UNIT-IV

File System- Interface

Concept of a file

→ File: (A file is a collection of related information and data that is recorded on a secondary storage)

→ (A file is the smallest allotment of logical secondary storage.)

→ (Data cannot be stored on secondary storage, unless they are within a file)

→ (Files represent programs (both source and object forms) and data.)

→ (Data files may be numeric, alphabetic, alphanumeric or binary.)

→ (A file is a sequences of bits, bytes, lines of records) the meaning of which is defined by its creator.

Types of file

1. Text file: It is a sequence of characters organized into lines.
2. Source file: It is a sequence of functions organized into declarations and executable statements.
3. Executable file: It is a series of code sections that the loader can bring into memory and execute.

File attributes:

→ File attributes are the properties of parameaters of a file. that are used to keep track of files in the operating system.

→ File attributes may vary from one operating system to another.

→ The following are the different attributes of a file

1. Name: It indicates the name of the file in human readable form.

2. Identifier: It is the unique tag, usually a number used to identify a file with in the system.

3. Type: It indicates the type of the file. It is needed for systems that support different types of files.

4. Location: It indicates a pointer to a device and to the location of file on that device. 1,2,3,5.

5. Size: It indicates current size of the file (in bytes, words or blocks).

6. Protection: It contains the access control information that determines who can do reading, writing, executing and so on.

7. Time, date and User identification: This information may be kept for creation, last modification and last use. These data can be useful for protection, security and usage monitoring.

File operations

→ The operating system provides various system calls to perform operation on a file.

→ The following operations are performed on a file.

1. Creating a file: There are two steps necessary for creating a file. First, space in the file system must be found for the file. Second an entry for the new file must be made in the directory.

2. Writing a file: To write a file, the os make a system call by specifying both the name of the file and the information to be written to the file.

3. Reading a file: To read a file, the operating system makes a system call by specifying the name of the file.

4. Repositioning within a file: The directory is searched for the appropriate entry and the current file position pointer is repositioned to a given value. This file operation is also known as a file seek.

5. Deleting a file: To delete a file, the directory is searched for the named file; then all file space is released so that it can be reused by other files and erase the entry in the directory.

of a file but keep its attributes. This function allows all attributes to remain unchanged except the file length.

The directory is

(3) Access methods

Allocation methods

There are three main file allocation methods on disk.

1. Contiguous allocation
2. Linked allocation
3. Indexed allocation

contiguous allocation

→ In contiguous allocation, each file occupies a set of contiguous blocks on disk.

→ Accessing a file that has been allocated contiguously is easy.

→ One difficulty with contiguous allocation is finding space for a new file.

→ First fit and best fit strategies are used to select free disk blocks from the set of available disk blocks.

→ Another major problem with this allocation method is that determining how much space is needed for a file.

→ when the file is created, the total amount of space it needs must be found and allocated.

→ If we allocate too little space to a file, we may find that file cannot be extended. Especially with best-fit allocation strategy.

→ There are two possibilities in such a case. First, the user program is terminated with error message, and then allocate more space and run the

→ The other possibility is to find a larger hole, copy the contents of the file to the new space, and release the previous space.

→ External fragmentation exists in this allocation method

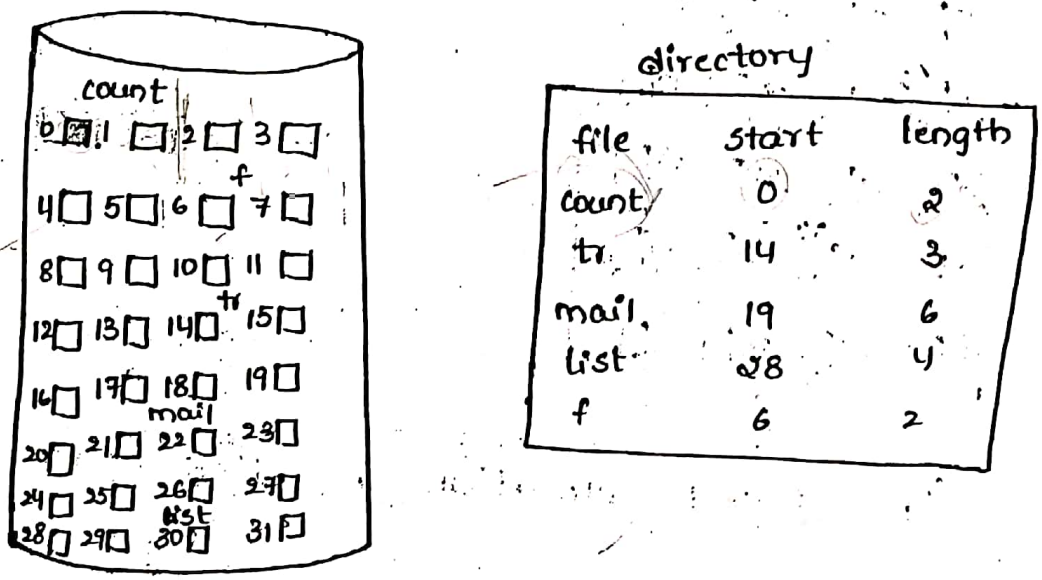


fig:- contigous allocation of disk space

Linked allocation

→ Linked allocation solves all problems of contiguous allocation

→ With linked allocation, each file is a linked list of disk blocks, the disk blocks are scattered on the disk.

→ The directory contains a pointer to the first and last blocks of the file.

→ When a file is created, an entry in the directory is created.

→ With linked allocation, accessing a disk block randomly is difficult.

→ Another problem disadvantage to linked allocation is the space required for the pointers.

→ There is no external fragmentation with linked allocation

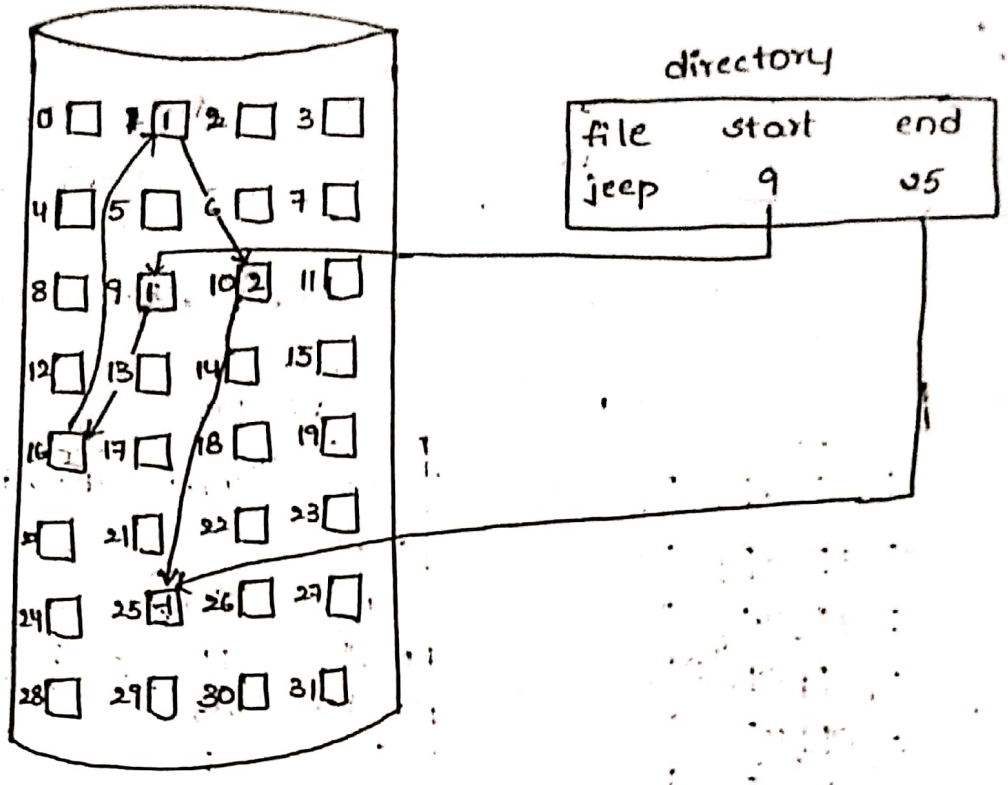


fig:- linked allocation of disk space

Indexed allocation

→ Linked allocation solves the external fragmentation and size declaration problems of contiguous allocation.

→ Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered all over the disk and need to be retrieved in order.

→ Indexed allocation solves this problem by bringing all pointers together into one location, the index block.

→ 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.

→ The directory contains the address of the index block.

→ Indexed allocation supports direct access without sufferi from external fragmentation. But it suffers from wasted space.

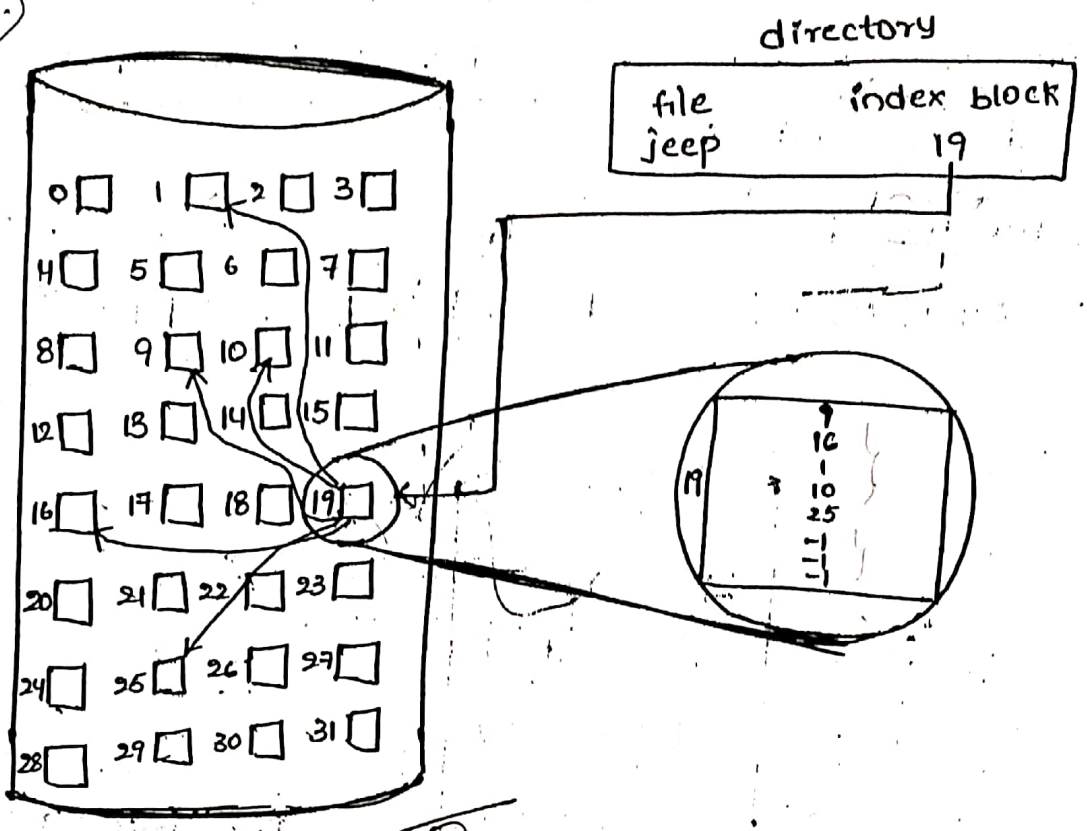


fig: Indexed allocation of disk space

4(b)

Free space Management

→ To keep the track of free disk space, the operating system maintains a free space list.

→ The free space list ~~contains~~ is a collection of all free disk blocks, that are not allocated to a file or directory.

→ When a new file is created, the free space list is search to allocate space and free space list is updated.

→ The free space list can be implemented in the following wa

1. Bit vector or Bit map.
2. Linked list
3. Grouping.
4. Counting

① Bit Vector

→ The free space list is implemented as a bit map or bit vector

→ Each block is represented by 1 bit.

→ If the block is free, the bit is 1; and if the block is allocated, the bit is 0.

→ Consider the following disk, the free space bit map would be

0 1 0 1 1 1 0 | 0 0 0 | 1 1 0 | 1 1

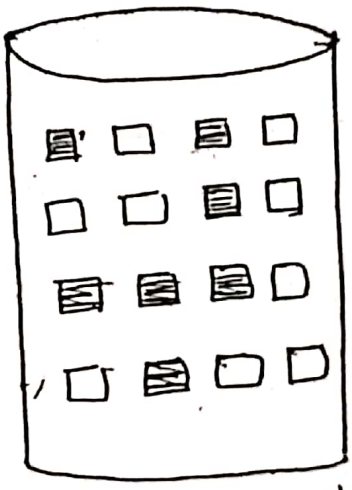


Fig: Disk using bit map.

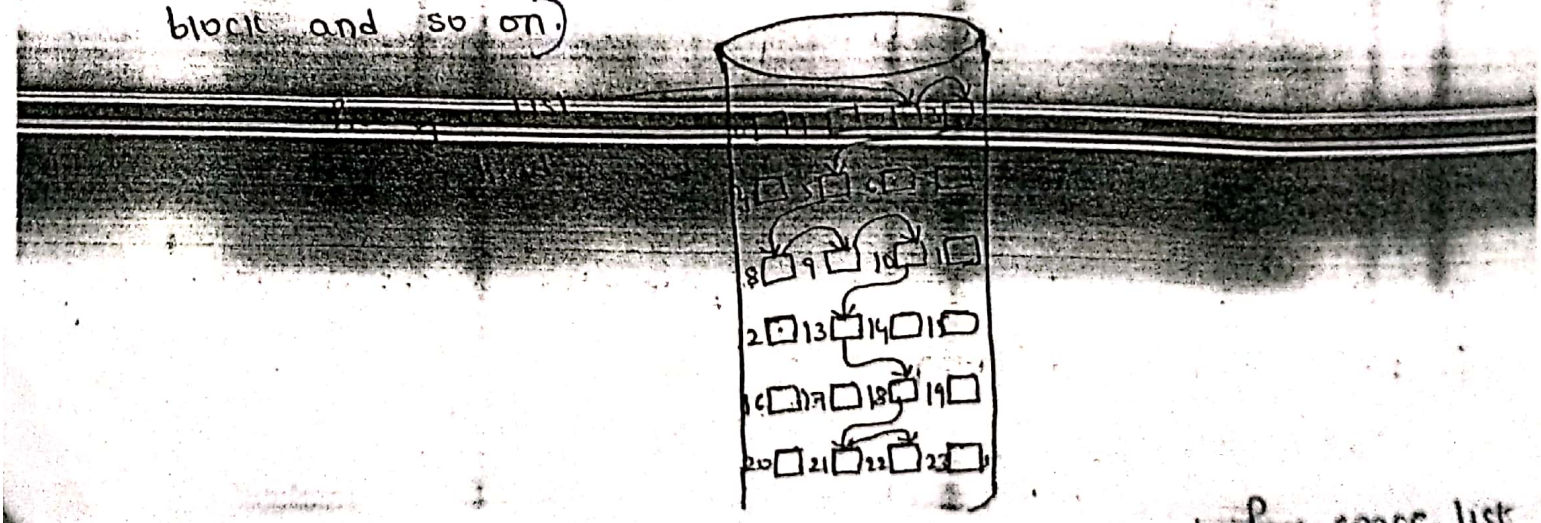
Free space list on

→ The main advantage of this approach is its simplicity and efficiency.

② Linked list

→ (Another approach to free space management is to link together all the free disk blocks) keeping a pointer to the first free block.

→ (This first block contains a pointer to the next free disk block and so on.)



3. Grouping:

→ In this method, the first free block stores the addresses of n free blocks.

→ The first n-1 of these blocks are actually free.

→ The last block contains the addresses of another n free blocks. and so on.

→ The addresses of a large number of free blocks can be found quickly.

4. Counting:

→ This method keeps the address of the first free block and the number (n) of free contiguous blocks that follow the first block, instead of keeping a list of n free disk addres

→ Each entry in the free space list consists of a disk addr address and a count.

4(b)

Disk scheduling

Seek time: The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.

Rotational latency: The rotational latency is the additional time for the disk to rotate the desired sector to the disk hea

The disk

Disk bandwidth: The disk bandwidth is the total number of by transfered, divided by the total time between the first request for service and the completion of the last transfe

The disk band width contains the number of bytes transferred

4(a) Disk scheduling

→ The following are the disk scheduling algorithms

1. FCFS scheduling (First come, First served)
2. SSTF scheduling (shortest seek time first)
3. SCAN scheduling
4. C-SCAN scheduling
5. Look scheduling

FCFS Scheduling

→ The simplest form of disk scheduling is First come First serve (FCFS).

→ The tracks are processed in sequential order.

→ This algorithm is simple, but it does not provide the fastest service.

→ Consider, for ocampk, a disk queue with requests for I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67 → cylinder nos

→ If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, 6

→ The problem with this is that the [illegible] decreased

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

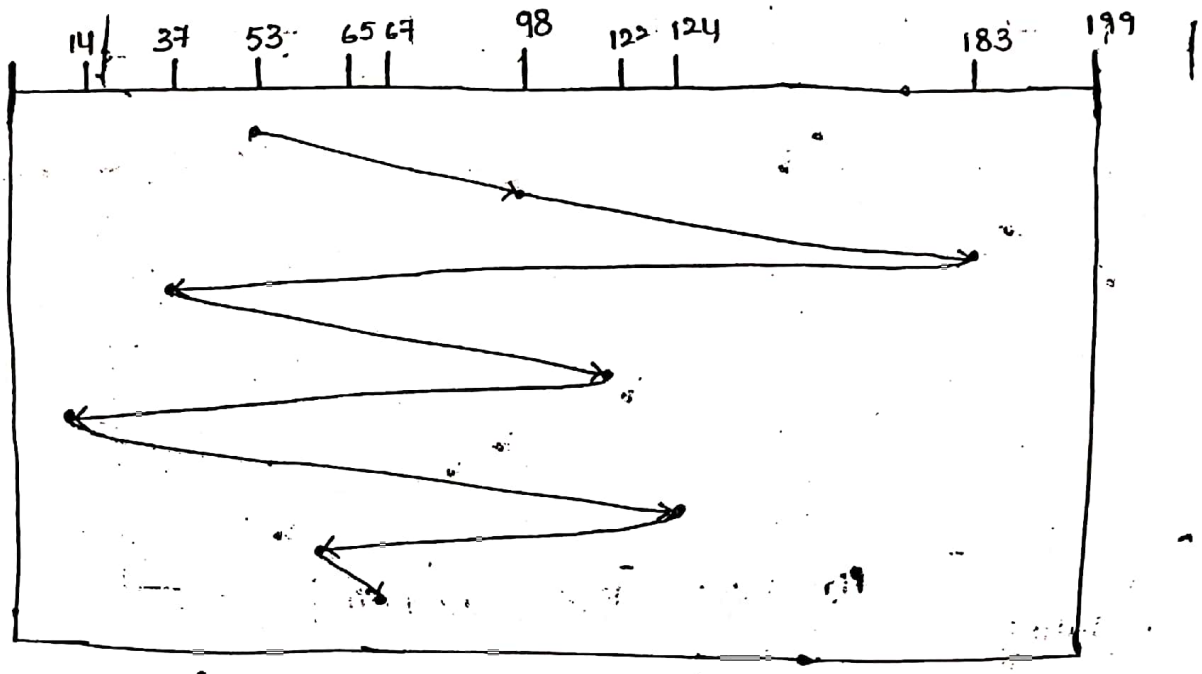


fig: FCFS disk scheduling

② SSTF scheduling (shortest seek time first)

→ In shortest seek time first algorithm, all the requests that are close to the current head position are serviced first before moving the head far away to service other requests.

→ The SSTF algorithm selects the request with the least seek time from the current head position.

→ It is an improvement over FCFS algorithm, but it is not optimal.

→ The requests that are far away from the head may

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

head starts at 53

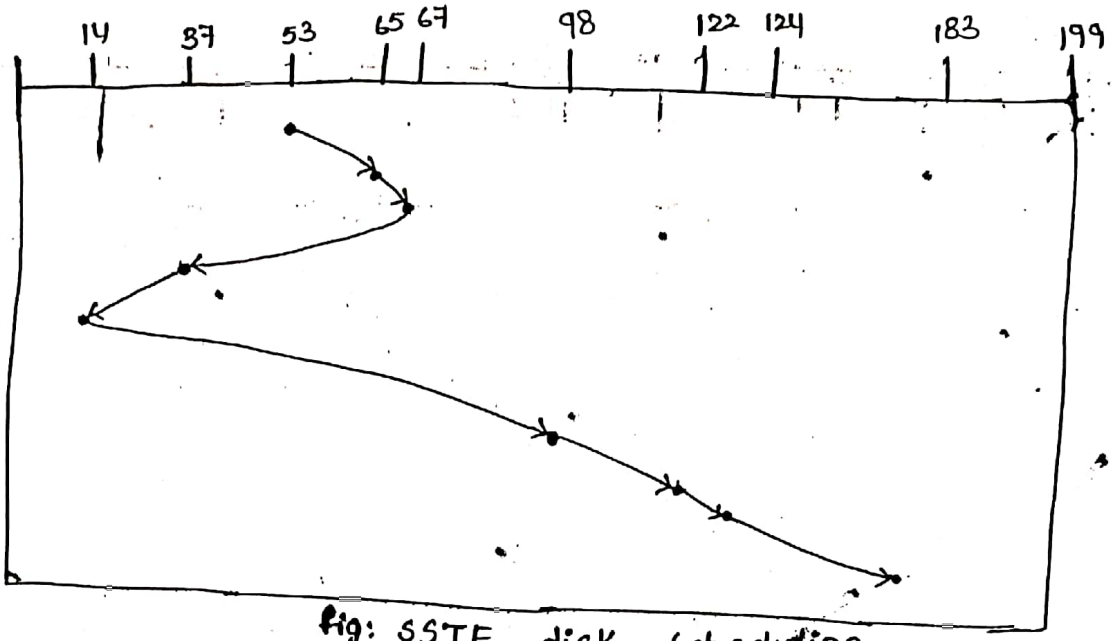


Fig: SSTF disk scheduling

SCAN scheduling

→ In the SCAN algorithm, the disk arm starts at one end of the disk and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.

→ At the other end, the direction of head movement is reversed and servicing continues.

→ The head continuously scans back and forth across the disk.

→ The SCAN algorithm is sometimes called as an elevator algorithm since the disk arm behaves just like an elevator in a building.

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

Head starts at 53

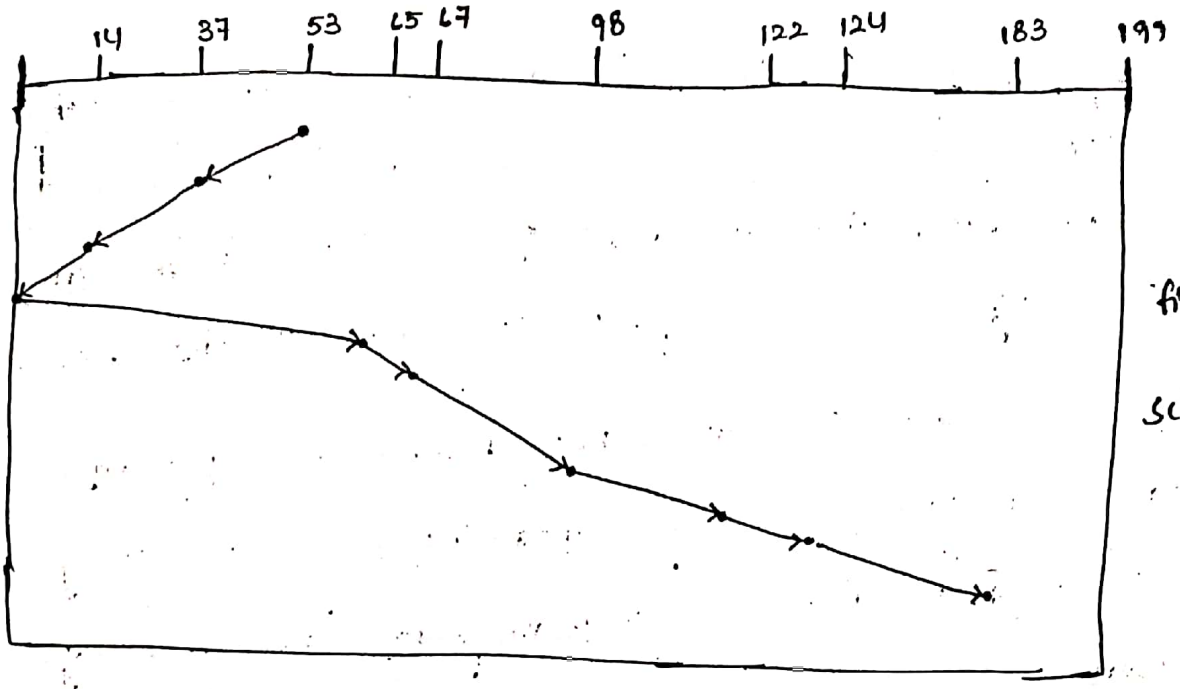


fig: SCAN disk scheduli

C-SCAN scheduling

→ Circular SCAN (C-SCAN) scheduling is a variant of SCAN scheduling designed to provide uniform wait time.

→ Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.

→ When the head reaches the other head, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

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

Head starts at 53

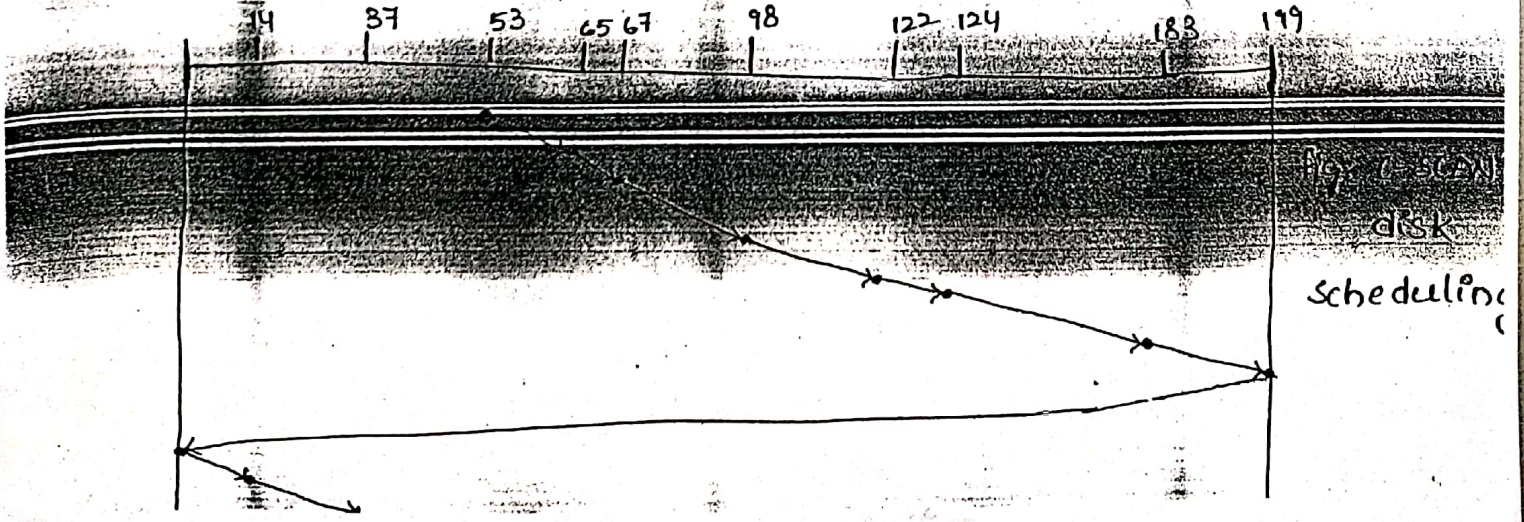


fig: C-SCAN disk scheduling

Look scheduling & C- Look scheduling

→ This algorithm, looks for the requests before moving.

→ The algm goes only as far as the final request in each direction, then it reverses direction immediately without going all the way to the end of the disk.

→ The SCAN and C-SCAN that follow this pattern are called Look and C-Look scheduling, because they look for a request before continuing to move in a given direction.

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

head starts at 53

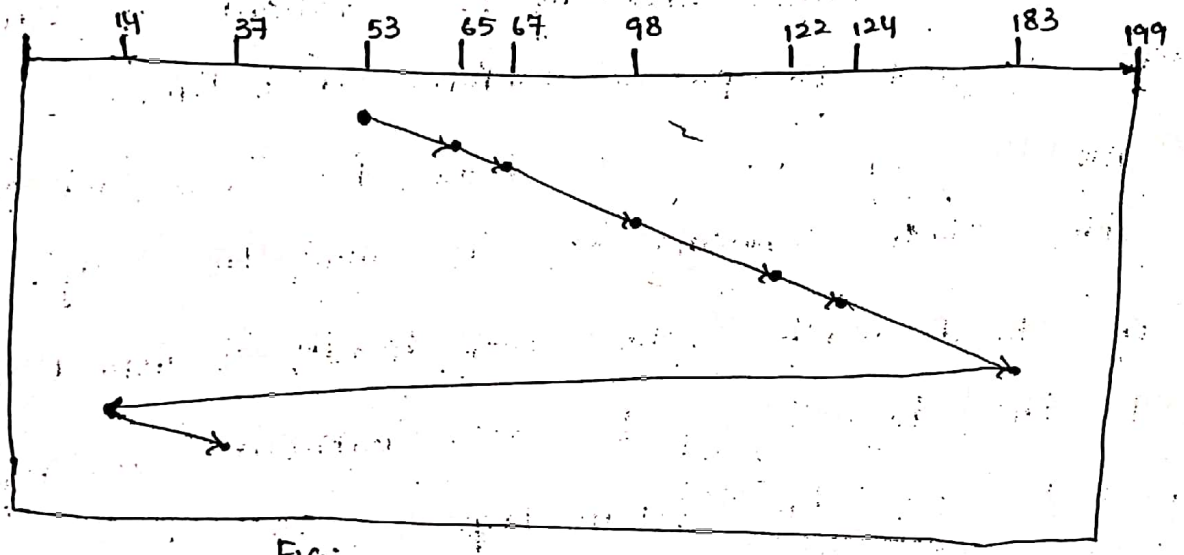


Fig: C- Look disk scheduling

Overview of mass storage structure.

Magnetic disks

→ Magnetic disks are the secondary storage devices.

→ They can store bulk amount of data.

→ Each disk plat (The suface of the disk is called platt

→ (Each platter has a flat circular shape), like a CD.

→ The information is stored by recording it magnetically on the platters.

→ (There is a read-write head on each surface of every platter)

→ (The heads are attached to a disk arm that moves all the heads as a unit)

→ The surface of a platter is logically divided into circular tracks, which are sub divided into sectors.

→ The set of tracks that are at one arm position makes up a cylinder.

→ (There may be thousands of concentric cylinders in a disk drive and each track may contain hundreds of sectors)

→ Storage capacity of common disk drives is measured in giga bytes.)

→ Disk speed has two parts.

1. transfer rate
2. positioning time or random access time
   a) seek time
   b) Rotational latency.

→ The transfer rate is the rate at which data flow between the drive and the computer.

→ The seek time is the time necessary for the disk arm to move to the desired cylinder.

→ The rotational latency is the time necessary for the desired sector to rotate to the disk head.