Unit 5. Device Management

Device Management Function
Device Characteristics
Disk space Management
Allocation and Disk Scheduling Methods

****

What is Device Management?

- Device Management is another important function of the operating system.
- Device management is responsible for managing all the hardware devices of the computer system.
- It may also include the management of the storage device as well as the management of all the input and output devices of the computer system. It is the responsibility of the operating system to keep track of the status of all the devices in the computer system.
- The status of any computing devices, internal or external may be either free or busy.
- If a device requested by a process is free at a specific instant of time, the operating system allocates it to the process.
- An operating system manages the devices in a computer system with the help of device controllers and device drivers. Each device in the computer system is equipped with the help of device controller. For example, the various devices controllers in a computer system may be disk controller, printer controller, tape-drive controller and memory controller.
- All these devices controllers are connected with each other through a system bus. The device controllers are actually the hardware components contains some buffers registers to store the data temporarily. The transfer of data between a running process and the various devices of the computer system is accomplished only through these devices controllers.

Device management Functions

- As the two main job of computer are input/output and processing. So that it become essential to know the role of an operating system in managing and controlling the I/O operations and I/O devices.
- OS manages device communication via their respective drivers.
- The **device management functions** that must be performed by an operating system are:
  1. Track the status of each device such as disk drive, printer, plotters and terminals.
  2. Use the algorithm to decide which process will get a device and for how long.
  3. Allocate the devices.
  4. De-allocate devices

At two level deallocation

1. At command Level: When the I/O command has been executed and the device has been temporary released.
2. At Process Level: When the process has been terminated and the device has been permanently released.

Block and Character Devices

All I/O devices are classified as either character or block devices.

Block Device

- Block devices data is transferred in a block of bytes and required a buffering mechanism to allow faster input and output operations.

- Block devices accept input and output only in blocks.
- They are usually storage devices that provide reading and writing operation of data in fixed-size blocks and also reading or writing the entire block at a time.
- Most file systems are based on block devices where data is stored and retrieved in blocks of bytes.
- Since the files length (size) usually occupies multiples of block size, and as such; a single block may only contains a part of a single file, such scenario lead to internal fragmentation.
- Memory access is required for file access, where the files need to be mapped in memory, and speed difference between the memory and the block devices can create performance problems.
- Some examples of such devices are: hard drives, floppy disks, and optical drives such as DVD-ROM, and CD-ROM.

Character Device

- Character devices (also called stream of bytes devices) transfer data in a few streams of bytes. Such devices don't required buffering mechanism, and they don't operate with a fixed block size and read and write character immediately.
- Character devices transmit data one character at a time. Such devices provide a stream of communication.
- The response time and the processing speed are faster than the block devices.
- Streaming devices used less memory than the block devices, since the stream devices required less data to be processed at a time while block devices required an access to a block of data at a time.
- Some examples of such devices are: Keyboard, Monitor, Printer, etc…

Blocking and Non-Blocking I/O

Blocking I/O

- ✓ Some control over how the wait for I/O to complete is accommodated is available to the programmer of user applications.
- ✓ Most I/O requests are considered blocking requests, meaning that control does not return to the application until the I/O is complete.
- ✓ The delayed from systems calls such read() and write() can be quite long.
- ✓ Using systems call that block is sometimes call synchronous programming.
- ✓ In most cases, the wait is not really a problem because the program cannot do anything else until the I/O is finished. However, in cases such as network programming with multiple clients or with graphical user interface programming, the program may wish to perform other activity as it continues to wait for more data or input from users.

Non- Blocking I/O

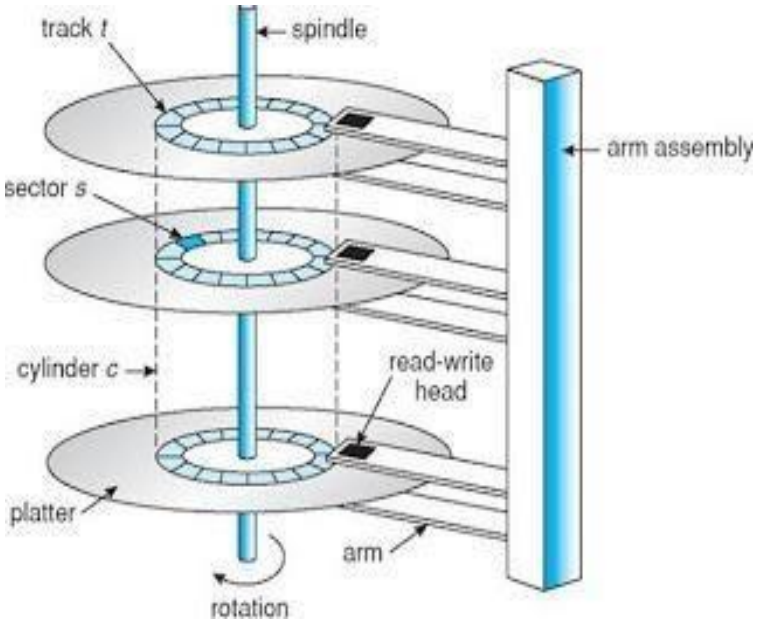
- ✓ To use multiple threads so that one part of the program is not waiting for unrelated I/O to complete.
- ✓ Another alternative is to use asynchronous programming techniques with no blocking system calls.
- ✓ An asynchronous call returns immediately, without waiting for the I/O to complete.
- ✓ The completion of the I/O is later communicated to the application either through the setting of some variable in the application or through the triggering of a signal or call-back routine that is executed outside the linear control flow of the application.

Characteristics of I/O Devices or Device Characteristics

1. Data Transfer Mode
   - ✓ Character Device
     e.g. Terminal
   - ✓ Block Device
     e.g. Disk
2. Access Method
   - ✓ Sequential
     e.g. Modem
   - ✓ Random
     e.g. CD-ROM
3. Transfer Schedule
   - ✓ Synchronous
     e.g. tape
   - ✓ Asynchronous
     e.g. Keyboard
4. Sharing
   - ✓ Dedicated
     e.g. Tape
   - ✓ Sharable
     e.g. Keyboard
5. Device Speed
   - ✓ Latency
   - ✓ Seek Time
   - ✓ Transfer Rate
   - ✓ Delay Between Operation
6. I/O direction
   - ✓ Read Only
     e.g. CD-ROM
   - ✓ Write Only
     e.g. Graphics Controller
   - ✓ Read-Write
     e.g. Disk

Disk Space Management

The disk space manager is the lowest level of software in the DBMS architecture, with manages space on disk. In short, the disk space manager supports the concept of a page as a unit of data, and provides commands to allocate or deallocate a page and read or write a page. the size of a page is chosen to be the size of a disk block and pages are stored as disk blocks so that reading or writing a page can be done in one disk Input/output.

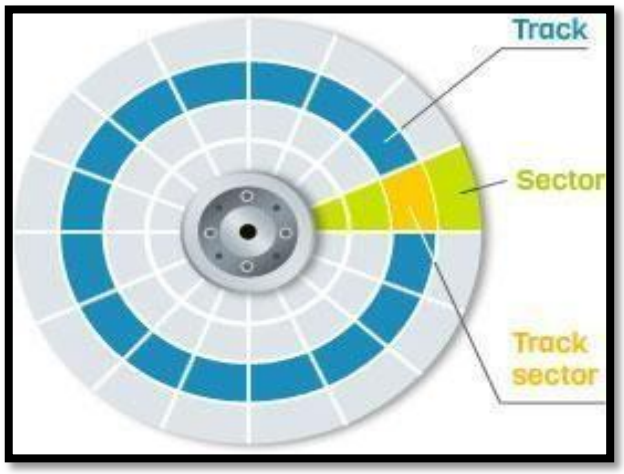


It is often useful to allocate a sequence of page as a contiguous sequence of blocks to hold data that is frequently accessed in sequential order this capability is essential for exploiting the advantages of sequentially accessing disk blocks. Such a capability, if desired, must be provided by the disk space manager to higher - level layers of the DBMS.

Thus, the disk space manager hides details of the underlying hardware (and possibly the operating system) and allows higher levels of the software to think of the data as a collection of pages.

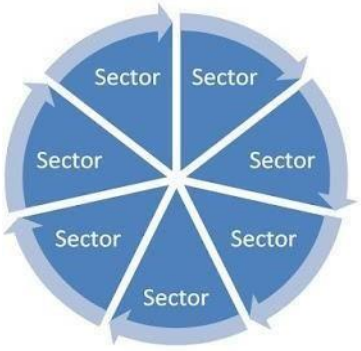
Track:

- A disk surface contains several concentric tracks. In fact, a track contains data which can be read by signal read head without changing its position.
- The maximum amount of information that can be read by a single read/write head system in one revolution is determined by the track length.
- The track length is expressed in bytes, words or characters.
- Each track, in addition to the data, contents some extra information such as : the address of tracks (i.e., cylinder number and track number), block number, gap between blocks cyclic check code etc. information are used by Input/output controls for proper processing of data.



Sectors:

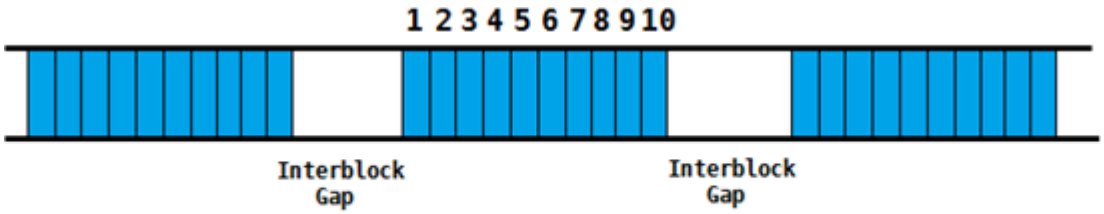
- The tracks are sub-divided into smaller regions called sectors.
- A sector is the smallest addressable segment (part) of a track.
- The division of a track into sectors is done by hardware or software format operation.



Blocks:

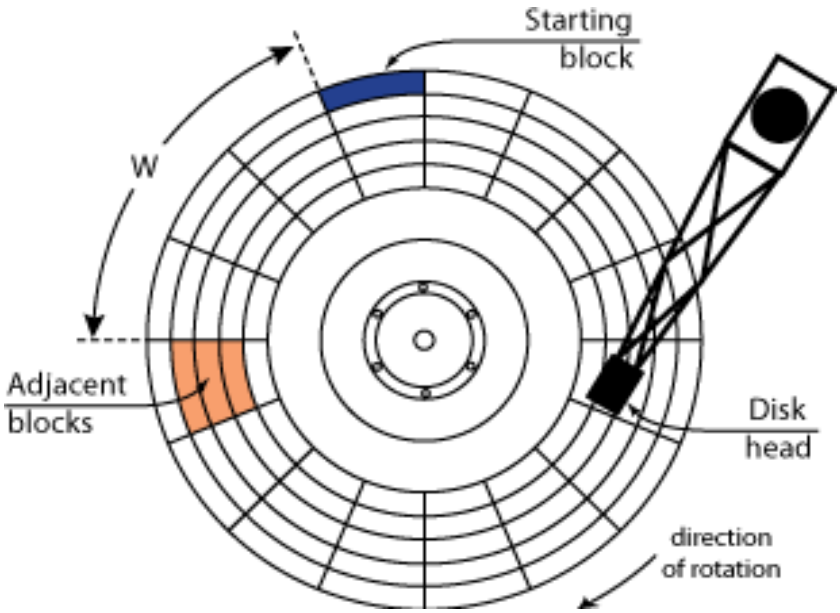
Since the entire track length is too large for data to be transferred by a single Input/Output command, the data is stored on the track in a number of blocks of equal length.

- A block may be equal to a sector or a number of sectors. Infact, the block size determines the basic unit of data which is read or written by a single Input/Output command. The blocks are separated by a gap (G) and this gap reduces this storage capacity of data.
- A small block size increases the number of gaps thereby causing wastage of storage space. Very large blocks on the other hand create problems for processor requiring larger main memory in which data is to be transferred.



BLOCK of RECORDS

- In order to refer to a particular block, an address identifying the track and the position of the block on the track is constructed.
- This block identification address is known as a **block pointer**, which is donated by P.
- The block pointer (i.e., the pointer to a block) can be its absolute address consisting of the cylinder number, surface number, sector or block number etc.



Allocation and Disk Scheduling Methods

❖ **File Allocation Method**

The allocation methods define how the files are stored in the disk blocks. There are three main *disk space or file allocation methods*. How to allocate space to these files so that

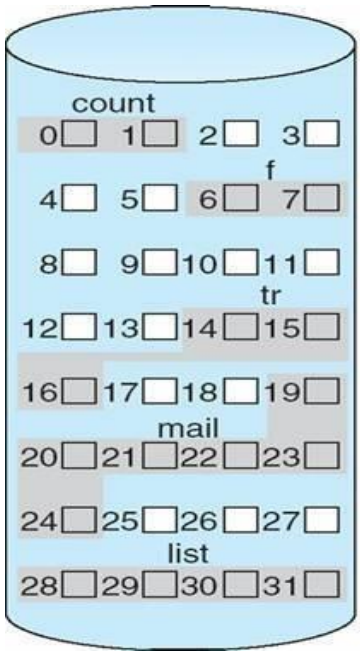
- disk space is utilized effectively
- files can be accessed quickly

Main allocation methods in use are:

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

1. Contiguous Allocation:

- This allocation method requires each file to occupy a set of contiguous blocks on the disk.
- Contiguous allocation of file is defined by disk address and length [In Block Unit]



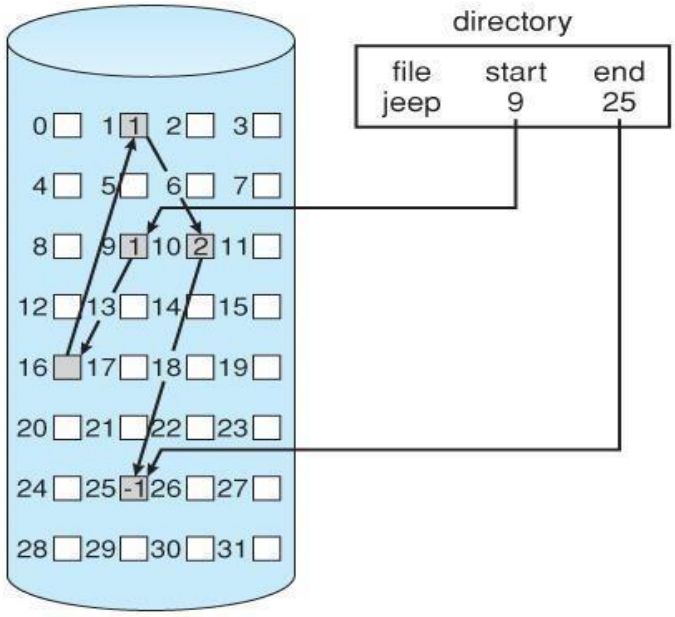
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- The directory entry for each file indicated the address of starting block and length of area allocated for this file.
- Sequential and direct access method can be supported by contiguous allocation.
- One difficulty with contiguous allocation is finding a space for new file of dynamic storage allocation.
- It is also suffer from external fragmentation.
- As file are allocated and deallocated, the free disk space is broken into pieces. External fragmentation exists whenever a free space is broken into chunks.
- It becomes a problem when the largest contiguous chunk is insufficient for a request, depending on total amount of disk storage and average file size; external fragmentation may be either a minor or major problem.
- We can solve **external fragmentation** using compaction technique.
  - Copy the entire file system onto another floppy disk or tap. The original disk was then free completely.
  - Creating one large contiguous free space then copy the files back onto the floppy disk by allocating contiguous space.
  - That also makes one large contiguous free space chunks.
  - The cost of this compaction is time.
- Another major problem is determining how much space is needed for a file.
- A file that grow slowly over long period of time must be allocated enough space for its final size, even through much of that space may be unused for a long time. This is known as a Internal Fragmentation.

2. Linked Allocation:

- It solves all the problem of contiguous allocation.
- With linked allocation each file is linked list of disk block, the disk block may be scattered anywhere on the disk.



- The directory contains a pointer to the first and last block of file.
- To create a new file, we simply create a new entry in the directory.
- With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. The value -1 may be used for NIL to differentiate it from block 0.
- The size field also set to zero.
- A write to a file removes the first free block and writes to that block. This new block is then linked to the end of the file. To read a file, the pointers are just followed from block to block.

Advantages:

1. There is no external fragmentation with linked allocation. Any free block can be used to satisfy a request.
2. There is no need to declare the size of a file when that file is created.

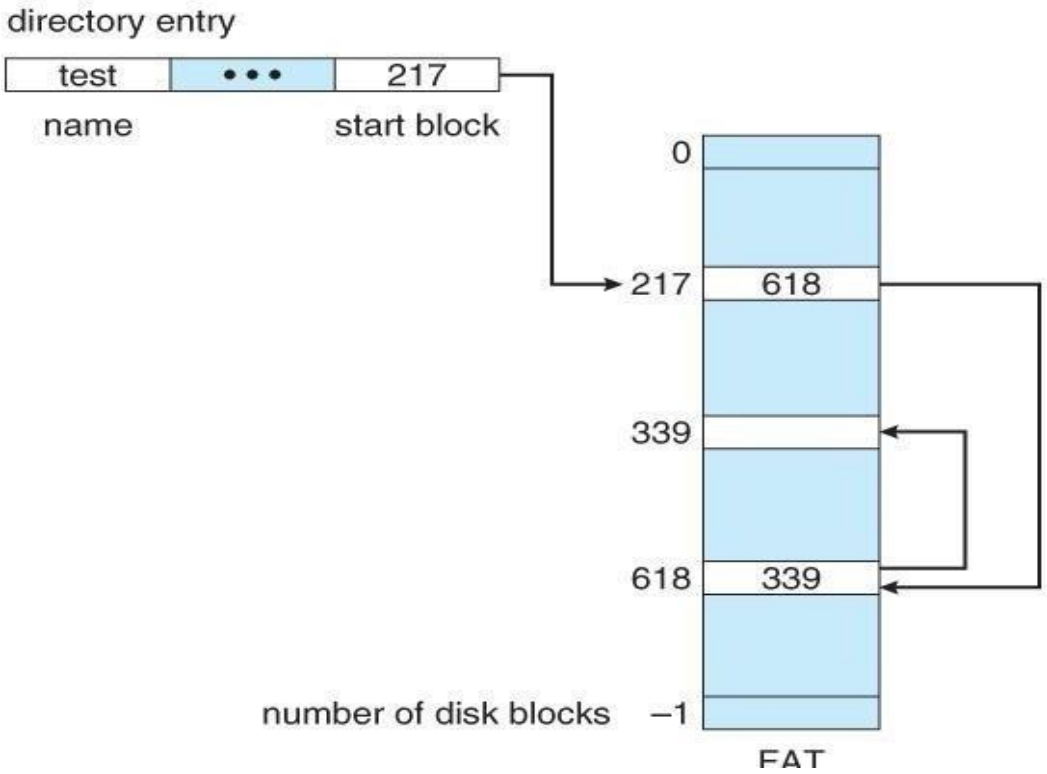
Disadvantages:

1. The major problem is that it is inefficient to support direct-access; it is effective only for sequential-access files. E.g. To find the i^{th} block of a file, it must start at the beginning of that file and follow the pointers until the i^{th} block is reached. Note that each access to a pointer requires a disk read.
2. Space required for the pointer that can be solves using clusters and to allocate clusters rather than blocks.
3. A bug in OS or disk hardware failure might result in pointers being lost and damaged. It also solves using doubly link list.

A section of disk at the beginning of each volume is reserved to store a table, called **FAT**.

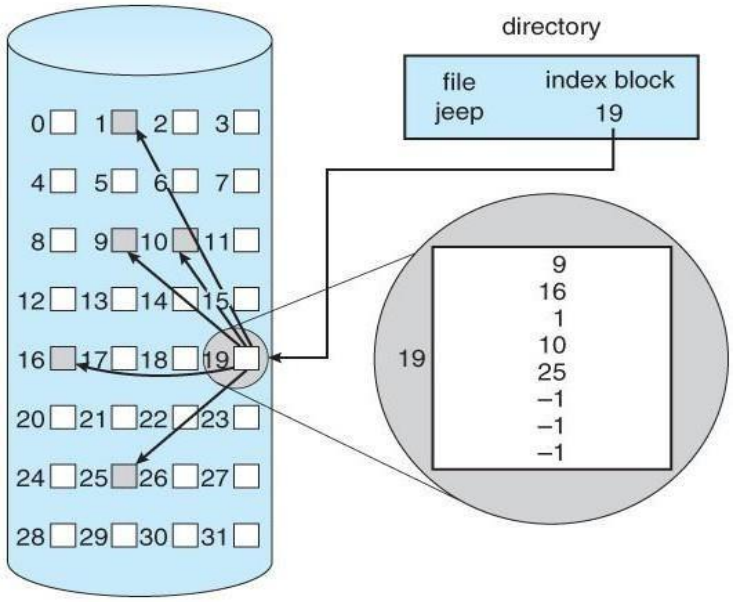
- This table has one entry for each block in the file system.
- It is indexed by block number.
- Then FAT is used just as linked list.
- The only difference is instead of block itself, the FAT has pointers.

An important variation on linked allocation method is the use of file allocation table (FAT).



3. Indexed Allocation:

- Indexed allocation solves the problem by bringing all the pointers together into one location is called indexed block.
- Each file has its own indexed block which is an array of disk block addresses.
- To read the i^{th} block we use the pointer to find and read desire block.
- When the file is created all the pointers in the indexed block are set to NIL.
- When the i^{th} block is first written, a block is obtained from free space manager, and its address is put in the i^{th} indexed block entry.
- It support direct access method without suffering from external fragmentation.
- It suffers from wastage space.
- It also raises the question of how long the indexed block should be.
- Every file must have an indexed block so it should be as small as possible and if it is too small however it will not able to hold enough pointers for large file and mechanism will have to be available to deal with this issue.



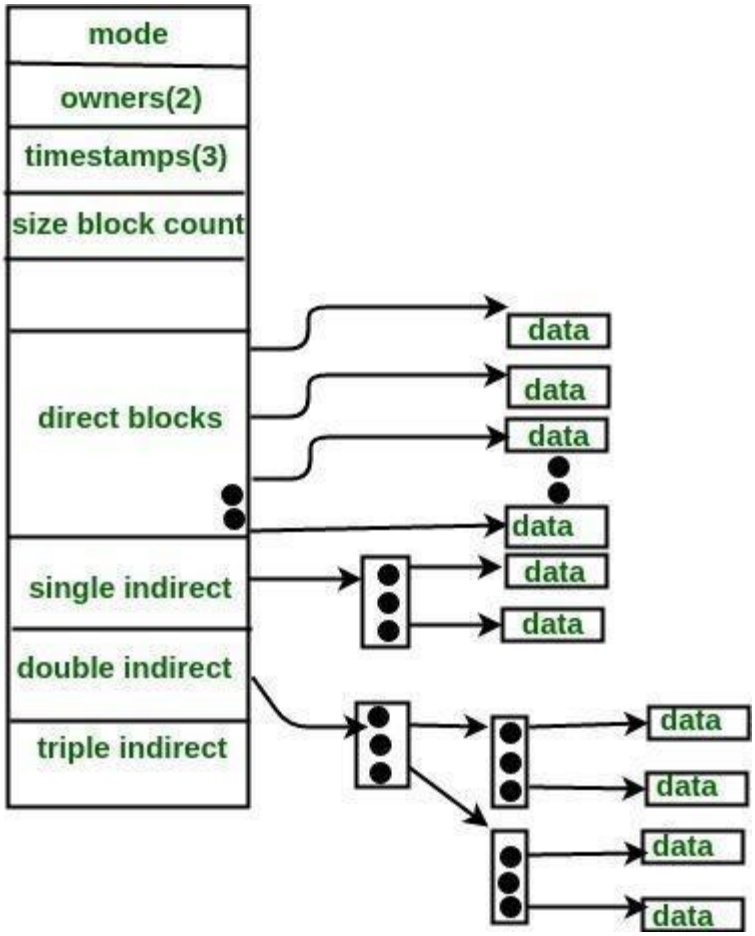
Advantages:

1. This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
2. It overcomes the problem of external fragmentation.

Disadvantages:

1. The pointer overhead for indexed allocation is greater than linked allocation.
2. For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.
3. For files that are very large, single index block may not be able to hold all the pointers. *Following mechanisms can be used to resolve this:*

1. **Linked scheme:** This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.
2. **Multilevel index:** In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.
3. **Combined Scheme:** In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file *as shown in the image below.* The first few of these pointers in Inode point to the **direct blocks** i.e the pointers contain the addresses of the disk blocks that contain data of the file. The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect. **Single Indirect block** is the disk block that does not contain the file data but the disk address of the blocks that contain the file data. Similarly, **double indirect blocks** do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.



❖ Disk Free space Management

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.

- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks.
- To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
- When a file is deleted, its disk space is added to the free-space list.

Free space is generally not managed by a simple free list. It can be implemented by various techniques Like,

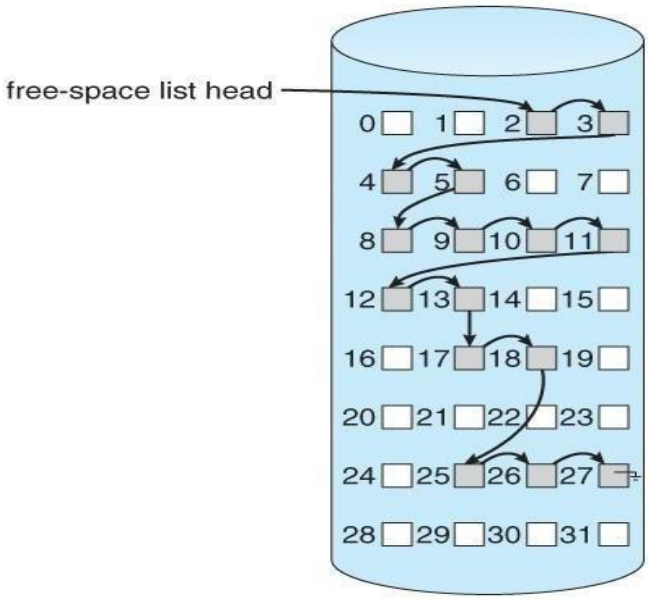
1. Bit Vector
2. Linked List
3. Grouping
4. Counting

1. Bit- Vector

- The Free space list is implemented as a bit map or bit vector
- Each block is represented by one bit (0 or 1)
- If the block is free the bit is 1. If the block is allocated bit is 0.
- For e.g. Consider a disk where blocks 2,3,5,8,9,12 are free and rest of the blocks are allocated. Then the free space bit map would be
- 001101001100100…
- The main advantages of this approach are that it is relatively simple and efficient to find first free block from 'n' consecutive free blocks on the disk.

2. Linked List

- Another approach is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and coaching it in memory.
- The first block contains a pointer to the next free disk block and so on.
- For e.g. We would keep a pointer to the block no.2 as the first free block, the block no.2 would contain the pointer to block no.3 which would point to block no.5 and so on.
- This scheme is not efficient to traverse a list we must read each block.



3. Grouping

- A modification of free list approach is to store the addresses of N free blocks in the first free block.
- The importance of this implementation is that the addresses of large number of free blocks can be found quickly.

4. Counting

- Another approach is to take the advantage of fact that, several contiguous block may be allocated or free simultaneously.
- In this approach keeping a list of n free disk addresses and the number n of free contiguous blocks that follow the first block.
- Each entry in the free space list consists of disk address and count.

❖ **Disk Scheduling Algorithms**

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

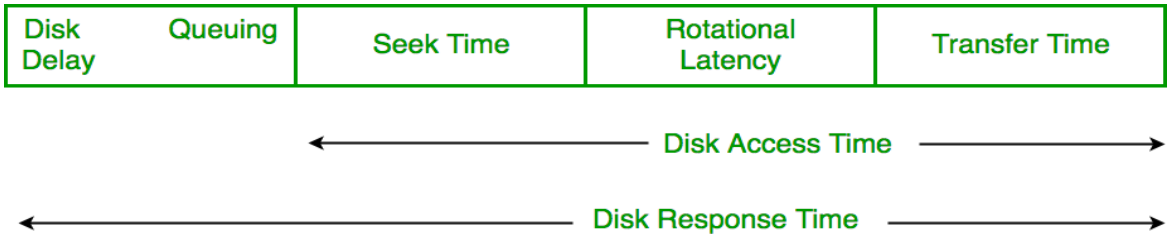
Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:

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



- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

❖ **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

❖ **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.

1. FCFS (First-Come First-Served)scheduling algorithm
2. SSTF (Shortest Seek Time First) algorithm
3. SCAN scheduling
4. C-SCAN scheduling
5. LOOK Scheduling
6. C-LOOK scheduling

1. FCFS: FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Advantages:

1. Every request gets a fair chance
2. No indefinite postponement

Disadvantages:

1. Does not try to optimize seek time
2. May not provide the best possible service

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

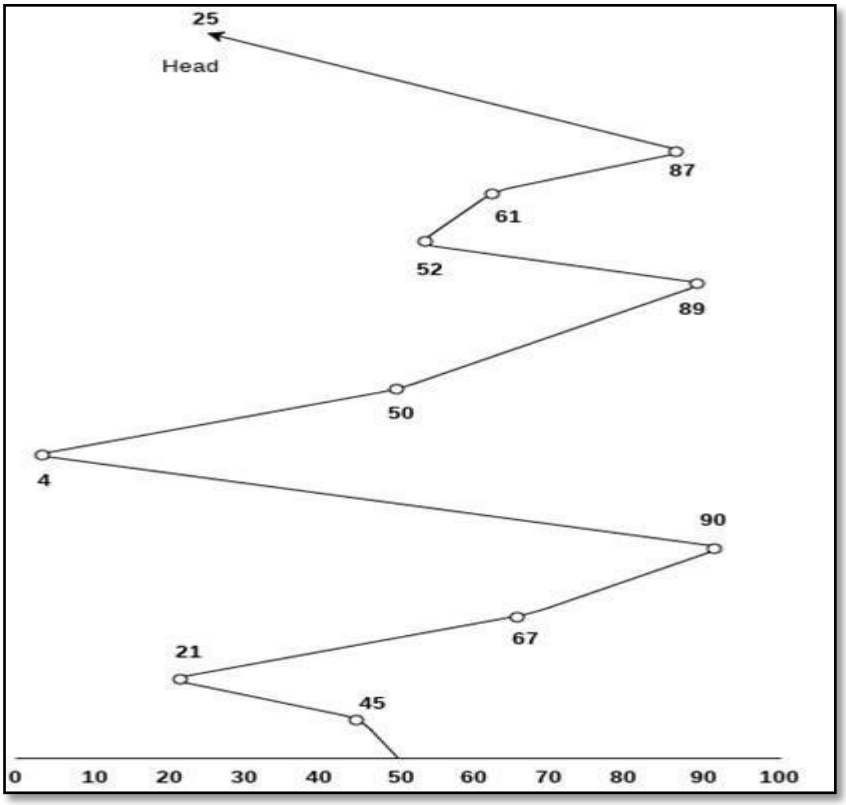


Fig: FCFS Scheduling

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

2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Advantages:

1. Average Response Time decreases
2. Throughput increases

Disadvantages:

1. Overhead to calculate seek time in advance
2. Can cause Starvation for a request if it has higher seek time as compared to incoming requests

3. High variance of response time as SSTF favours only some requests

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:

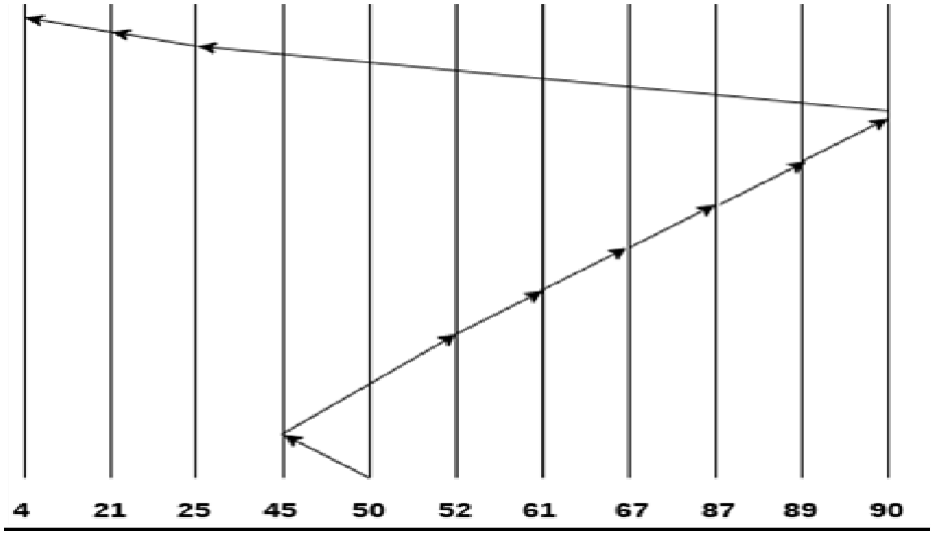


FIG: FCFS Scheduling

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

3. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Advantages:

1. High throughput
2. Low variance of response time
3. Average response time

Disadvantages:

1. Long waiting time for requests for locations just visited by disk arm

Example

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

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

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

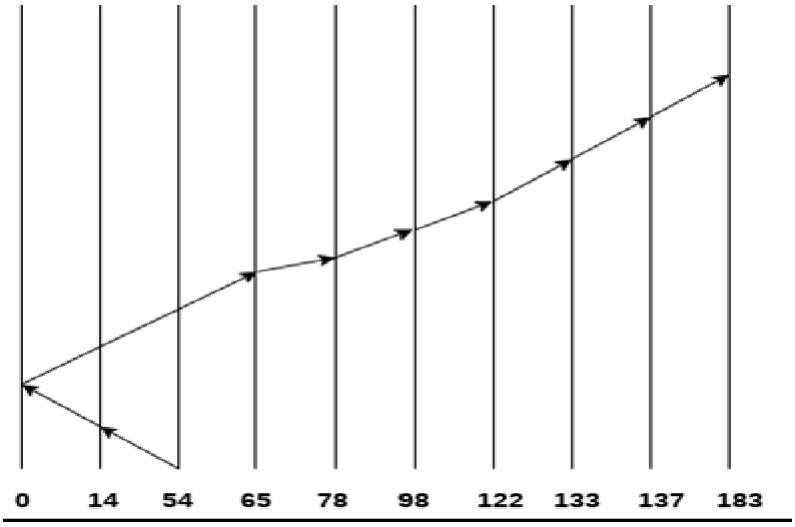


Fig: SCAN Scheduling

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

4. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area. These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

   Advantages:
   - Provides more uniform wait time compared to SCAN

Example

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

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

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

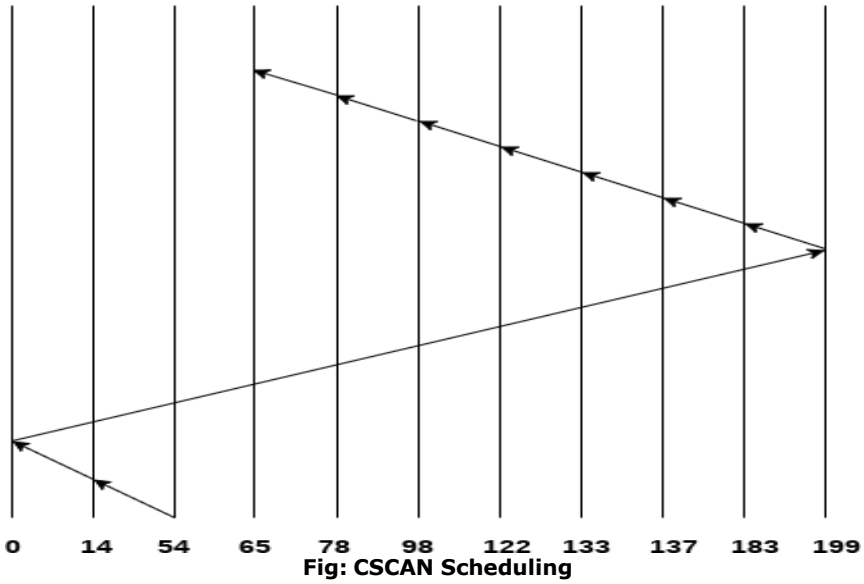


Fig: CSCAN Scheduling

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

5. **LOOK:** It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example

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

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

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

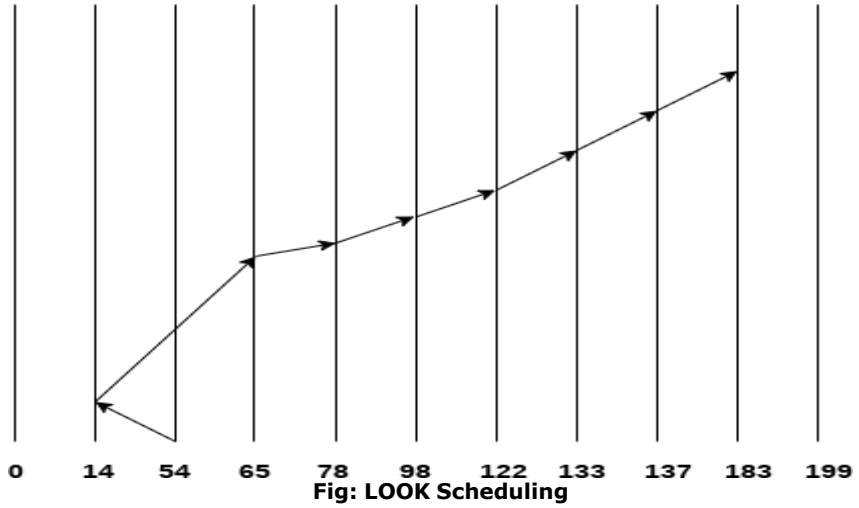


Fig: LOOK Scheduling

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

6. **CLOOK:** As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example

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

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

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

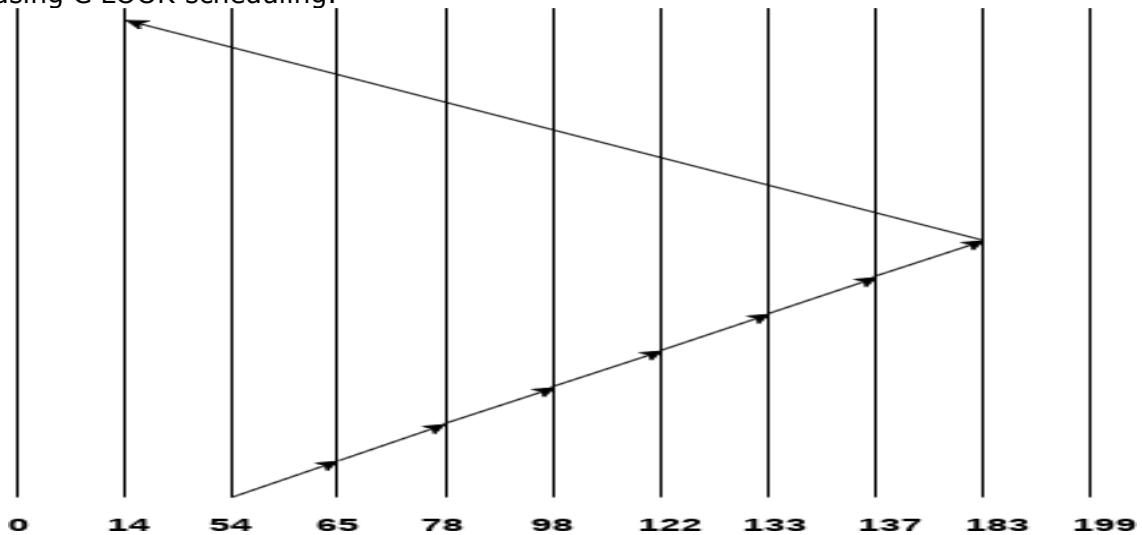


Fig: CLOOK Scheduling

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

❖ **Previous year Problems for Practice**

1. **March /April 2018**

suppose that a disk drive has 500 cylinders, numbered 0 to 499. The disk drive is currently serving at cylinder 346, and the previous request was at cylinder 262. The queue of pending requests, in FIFO order, is:

389,26,243,178,199,458,39,211,208,54

Staring from the current head position, what is the total distance that the disk arm moves to satisfy all the pending requests for each of the following diskscheduling algorithms?

(a) FCFS (b) SCAN (c) SSTF

2. **Oct/Nov 2018**

Suppose a disk drive has 200 cylinders number from 0 to 199. The disk drive is currently served the request at cylinder no 85. The queue for pending request in FIFO order is as follows :

12 36 129 98 88 175 142 140 68 40 37 6

Show the disk scheduling for the following algorithm.

(i) FCFS (ii) LOOK

(iii) SCAN (iv) SSTF

3. **March/April 2018**

(A) Suppose a disk drive has 150 cylinders number from 0 to 149. The disk drive is currently served the request at cylinder no 35. The queue for pending request in FIFO order is as follows :

11 46 120 90 88 128 142 145 66 40 30 9

Show the disk scheduling for the following algorithm.

(I) FCFS

(II) LOOK

(III) SCAN

(IV) SSTF

March/april-2017

(A) Suppose that a disk drive has 300 cylinders, numbered 0 to 299. The disk drive is currently serving at cylinder 146 and the previous request was at cylinder 122. The queue of pending request in FIFO order is.

89 ,126,43,178,99,259,39,112

Starting from current head position, what is the total distance that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms?

1)FCFS 2)SCAN 3)LOOK

October 2017

(A) Suppose that a disk drive has 300 cylinders, numbered 0 to 299. The disk drive is currently serving at cylinder 148, and the previous request was at cylinder 124. The queue of pending request in FIFO order is:

91, 128, 45, 180, 101, 261, 41, 114

Starting from current head position, what is the total distance that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms?

1)FCFS 2)SCAN 3)LOOK