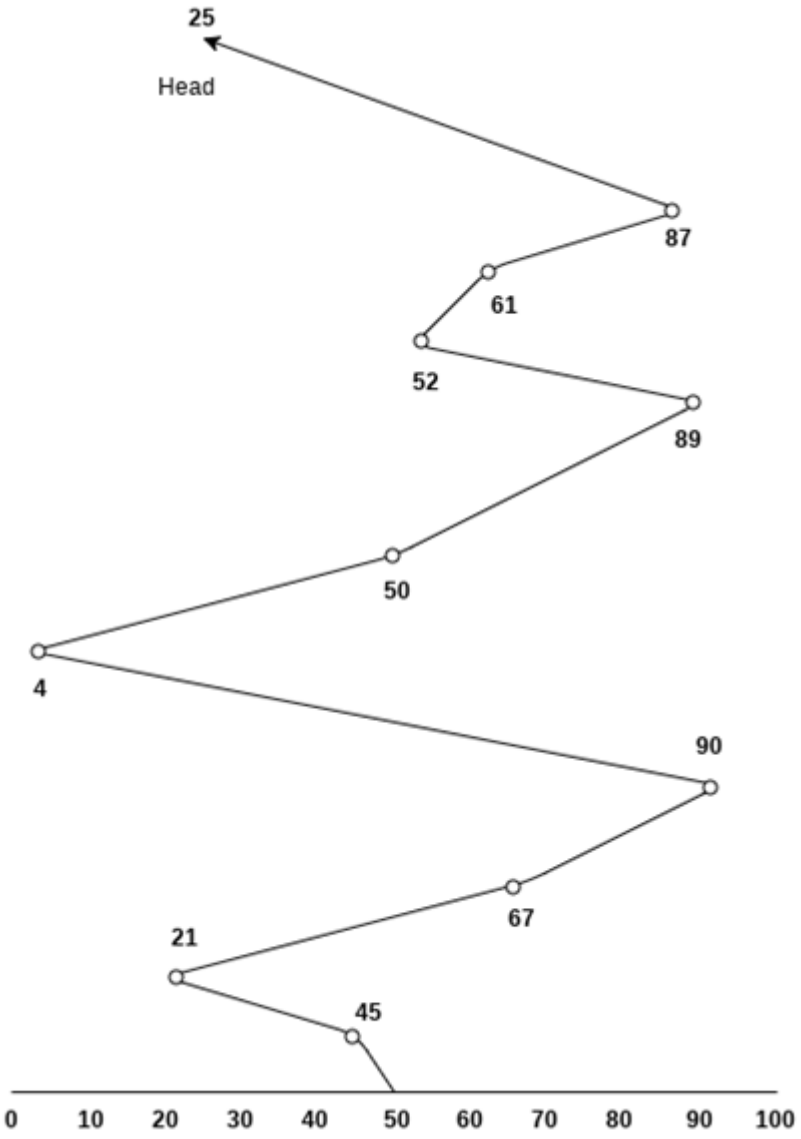
Disadvantages

- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

SSTF Scheduling Algorithm

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It allows the head to move to the closest track in the service queue.

Disadvantages

- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

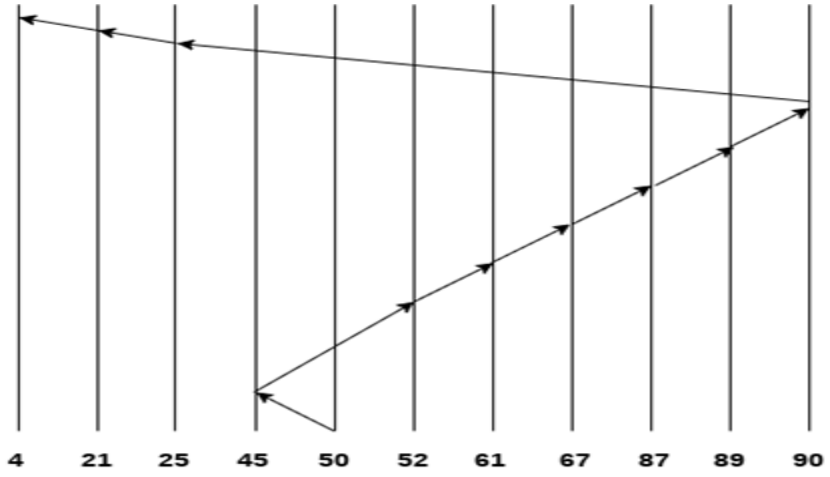
Example

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

Solution:



Number of cylinders = (50-45)+(52-45)+(61-52)+(67-61)+(87-67)+(89-87)+(90-89)+(90-25)+(25-21)+(21-4)

5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

SCAN Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path,and then it turns backand moves in the reverse direction satisfying requests coming in its path.

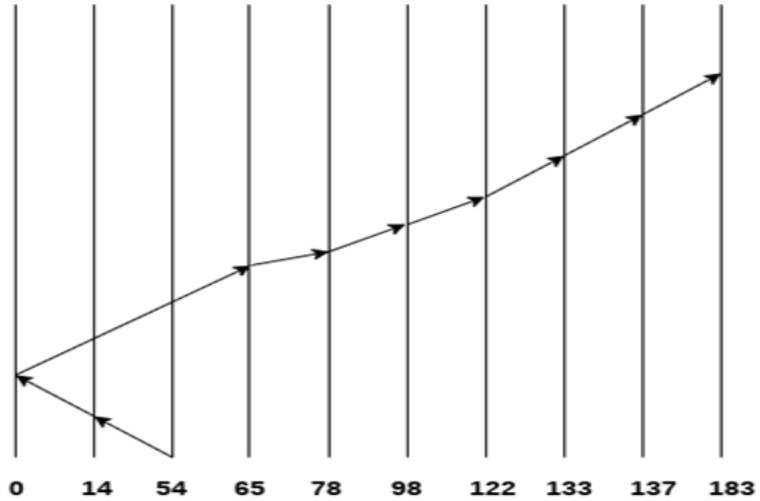
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Example

Consider the following disk request sequence for a disk with 200 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

C-SCAN algorithm

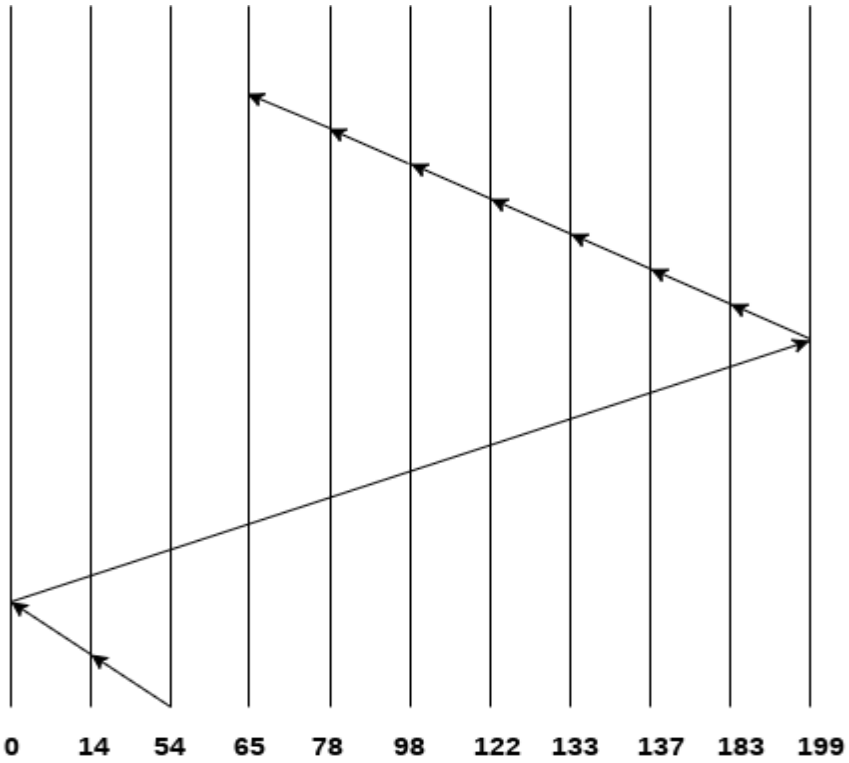
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Example

Consider the following disk request sequence for a disk with 200 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

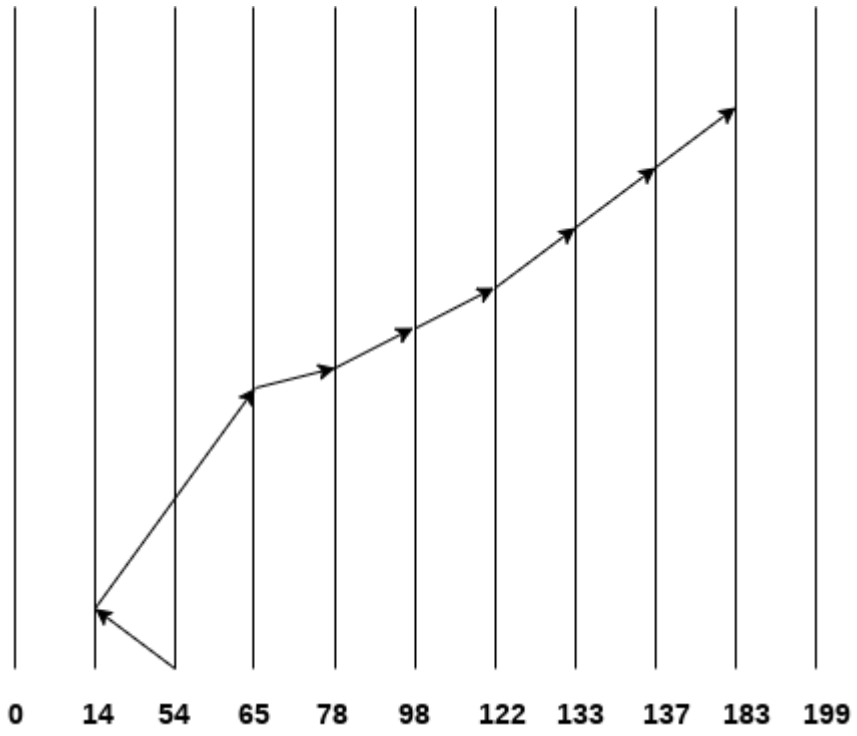
Look Scheduling

It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

Example

Consider the following disk request sequence for a disk with 200 tracks

98, 137, 122, 183, 14, 133, 65, 78. Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using LOOK scheduling.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209

C Look Scheduling

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

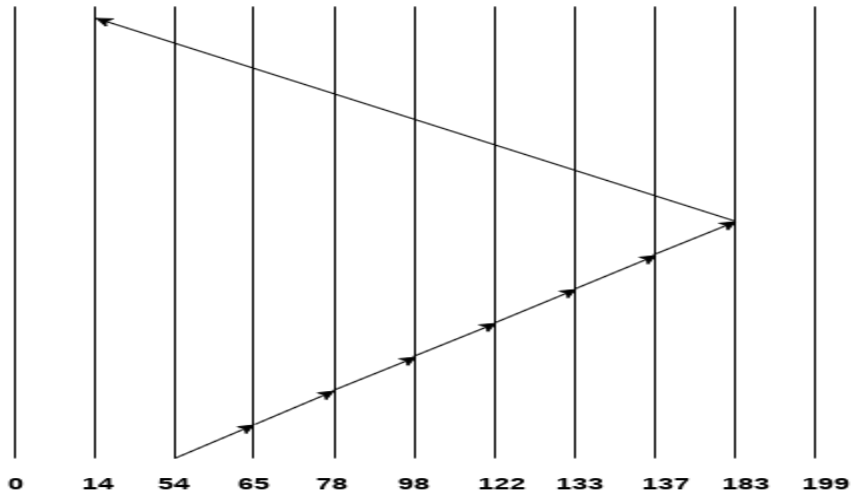
It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

Example

Consider the following disk request sequence for a disk with 200 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in right direction. Find the number of head movements in cylinders using C LOOK scheduling.



Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298

Problem:

1. A Hard Disk has 63 sectors per track, 10 platters each with two recording surfaces and 1000 cylinder. The address of a sector is given as a triple <C, H, S> where C is cylinder number, H is the surface number and S is the sector number. Thus the 0^{th} sector address will be <0, 0, 0> and 1^{st} sector as <0, 0, 1> and so on. What is the total sector number corresponding to the address<400,16,29>

   Ans:

   #sector=63

   #platter =10

   #surface=2

   So to reach cylinder 400(i.e 401^{th} cylinder), we need to skip =400*(10*2)*63=504,000 sectors.

   Then to skip to the 16^{th} surface of the cylinder numbered 400, we need to skip another= 16*63=1,008 sector.

   Finally to find the 29^{th} sector, we need to skip another 29 sector.

   So, total sectors, we moved= 504,000+1, 008+29=50, 5037

2. An application loads 100 libraries at the startup loading each library requires exactly one disk access. The seek time of the disk to random location is given as 10ms.Rotational speed of the Disk is 6000rpm.If all 100 libraries are loaded from random locations on the disk, how long does it take to load all libraries?
   (The time to transfer data from the disk block once the head has taken positioned at the start up the block may be neglected)

Ans:

Disk transfer time=seek time +transfer time+ rotational latency

Here transfer time=0

Seek time=10msec.

Rotational speed =6,000rpm

So, time to rotate once=60/6,000 sec.

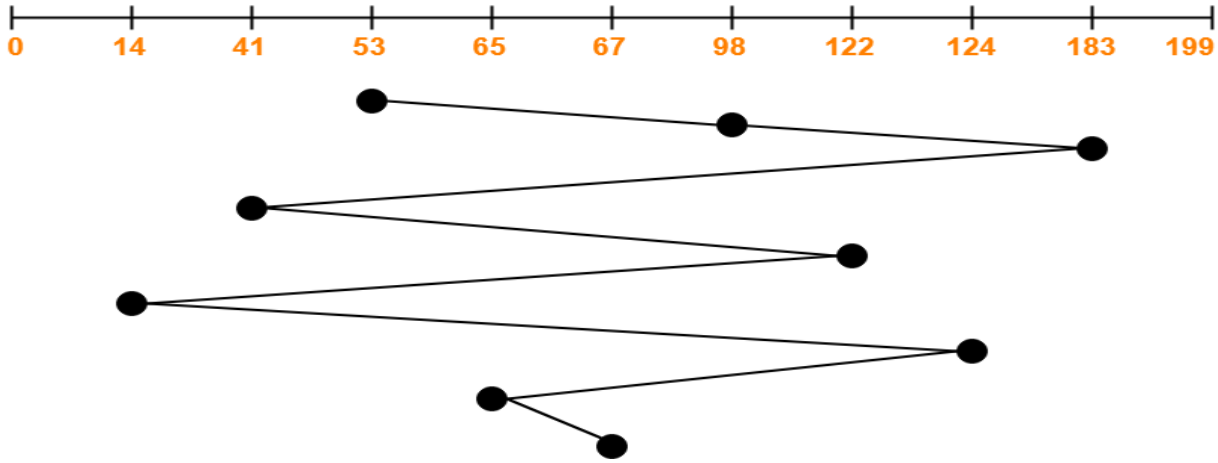
Rotational latency=1/2 *60/6,000 sec=5 ms

So, total time to transfer one library =10+5=15 msec

So, for 100 libraries=100*15=15,00msec=1.5sec.

Problem-3

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.



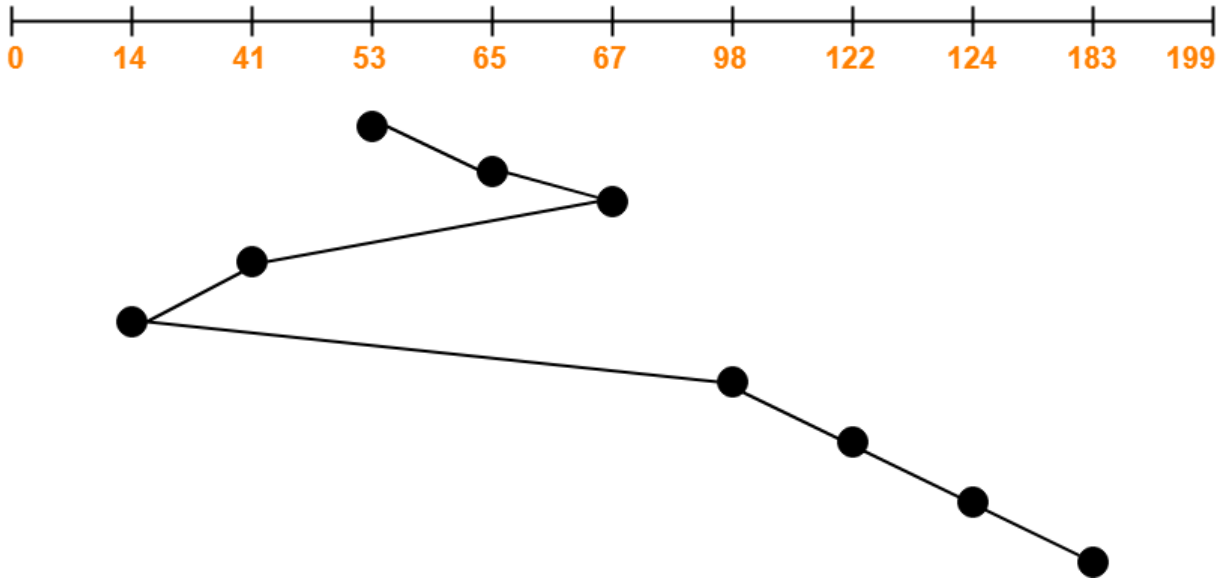
Total head movements incurred while servicing these requests

= (98 – 53) + (183 – 98) + (183 – 41) + (122 – 41) + (122 – 14) + (124 – 14) + (124 – 65) + (67 – 65)

= 45 + 85 + 142 + 81 + 108 + 110 + 59 + 2= 632

Problem-04:

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SSTF scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.



Total head movements incurred while servicing these requests

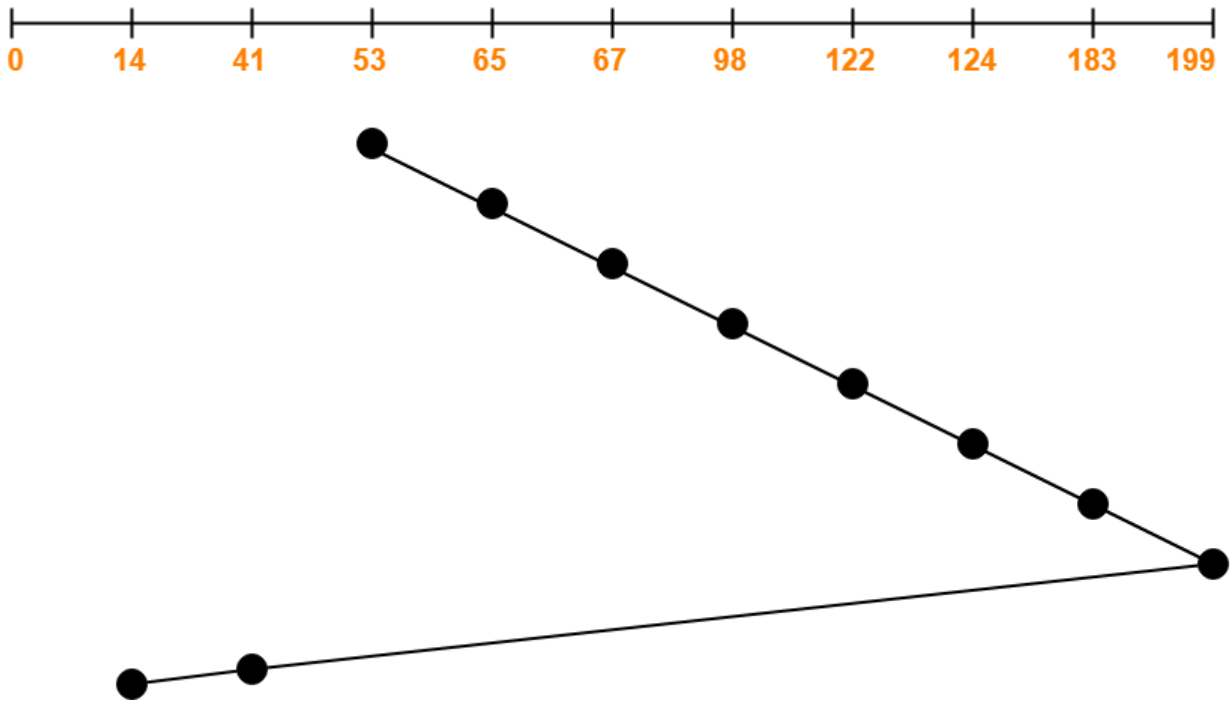
= (65 – 53) + (67 – 65) + (67 – 41) + (41 – 14) + (98 – 14) + (122 – 98) + (124 – 122) + (183 – 124)

= 12 + 2 + 26 + 27 + 84 + 24 + 2 + 59

= 236

Problem-5

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.



Total head movements incurred while servicing these requests

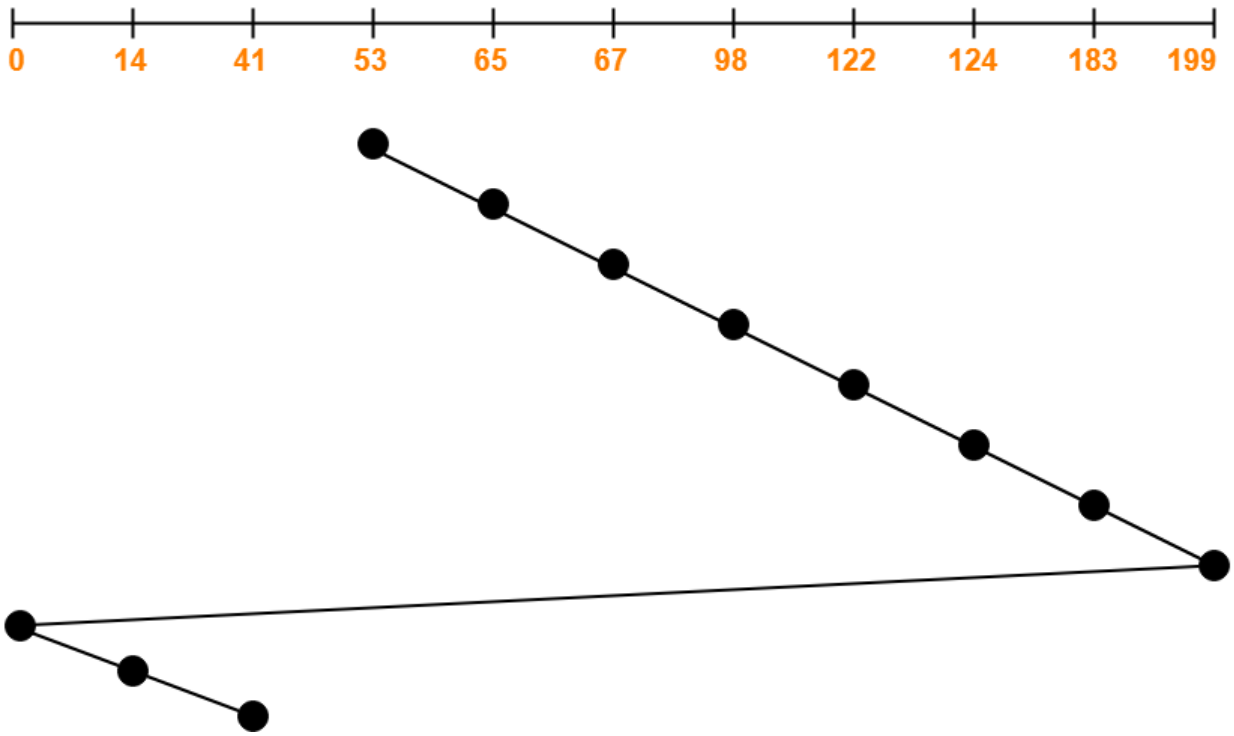
= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 41) + (41 – 14) =

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 158 + 27

= 331

Problem-6

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.



Total head movements incurred while servicing these requests

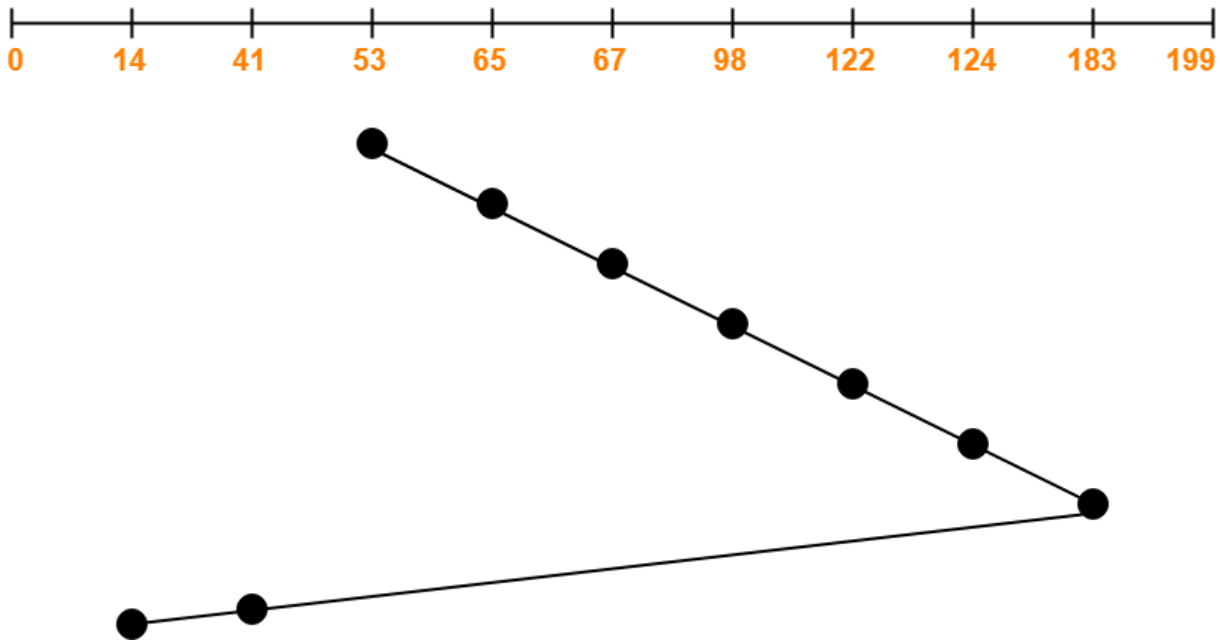
= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 0) + (14 – 0) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 27

= 386

Problem-7

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.



Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 41) + (41 – 14)

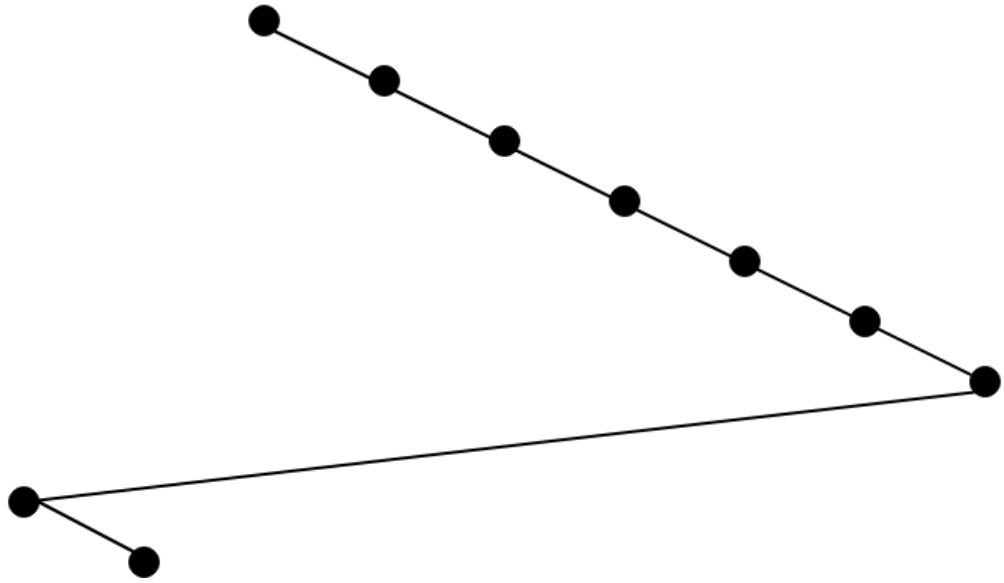
= 12 + 2 + 31 + 24 + 2 + 59 + 142 + 27

= 299

Problem-08:

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199.

The total head movement (in number of cylinders) incurred while servicing these requests is _____.



Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 14) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 27

= 326