DEVICE MANAGEMENT

Devices are all pieces of equipment for a computer that perform Input / Output (*short:* I/O) operations but that the computer does not necessarily need to work.

In some way all parts of a computer perform I/O operations, but without CPU or memory a computer would not work at all. Therefore CPU and memory are no devices whereas hard disk drives, mouse, keyboard, scanner, sound card and so on all are devices.

Devices are sometimes referred to as *peripheral devices* to emphasize the point that they are "additional" to the core hardware system. In this section the main effort is put on storage devices, that are devices that store data permanently.

Device Management is needed for offering a uniform and consistent approach to all I/O operations. There are considerable differences between all the system's devices; their speed, how data are transferred and represented, how to prevent and detect errors and how they are handled.

DEVICE MANAGEMENT FUNCTIONS

The basic functions of device management are as mentioned below:

1. Keeping track of the status of all devices, which requires special mechanism. One commonly used mechanism is to have a database such as Unit Control Block (UCB) associated with each device.
2. Deciding on policy to determine who gets a device, for how long and when. A wide range of techniques is available for implementing these policies. For example, high device utilization attempts to match the non-uniform requests by processes to the uniform speed of many I/O devices. There are 3 basic techniques for implementing the policies of device management:
   a. **Dedicated**: A technique whereby a device is assigned to a single process.
   b. **Shared**: A technique whereby a device is being shared by many processes.
   c. **Virtual**: A technique where one physical device is simulated on another physical device.
3. Allocation- physical assigning device to a process. Likewise the corresponding control units and channels must be assigned.
4. De-allocation policy and techniques. De-allocation may be done on either a process or a job level. On a job level, a device is assigned for as long as the job exists in the system. On process level, a device is assigned for as long as the process needs it.

DEVICE CHARACTERISTICS

Devices which are used to perform I/O can be grouped into three categories:

- **Human readable:** Suitable for communication with the computer user. Examples include printers and terminals, keyboard, and perhaps other devices such as a mouse.
- **Machine readable:** Suitable for communicating with electronic equipment. Examples are disk drives, USB keys, sensors, controllers and actuators.
- **Communication:** Suitable for communicating with remote devices. Examples are digital line drivers and modems.

Devices can be categorized using following criteria,

- **Data rate:** There may be differences of several orders of magnitude between the data transfer rates.
- **Application:** The user to which a device is put has an influence on the software and policies in the operating system and supporting utilities. For example a disk used for files requires the support of file management software. A disk used as s backing store for pages in a virtual memory scheme depends on the use of virtual memory hardware and software. These applications have an impact on disk scheduling algorithms. For example a terminal may be used by an ordinary user or a system administrator.
- **Complexity of control:** A printer requires a relatively simple control interface. A disk much more complex. The effect of these differences on the operating system is filtered to some extent by the complexity of the I/O module that controls the device.
- **Unit of transfer:** Data may be transferred as a stream of bytes or characters or in large blocks.
- **Data representation:** Different data encoding schemes are used by different devices, including differences in character code and parity conventions.
- **Error condition:** The nature of errors, the way in which they are reported, their consequences, and the available range of responses differ widely from one device to another.

DISK SPACE MANAGEMENT

- ### DISK FORMATTING

A new magnetic disk is a blank slate. It is just a platter of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting, or physical formatting. Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area, and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC). When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area. When the sector is read, the ECC-is recalculated and is compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates

that the data area of the sector has become corrupted and that the disk sector may be bad. The ECC is an error-correcting code because it contains enough information that, if only a few bits of data have been corrupted, the controller can identify which bits have changed and can calculate what their correct values should be.

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. The first step is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. After partitioning, the second step is logical formatting (creation of a file system). The OS stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory. To increase efficiency, most file systems group blocks together into larger chunks, frequently called clusters. Disk I/O is done via blocks, but file system I/O is done via clusters.

- **BOOT BLOCK:**

For a computer to start running-for instance, when it is powered up or rebooted-it must have an initial program or bootstrap program to run. It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system. The bootstrap program finds the operating system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating-system execution. The bootstrap is stored in ROM. ROM is read only so it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips. Most systems store a tiny bootstrap loader program in the boot ROM whose only job is to bring in a full bootstrap program from disk. The full bootstrap program can be changed easily: A new version is simply written onto the disk. The full bootstrap program is stored in the boot blocks at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk. The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code. The full bootstrap program is more sophisticated than the bootstrap loader in the boot ROM; it is able to load the entire OS from a non-fixed location on disk and to start the operating system running.

- **BAD BLOCKS:**

Because disks have moving parts and small tolerances, they are prone to failure. Sometimes the failure is complete; in this case, the disk needs to be replaced its contents restored from backup media to the new disk. More frequently, one or more sectors become defective. Most disks even come from the factory with bad blocks. Depending on the disk and controller in use, these blocks are handled in a variety of ways.

On simple disks, such as some disks with IDE controllers, bad blocks are handled manually. If format finds a bad block it writes as a special value in corresponding FAT entry to tell the allocation routine not to use that block. If blocks go bad during normal operation then a special program (chkdsk) must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

The controller maintains a list of bad blocks on the disk. The list is initialized during

the low-level formatting at the factory and is updated over the life of the disk. Low-level formatting also sets aside spare sectors not visible to the operating system. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding. As an alternative to sector sparing, some controllers can be instructed to replace a bad block by sector slipping.

DISK ALLOCATION

When file is created, storage space is allocated to it. Also, when new data is added in an existing file, file size grows and it needs extra storage space. In a similar way, storage space is released when a file is deleted or data from a file is deleted. An important function of the file system is to manage space on the secondary storage, which includes keeping track of both disk blocks allocated to files and the free blocks available for allocation.

There are two main goals, which should be fulfilled while allocating space on disk to files. These goals are as below:

1. Disk space should be utilized effectively.
2. Files should be accessed quickly.

At the time of space allocation, system must keep track of which disk blocks go with which files. Here the disk is considered as a collection of fixed size of blocks, where size varies from system to system. Most commonly it is of 512 bytes of 1kb in size. These blocks are being numbered starting from 0 or 1 to some maximum.

But, secondary storage introduces two additional problems:

1. slows disk access time and
2. larger number of blocks to deal with

In spite of that, many considerations are similar to both environments, particularly, contiguous and non contiguous allocation of files.

Whenever there is need to allocate storage space to a file one or more disk blocks are allocated to the file. in a similar way, whenever a file is being deleted, allocated blocks will become free for reallocation.

There are three main methods for disk allocation:

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

1. CONTIGUOUS ALLOCATION (NOT CONTINUOUS)

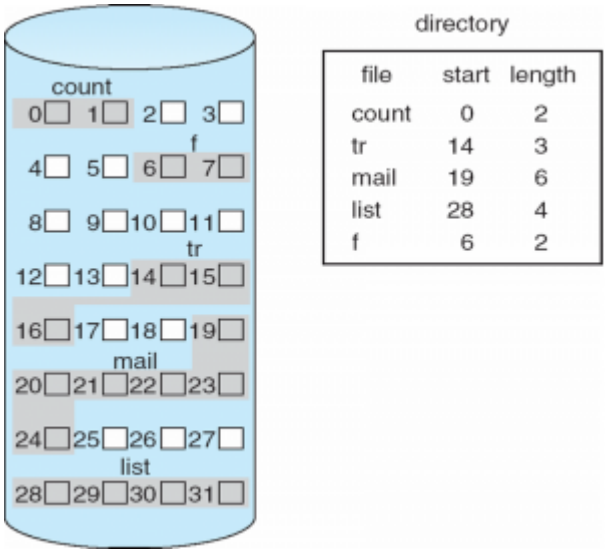
In contiguous allocation, files are assigned to contiguous areas of secondary storage. A user specifies in advance the size of the area needed to hold a file to be created. If the desired amount of contiguous space is not available, the file cannot be created.

Each file occupies asset of contiguous blocks on the disk. Thus, on a disk block of 1kb, a file of 50 kb would contain 50 consecutive disk blocks. Whereas, if block size is 2kb; it will occupy 25 consecutive blocks. When a file is created, a disk is searched to find out a chunk of free memory having enough size to store a file. If such chunk is found, required

memory is allocated. The directory entry contains file name, starting block number and length of a file.

File Name	Starting Block #	Length

This method is widely used on CD-ROMs. Here, all the files sizes are known in advance. Also, they will never change during subsequent uses of CD-ROM. Figure shown besides is depicting such allocation for 4 different file.



directory

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

Advantages:

1. Simple to implement. Information here required is only two things; one starting block #; and second, length of file as a total numbers of blocks.
2. All successive records of a file are normally physically adjacent to each other. This increases the accessing speed of records. It means that if records are scattered through the disk it's accessing will be slower. Accessing a file which has been contiguously allocated is fairly easy.

Disadvantages:

1. Finding free space for a new file is time consuming. This requires searching an entire disk until required free memory is found.
2. If size of an existing file increases, it may not be possible to accommodate such extension.
3. External fragmentation is possible. When file is deleted, its blocks are free leaving hole on the disk. With time, disk will consist of files and holes. Such hole may be too small to accommodate new files and will waste space on the disk. It is known as **external fragmentation**. Solution for this problem is defragmentation or compaction the file.

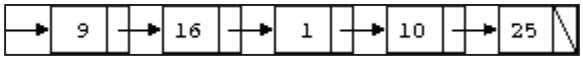
2. LINKED ALLOCATION

Each file is a linked list of disk blocks. Each linked block contains pointer to the next block in the list. These disk blocks may be scattered anywhere on the disk. Directory entry contains the start and last block numbers in a linked list. The directory entry structure is shown below:

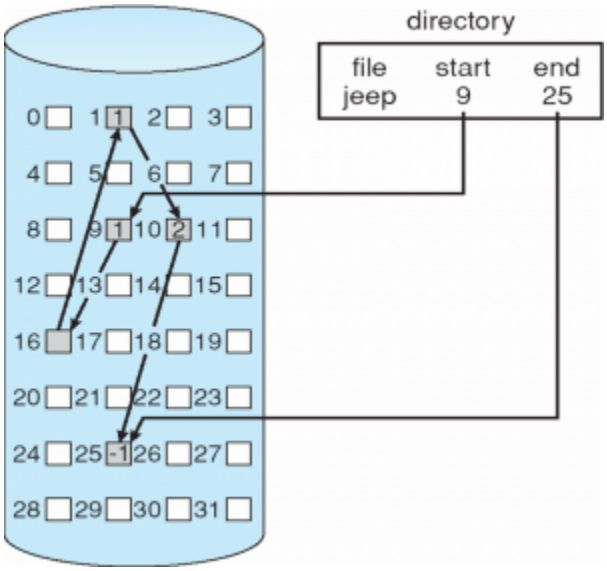
File Name	Starting Block#	Last Block#

When a new file is created; a new directory entry is created. Initially it contains 'null' as both the block number. A write to the file causes free data blocks to be added to the file; such blocks are added t the end of linked list. Directory entry is updated on each such occasion. To read a file, all blocks are read by following the pointers from block to block. Following figure is depicting the linked allocation.

In the figure beside; to reach block # 10, it is required to traverse through block # 9, 16 and 1.



An important variation on the linked allocation method, called File Allocation Table (FAT), is used by the MS-DOS and older versions of Windows Operating Systems.



Advantages:

1. It does not suffer from external fragmentation.
2. Any free disk block can be allocated to a file. Such block does not need to be a consecutive block as in previous method. So, disk space can be utilized effectively.

Disadvantages:

1. File access is time consuming. It is required to access all the data blocks in a linked list to reach some particular block.

2. Random access is not possible directly.
3. Extra space is required for pointers in each data block.

3. INDEX ALLOCATION

In linked allocation, pointers to various disk blocks are scattered on disk among various disk blocks. |Due to this reason, linked allocation cannot support efficient direct access. Indexed allocation solves this problem.

It brings all the pointers together into one location; the Index Block. Each file contains its own index block. An index block is an array of 'disk block addresses'. The 'ith' entry in the index block points to the 'ith' block of the file. Directory entry contains file name and the index block #. This looks as shown below:

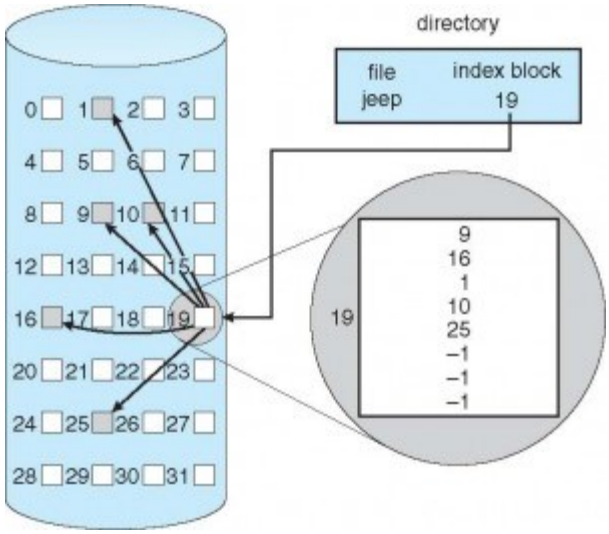
File Name	Index Block#

When new file is created, a new directory entry is created. Initially all pointers in index block are set null. When the 'ith' block is first written, a free disk block is allocated and its address is put in the 'ith' entry in the index block.

The following figure depicts the indexed allocation for the file 'Hw1'. The directory entry for the index block is done as block #3 for this file. This index block entry is composed of all other block # entries which are having the data contents of the file Hw1.

The index entry consists of all the block entries which consist of the data contents.

13	26	22



Advantages:

1. It does not suffer from external fragmentation.
2. Direct access is efficient.

Disadvantages:

1. It suffers from wasted space. Index block may be partially filled. This wastes remaining memory space of an index block. For example, if file contains two data blocks, then index block will have only two entries to point to theses two data blocks, remaining entire index block will be wasted,
2. Maximum allowable file size depends on the size of an index block.

Solutions:

In this allocation method the main problem is the size of index block. If it is too large, it may waste space. If it is too small it cannot accommodate enough pointers for a large file. Following are the solutions:

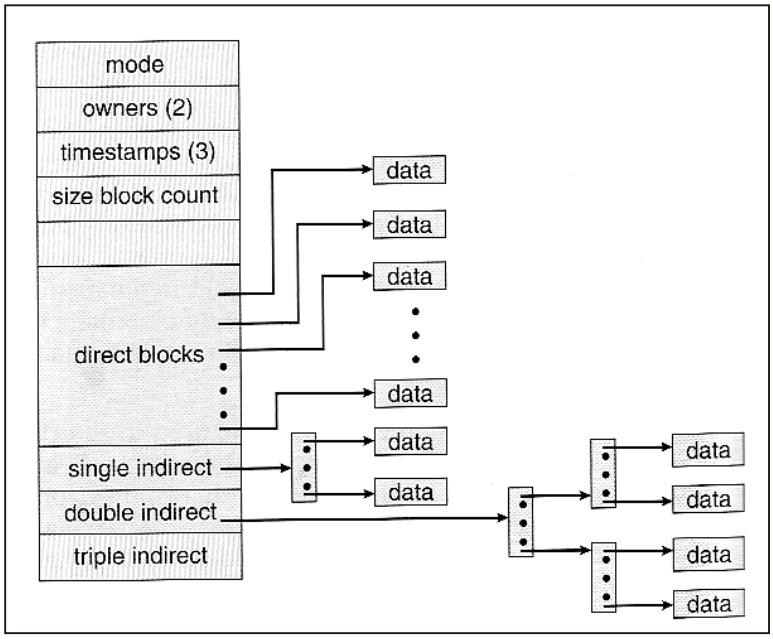
1. Linked Scheme

An index block is normally of one disk block size. For larger files more than one index block can be used by linking them together by a linked list.

2. Multilevel Index / Hierarchical Indexing

Two or more level of index blocks is used here. First level index blocks point to a set of second level index blocks. Second level index blocks point to disk blocks containing the data. For larger and larger file, levels can be increased.

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 SCHEDULING

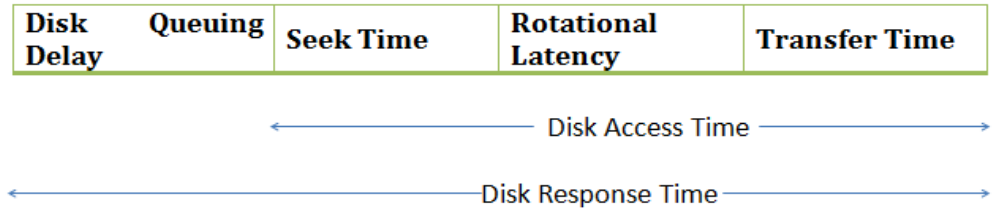
Disk scheduling is is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

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 disk controller. Thus other I/O requests need to wait in 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 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.

TYPES OF DISK SCHEDULING ALGORITHMS

Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms:

1. First Come-First Serve (FCFS) 2. Shortest Seek Time First (SSTF) 3. Elevator (SCAN) 4. Circular SCAN (C-SCAN) 5. LOOK 6. C-LOOK

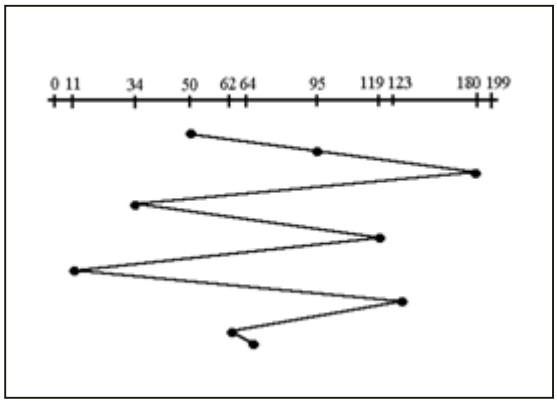
These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why C-LOOK is the best algorithm to use in trying to establish less seek time.

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. First Come First Serve (FCFS)

All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next.

For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.



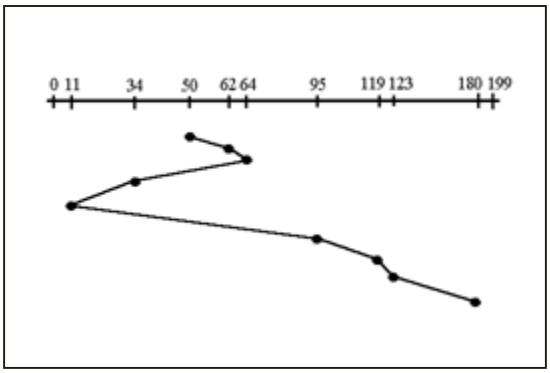
Advantage Simple and Fair to all requests

Disadvantage Not efficient, because the average seek time is very high. It suffers from zigzag effect.

2. Shortest Seek Time First (SSTF)

Selects the request with the minimum seek time from the current head position. Also called Shortest Seek Distance First (SSDF) – It's easier to compute distances. It's biased in favor of the middle cylinders requests. SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the processes are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.



Advantage More efficient than FCFS
Disadvantage Starvation is possible for requests involving longer seek time

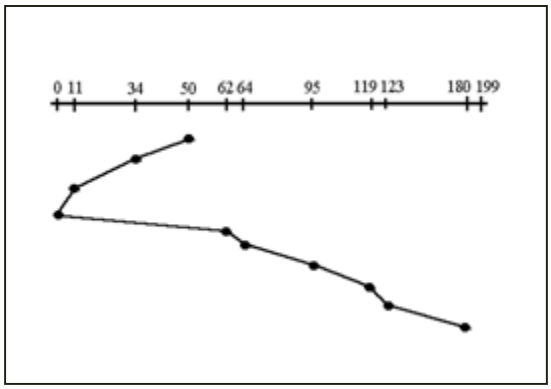
Elevator Algorithms

Theses algorithms are based on the common elevator principle. Four combinations of Elevator algorithms: **SCAN**, **LOOK**, **C-SCAN** and **C-LOOK**. They services in both directions or in only one direction. They operates until last cylinder or until last I/O request encountered.

3. Elevator (SCAN)

This approach works like an elevator does. The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. It moves in both directions until both ends. IT tends to stay more at the ends so more fair to the extreme cylinder requests. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it

has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.



Advantage

More efficient than FCFS

Also there is no starvation for any requests

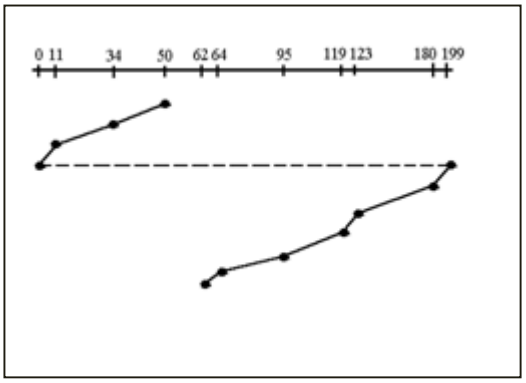
Disadvantage

Require extra head movement between two extreme points. For example, after servicing 5th cylinder there is no need to visit the 0th cylinder. Though, this algorithm visits the end points.

Not so fair; cylinders which are just behind head will wait longer

4. Circular Scan (C-SCAN)

Circular scanning works just like the elevator to some extent. The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. It treats the cylinders as a circular list that wraps around from the last cylinder to the first one. It provides a more uniform wait time than SCAN; it treats all cylinders in the same manner. It begins its scan toward the nearest end and works its way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 tracks, but still this isn't the more sufficient.



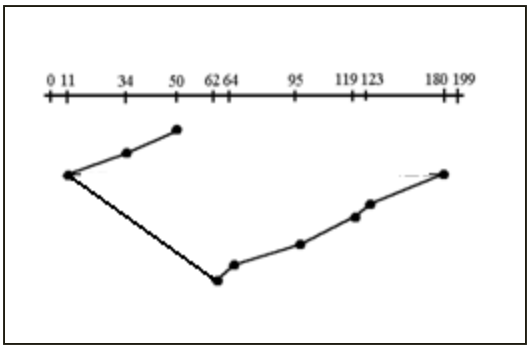
Advantages:

- Provides more uniform wait time compared to SCAN

5. LOOK

The disk arm starts at the first I/O request on the disk, and moves toward the last I/O request on the other end, servicing requests until it gets to the other extreme I/O request on the disk, where the head movement is reversed and servicing continues. It moves in both directions until both last I/O requests; more inclined to serve the middle cylinder requests.

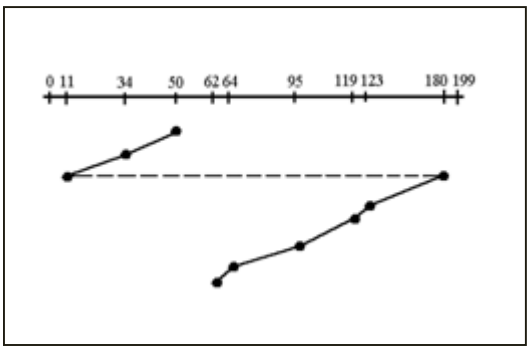
In this example head will move from 11 to 62 rather going to 0 from 11.



6. Circular LOOK(C-Look)

It is an enhanced Look version of C-Scan. Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk. Scan versions have a larger total seek time than the corresponding Look versions.

This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.



From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

Selecting Disk Scheduling Algorithms

Here some factors are given which effect the performance of a disk scheduling algorithm as shown below:

1. **Number and types of Requests:**
   - Performance of a disk scheduling algorithm heavily depends upon the number and type of request
   - For example, if the device queue contains only one choice is available, all algorithms will behave like FCFS.
2. **File Allocation Method:**
   - File allocation method also has good contribution in deciding performance.
   - For example, when a contiguously allocated file is accessed, the disk requests are normally close to each other. Whereas for a linked or indexed file, disk blocks are scattered over disk, and disk head requires great movements.
3. **Location of directories and index blocks:**
   - Every file must be opened before use. For this, there is a need to search directory structure to determine location of files. So, directories are accessed frequently.
   - If the directory entry is on the first cylinder, and file's data are on the final cylinder, then disk head required to move the entire width of the disk.

These are the main three factors that must be observed at the time of selecting the algorithm. An average seek time with FCFS is very high. SCAN required unnecessary head movements between two end points. So, in default case, either SSTF or LOOK is reasonable choice as a disk scheduling algorithm.

FREE SPACE MANAGEMENT

There is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files.
To keep track of free disk space, the system maintains a free-space list. The free-space list records all disk blocks that are free
The process of looking after and managing free blocks of disk is called **Free Space Management**
Free Space List records all free disk blocks that are not allocated to any file or directory. Whenever, a new file is created, the free space list is searched for the required amount of space. Then the space is allocated to that newly created file. Its entry is removed from the Free Space List.
There are some methods or techniques to implement free space list. These are as follows:

- Bitmap
- Linked Lists
- Grouping
- Counting

Bitmap or bit vector:

When free space is list is implemented as bit map, each block of disk is represented by a bit.
If the block is free, its bit is set to 1. If the block is allocated, the bit is 0.
For Example: Apple Macintosh operating system uses bit map method to allocate the disk space.
Assume the following are free. Rests are allocated:

2,3,4,5,9,10,13

Blocks →	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Bits →	0	0	1	1	1	1	0	0	0	1	1	0	0	1

Advantages:

- It is relatively simple.
- Efficient to find the free space on the disk.
- Fast random access allocation check

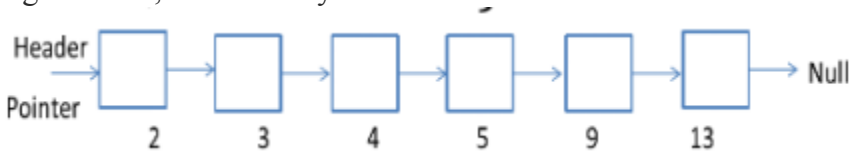
Disadvantages:

- This may require special hardware support to find the first 1 in a word if it is not 0.
- Wasteful on larger disks
- Poor scalability

Linked List:

In this method, a linked list of all the free disk blocks is maintained.
The first free block in the list can be pointed out by a head pointer, which is kept in a special location on the disk.

Using this disk, It is not easy to search the free list.



Advantages:

- Whenever a file is to be allocated a free block, the operating system can simply allocate the first block in free space list and move the head pointer to the next free block in the list.

Disadvantages:

- Searching the free space list will be very time consuming; each block will have to be read from the disk, which is read very slowly as compared to the main memory.
- Not Efficient for faster access.

Grouping:

A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
The first n-1 of these are actually free.
The last one is the disk address of another block containing addresses of other n free blocks.
The importance of this implementation is that addresses of a large number of free blocks can be found quickly.

Counting:

This method is based on the fact that we can allocate o free several contiguous blocks at the same time.
This method keeps the address of the first free block and the number n of the free contiguous blocks that follows the first block.
Each entry in the free-space list then consists of a disk address and a count.
Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.