Unit-5

SYLLABUS:

Storage Management:
File-System Interface: File concept, Access Methods, Directory & Disk Structure.
File-System Implementation: File-System Structure, Allocation Methods, Free Space Management.
Mass-Storage Structure: Overview of Mass-Storage Structure, Disk Scheduling. FCFS Scheduling, SSTF Scheduling, SCAN Scheduling, C-SCAN Scheduling, LOOK Scheduling, Selection of a Disk-Scheduling Algorithm.

Storage Management:

File System Interface:

File Concept:

- A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities.

File Attributes:

- A file's attributes vary from one operating system to another but typically consist of
    1) Name
    2) Identifier
    3) Type
    4) Location
    5) Size
    6) Protection
    7) Time, date, and user identification

1) **Name:** The symbolic file name is the only information kept in human readable form.
2) **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
3) **Type:** This information is needed for systems that support different types of files.
4) **Location:** This information is a pointer to a device and to the location of the file on that device.
5) **Size:** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
6) **Protection:** Access-control information 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:

- A file is an abstract data type.
- To define a file properly, we need to consider the operations that can be performed on files.
- The operating system can provide system calls to **create, write, read, reposition, delete, and truncate files.**
- The operating system must do to perform each of these six basic file operations.

Creating a file

- Two steps are necessary to create 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.

Writing a file

- To write a file, we make a system call specifying both the name of the file and the information to be written to the file.
- Given the name of the file, the system searches the directory to find the file's location.
- The system must keep a write pointer to the location in the file where the next write is to take place.
- The write pointer must be updated whenever a write occurs.

Reading a file

- To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.
- Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place.
- Once the read has taken place, the read pointer is updated.
- Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process.
- Both the read and write operations use this same pointer, saving space and reducing system complexity.

Repositioning within a file

- The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.
- Repositioning within a file need not involve any actual I/0. This file operation is also known as files seek.

Deleting a file

- To delete a file, we search the directory for the named file.
- Having found the associated directory entry, we release all file space, so
- That it can be reused by other files, and erase the directory entry.

Truncating a file

- The user may want to erase the contents of a file but keep its attributes.
- Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged.

File Types:

- If an operating system recognizes the type of a file, it can then operate on the file in reasonable ways.
- A common technique for implementing file types is to include the type as part of the file name.
- The name is split into two parts-a name and an extension, usually separated by a period character.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File structure

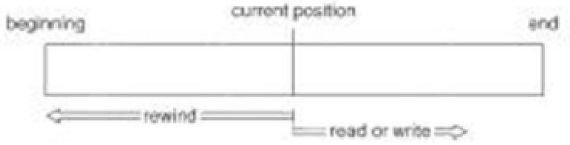
- A File Structure needs to be predefined format in such a way that an operating system understands
- It has an exclusively defined structure, which is based on its type.
- Three types of files structure in OS:
- **A text file:** It is a series of characters that is organized in lines.
- **An object file:** It is a series of bytes that is organized into blocks.
- **A source file:** It is a series of functions and processes.

File Access Methods:

- Files store information.
- When it is used, this information must be accessed and read into computer memory.
- The information in the file can be accessed in several ways.
    1) Sequential access
    2) Direct-access
    3) Other Access(index) Methods

1) Sequential access:

- The phenomena in which, information in the file is processed in order, one record after the other is called as sequential access.
- Read operation-**read *next*** *-reads* the next portion of the file and automatically advances a file pointer, which tracks the I/O location.
- Write *operation*-**write *next***-appends to the end of the file and advances to the end of the newly written material (the new end of file).
- **A file can be reset to the beginning** and on some systems, a program may be able to skip forward or backward *n* records for some integer n-perhaps only for *n* = 1.



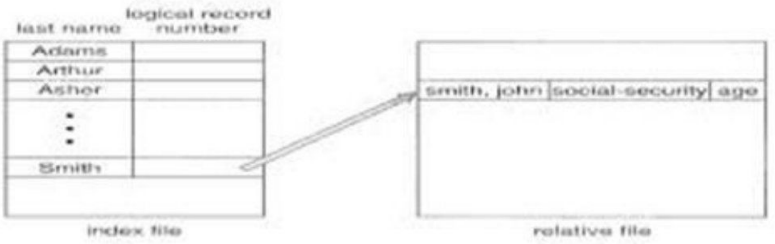
2) Direct-access:

- The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records.
- Direct-access files are of great use for immediate access to large amounts of information.
- For the direct-access method, the file operations must be modified to include the block number as a parameter.
- Thus, we have *read n* rather than *read next,* and *write n,* rather than *write next, ,* where *n* is the block number.
- We can implement sequential access with random access

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp;* *cp* = *cp* + 1;
write next	*write cp;* *cp* = *cp* + 1;

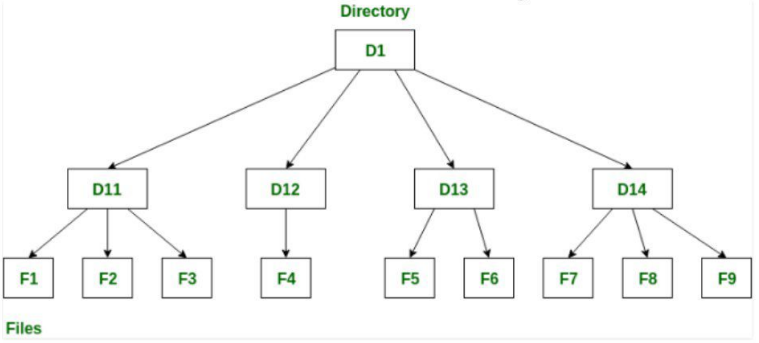
3) Other Access(index access) Methods:

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.
- To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.



Directory Structure:

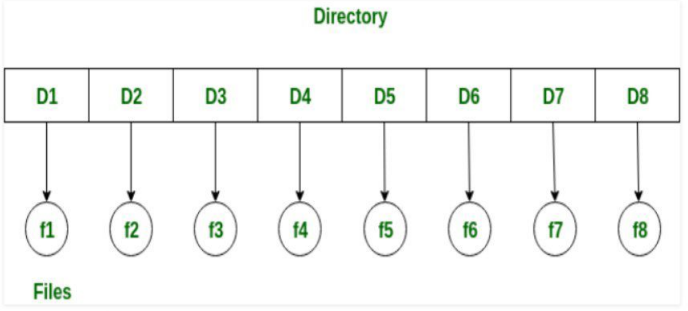
- A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner.



- The most common schemes for defining the logical structure of a directory are Single-level Directory
    1) Single Level Directory
    2) Two-Level Directory
    3) Tree-Structured Directories
    4) Acyclic-Graph Directories
    5) General Graph Directory

1) Single Level Directory

- The simplest directory structure is the single-level directory.
- All files are contained in the same directory, which is easy to support and understand.
- Since all files are in the same directory, they must have unique names.
- Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.



➢ **Advantages of Single Level Directory:**

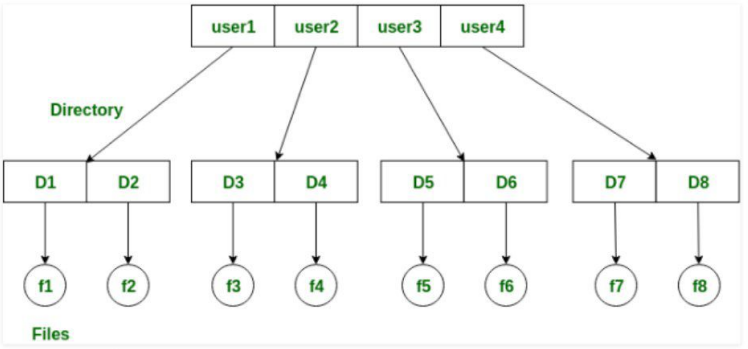
- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

➢ **Disadvantages of Single Level Directory:**

- There may chance of name collision because two files can not have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

2) Two-Level Directory

- In the two-level directory structure, each user has own user file directory (UFD).
- The UFDs have similar structures, but each lists only the files of a single user.
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched.
- The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- To name a particular file uniquely in a two-level directory, we must give both the user name and the file name
- A two-level directory can be thought of as a tree, or an inverted tree, of height 2.
- The root of the tree is the MFD.
- Its direct descendants are the UFDs.
- The descendants of the UFDs are the files themselves.
- The files are the leaves of the tree.
- Thus, a user name and a file name define a path name. Every file in the system has a path name.
- To name a file uniquely, a user must know the path name of the file desired.



➢ **Advantages of Two-Level Directory:**

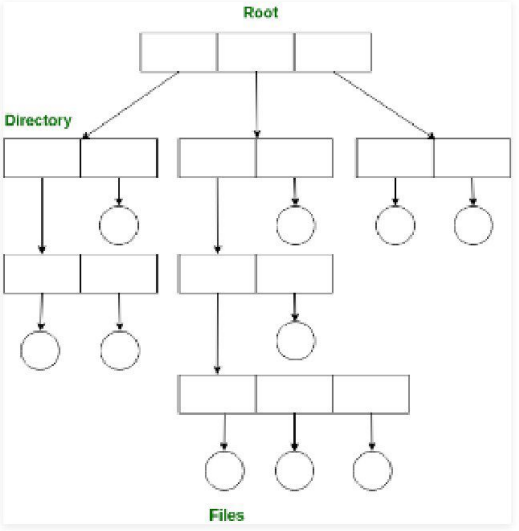
- We can give full path like /User-name/directory-name/.
- Different users can have the same directory as well as the file name.
- Searching of files becomes easier due to pathname and user-grouping.

➢ **Disadvantages of Two-Level Directory:**

- A user is not allowed to share files with other users.
- Still, it not very scalable, two files of the same type cannot be grouped together in the same user.

3) Tree-Structured Directory:

- The directory structure as a tree of arbitrary height allows users to create their own subdirectories and to organize their files accordingly.
- The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
- A directory is simply another file, but it is treated in a special way.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.
- Each process has a current directory.
- It contains most of the files that are of current interest to the process.
- Path names can be of two types: absolute and relative.
- An absolute Path name begins at the root and follows a down to the specified file, giving the directory names on the path.
- A relative Path name defines a path from the current directory.



➢ **Advantages of Tree-Structured Directory:**

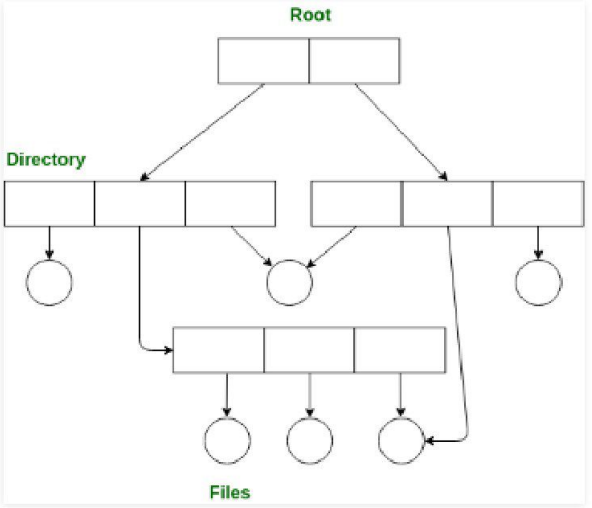
- Very general, since full pathname can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute paths as well as relative.

➢ **Disadvantages of Tree-Structured Directory:**

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.

4) Acyclic-Graph Directories

- A tree structure prohibits the sharing of files or directories.
- Acyclic-Graph-a graph with no cycles- allows the sharing of files or directories.
- A shared directory or file will exist in the file system in two (or more) places at once.
- The same file or subdirectory may be in two different directories.



➢ **Advantages of Acyclic Graph Directories:**

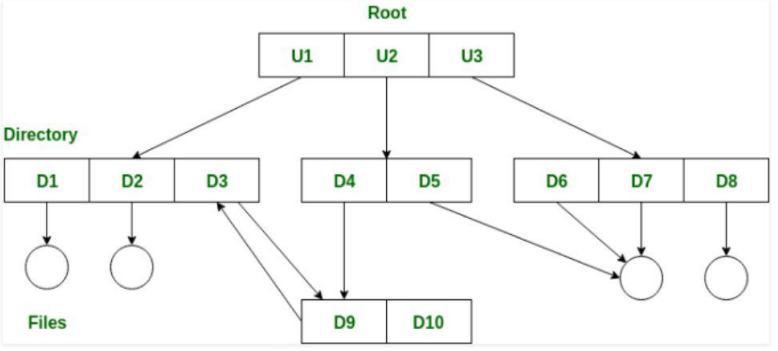
- We can share files.
- Searching is easy due to different-different paths.

➢ **Disadvantages of Acyclic Graph Directories:**

- We share the files via linking, in case deleting it may create the problem,
- If the link is a soft link then after deleting the file we left with a dangling pointer.
- In the case of a hard link, to delete a file we have to delete all the references associated with it.

5) General Graph Directories

- A serious problem with using an acyclic-graph structure is ensuring that there are no cycles.
- We want to avoid traversing shared sections of an acyclic graph twice, mainly for performance reasons.
- If cycles are allowed to exist in the directory, we likewise want to avoid searching any component twice, for reasons of correctness as well as performance.



➢ **Advantages of General Graph Directories:**

- It allows cycles.
- It is more flexible than other directories structure.

➢ **Disadvantages of General Graph Directories:**

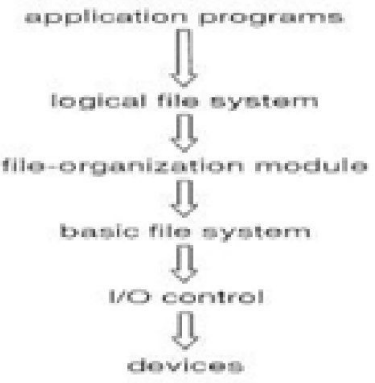
- It is more costly than others.
- It needs garbage collection.

File-System Implementation:

File-System Structure:

- (File systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.
- A file system poses two quite different design problems.
- The first problem is defining how the file system should look to the user. This task involves defining a file and its attributes, the operations allowed on a file and the directory structure for organizing files.
- The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.
- The file system itself is generally composed of many different levels.
- The File system structure is a layered design.
- Each level in the design uses the features of lower levels to create new features for use by higher levels.)

➢ **Layered file system structure**



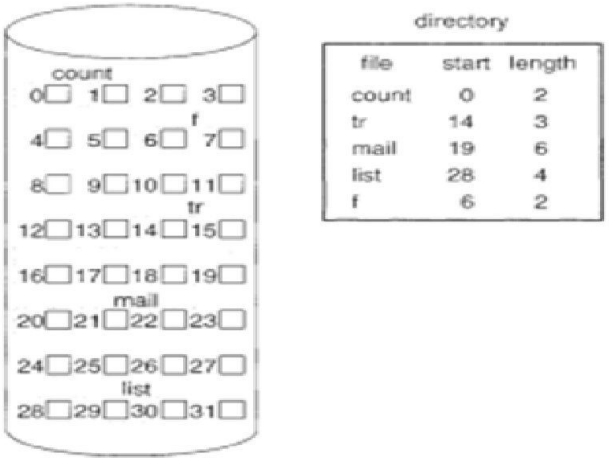
- The lowest level **I/O control** consists of device drivers and interrupts handlers to transfer information between the main memory and the disk system.
- The **basic file system** needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk.
- The **file organization module** knows about files and their logical blocks, as well as physical blocks.
- The **logical file system** manages metadata information. Metadata includes all of the file-system structure except the actual *data* (or contents of the files).

Allocation Methods:

- The direct-access nature of disks allows us flexibility in the implementation of files.
- The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed.

Contiguous Allocation:

- Contiguous allocation requires that each file occupy a set of contiguous blocks on disk.
- Disk addresses define a linear ordering on the disk.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.
- If the file is n blocks long and starts at location b, then it occupies blocks b, b + 1, b + 2,... , b + n - 1.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- Both sequential and direct access can be supported by contiguous allocation.

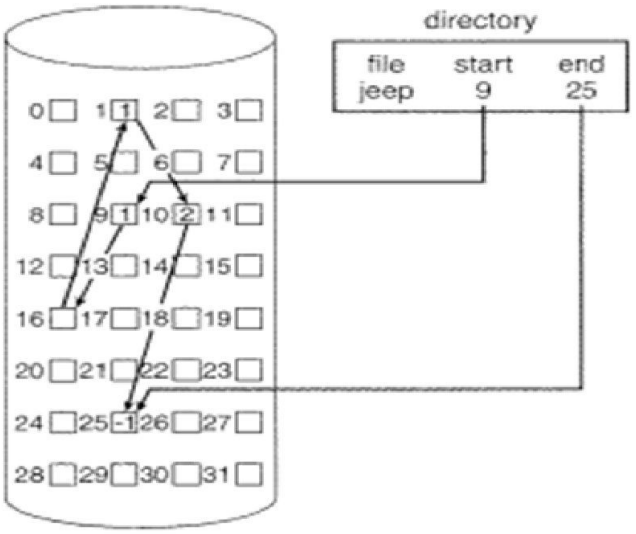


Contiguous allocation

- Contiguous allocation has some problems
  - Finding space for a new file
  - External fragmentation occurs

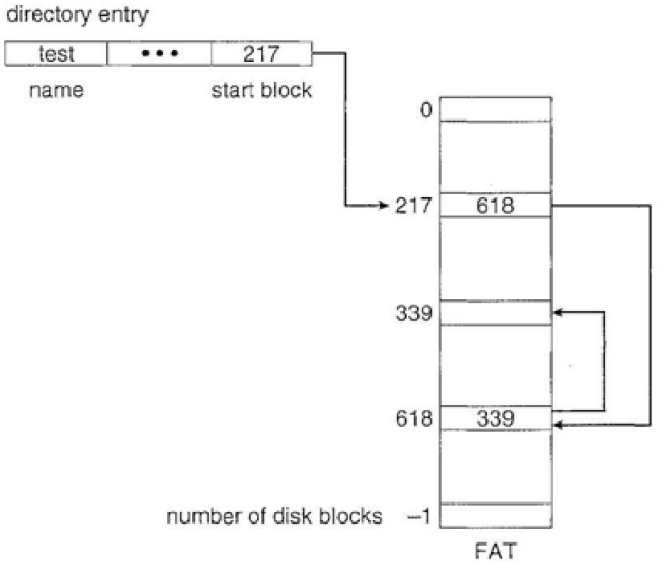
Linked Allocation:

- Linked Allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file.
- Each block contains a pointer to the next block and these are not available to the user.
- Thus, if each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508bytes.



Linked Allocation

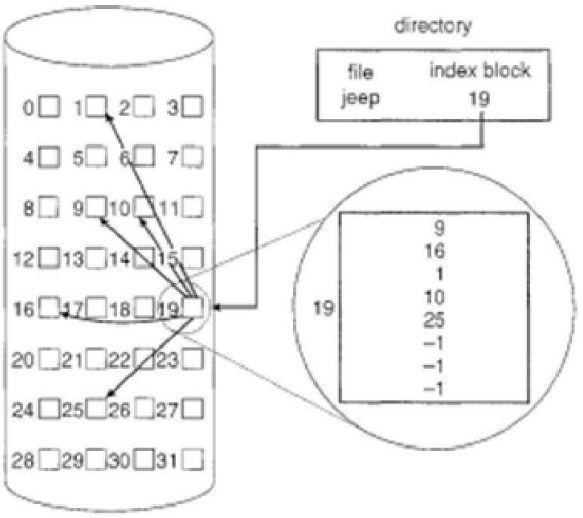
- For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25.
- To create a new file, we simply create a new entry in the directory.
- With linked allocation, each directory entry has a pointer to the first disk block of the file.
- This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. The size field is also set to 0.
- There is no external fragmentation
- Linked allocation does have disadvantages
- An important variation on linked allocation is the use of a File Allocation Table (FAT).
- The directory entry contains the block number of the first block of the file.
- The File Allocation table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until it reaches the last block, which has a special end-of-file value as the table entry.
- (An unused block is indicated by a table value of 0.
- Allocating a new block to a file is a simple matter of finding the first 0-valued table entry and replacing the previous end-of-file value with the address of the new block.
- The 0 is then replaced with the end-of-file value.)



File Allocation Table

Indexed Allocation:

- Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.
- Indexed Allocation solves this problem by bringing all the 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 entry contains the address of the index block.
- To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.
- Indexed allocation supports direct access
- No External fragmentation.
- Indexed allocation does suffer from
  - Waste space.
  - Pointer overhead of the index block
- Pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.



Indexed Allocation

- If the file is small, then more space in index block is wasted, because of fewer indexes needed.
- We can overcome the problem of how large the index block should be with linked scheme, multilevel index, and combined scheme

Mass-Storage Structure:

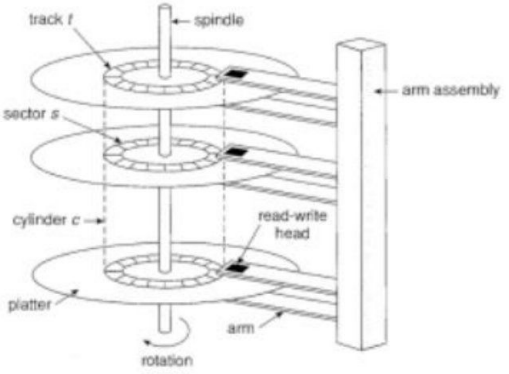
Overview of Mass-Storage Structure:

Q) Describe Mass Storage structure?

Ans:

- Secondary and tertiary storage devices are
  1) Magnetic Disks
  2) Magnetic Tapes

1) Magnetic Disks:



Moving-head disk mechanism

- **Magnetic Disks** provide the bulk of secondary storage for modern computer systems.
- Each disk **platter** has a flat circular shape, like a CD.
- The two surfaces of a platter are covered with a magnetic material.
- We store information by recording it magnetically on the platters.
- A read -write head "flies" just above 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 subdivided into **sectors.**
- The set of tracks that are at one arm position makes up a **cylinder.**
- Disk speed has two parts, transfer rate and positioning time.
- The **transfer rate** is the rate at which data flow between the drive and the computer.
- The **positioning time** sometimes called the **random access time** consists of the seek time and rotational latency.
- The time necessary to move the disk arm to the desired cylinder, called the **seek time**.
- The time necessary for the desired sector to rotate to the disk head, called the **rotational latency.**
- Although the disk platters are coated with a thin protective layer, the head will sometimes damage the magnetic surface. This accident is called a **head crash.**

2) Magnetic Tapes:

- Magnetic Tapes was used as an early secondary-storage medium
- Although it is relatively permanent and can hold large quantities of data, its access time is slow compared with that of main memory and magnetic disk.
- In addition, random access to magnetic tape is about a thousand times slower than random access to magnetic disk.
- So tapes are not very useful for secondary storage.
- Tapes are used mainly for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another.
- A tape is kept in a spool and is wound or rewound past a read-write head.
- Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
  - ❖ Tape capacities vary greatly, depending on the particular kind of tape drive. Typically, they store from 20GB to 200GB.

Disk Structure:

- Modern magnetic disk drives are addressed as large one-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer.
- The size of a logical block is usually 512 bytes, although some disks can be **low-level formatted** to have a different logical block size, such as 1,024 bytes.
- The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- *Sector 0 is the first sector of the first track on the outermost cylinder.*
- *The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.*
- By using this mapping, we can convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder, and a sector number within that track.
- In practice, *it is difficult to perform this translation, for two reasons*.
- **First**, **most disks have some defective sectors**, but the mapping hides this by substituting spare sectors from elsewhere on the disk.
- **Second**, **the number of sectors per track is not a constant on some drives**.
- Let's look more closely at the second reason.
- **Constant linear velocity (CLV)**, *the density of bits per track is uniform.*
- *The farther a track is from the center of the disk, the greater its length, so the more sectors it can hold*.
- *As we move from outer zones to inner zones, the number of sectors per track decreases.*
- *Tracks in the outermost zone typically hold 40 percent more sectors than do tracks in the innermost zone.*
- *The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head.*
- *This method is used in CD-ROM and DVD-ROM drives.*
- **Constant angular velocity (CAV)**, the disk rotation speed can stay constant.
- In this case, *the density of bits decreases from inner tracks to outer tracks to keep the data rate constant*.
- This method is used in hard disks.
- *The number of sectors per track has been increasing as disk technology improves*, and the *outer zone of a disk usually has several hundred sectors per track*.
- Similarly, *the number of cylinders per disk has been increasing; large disks have tens of thousands of cylinders.*

Disk Attachment:

Computers access disk storage in two ways.

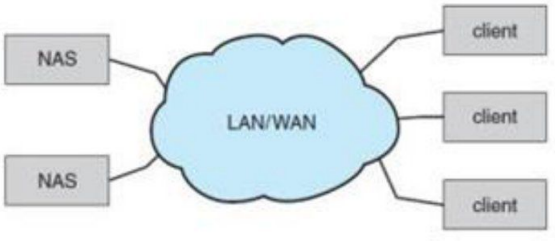
1. One way is via **I/O ports** (or **host-attached storage**); this is common on small systems.
2. The other way is via a **remote host in a distributed file system**; this is referred to as **network-attached storage**.

1) Host Attached Storage:

- Host-attached storage is storage accessed through local I/O ports.
- These ports use several technologies.
- The typical desktop PC uses an I/O bus architecture called **IDE or ATA**.
- This architecture supports a maximum of two drives per I/O bus.
- A newer, similar protocol that has simplified cabling is **SATA**.
- **High-end workstations and servers generally use** more sophisticated I/O architectures such as **fibre channel (FC)**, a high-speed serial architecture that can operate over optical fiber or over a four-conductor copper cable.
- It has two variants.
- One is a large switched fabric having a 24-bit address space. This variant is expected to dominate in the future and is the basis of **storage-area networks (SANs)**.
- Because of the large address space and the switched nature of the communication, multiple hosts and storage devices can attach to the fabric, allowing great flexibility in I/O communication.
- The other FC variant is an **arbitrated loop (FC-AL)** that can address 126 devices (drives and controllers).
- A wide **variety of storage devices are suitable for use as host-attached storage**.
- Among these are **hard disk drives, RAID arrays, and CD, DVD, and tape drives**.
- **The I/O commands that initiate data transfers to a host-attached storage device are reads and writes of logical data blocks directed to specifically identified storage units** (such as bus ID or target logical unit).

2) Network Attached Storage:

- A **network-attached storage (NAS)** device is a special-purpose storage system that is **accessed remotely over a data network.**
- **Clients access network-attached storage via a remote-procedure-call interface** such as **NFS for UNIX** systems or CIFS for Windows machines.
- The **remote procedure calls (RPCs) are carried via TCP or UDP over an IP network** usually the same local area network (LAN) that carries all data traffic to the clients.
- Thus, it may be easiest to think of NAS as simply another storage-access protocol.
- The **network attached storage unit is usually implemented as a RAID array** with software that implements the RPC interface.



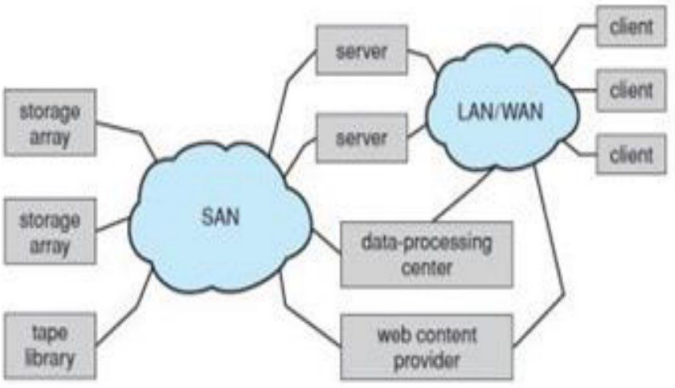
Network Attached Storage

- Network-attached storage provides a convenient way for all the computers on a **LAN to share a pool of storage with the same ease of naming and access enjoyed with local host-attached storage.**

- However, it tends to be **less efficient and have lower performance than some direct-attached storage options.**
- **ISCSI** is the *latest* **network-attached storage protocol**. In essence, it uses the IP network protocol to carry the SCSI protocol.
- Thus, networks rather than SCSI cables can be used as the interconnects between hosts and their storage.
- As a result, hosts can treat their storage as if it were directly attached, even if the storage is distant from the host.

3) Storage Area Network:

- **One drawback of network-attached storage** systems is that the storage I/O operations consume **bandwidth on the data network, thereby increasing the latency of network communication.**
- This problem can be particularly acute in large client–server installations—the communication between servers and clients competes for bandwidth with the communication among servers and storage devices.
- A **storage-area network (SAN) is a private network** (using storage protocols rather than networking protocols) **connecting servers and storage units.**



Storage Area Network

Disk Scheduling:

- **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.

Important terms of Disk Scheduling Algorithms are:

- **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.

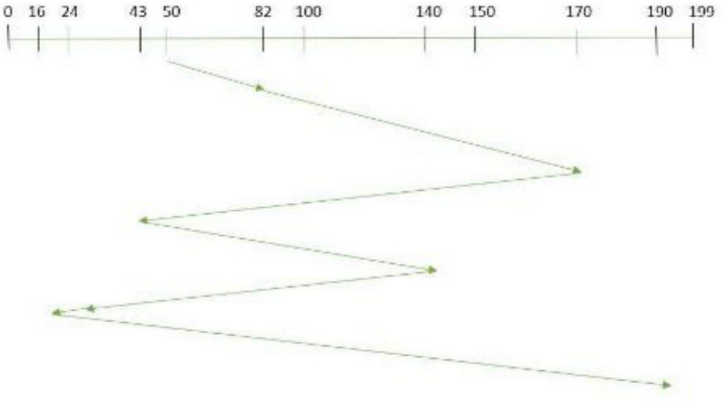
FCFS Disk Scheduling:

(First-Come First-Served Disk Scheduling Algorithm)

- 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.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50



FCFS Disk Scheduling Algorithm

So**, total seek time**: =(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) = 642

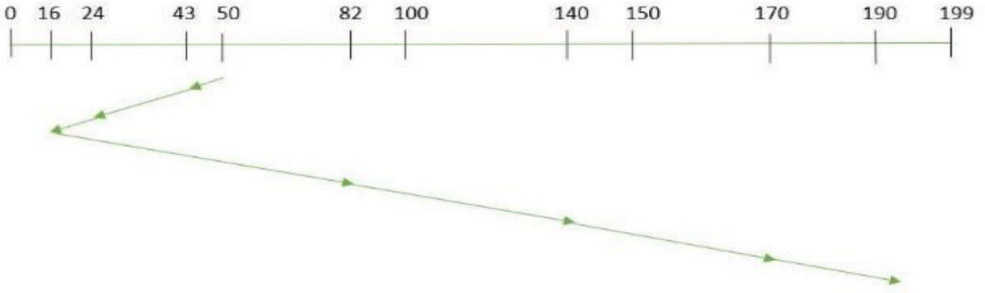
SSTF Disk Scheduling:

(Shortest-Seek-Time-First Disk Scheduling Algorithm)

- 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.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50



SSTF Disk Scheduling

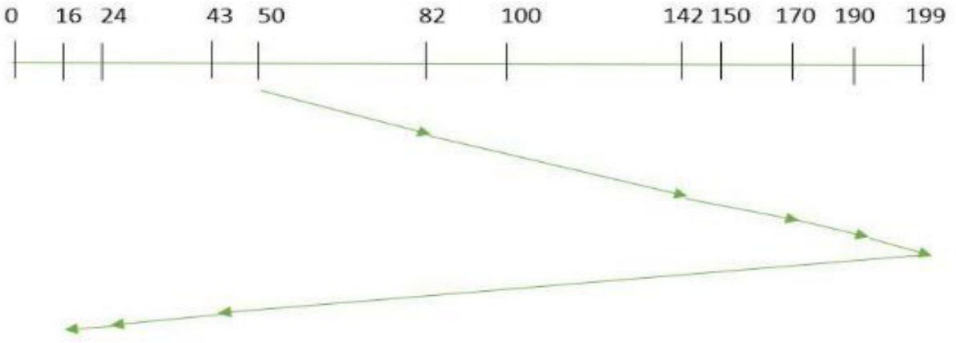
So, **total seek time**: =(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-40)+(190-170) = 208

SCAN Disk Scheduling:

- 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.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



SCAN Disk Scheduling

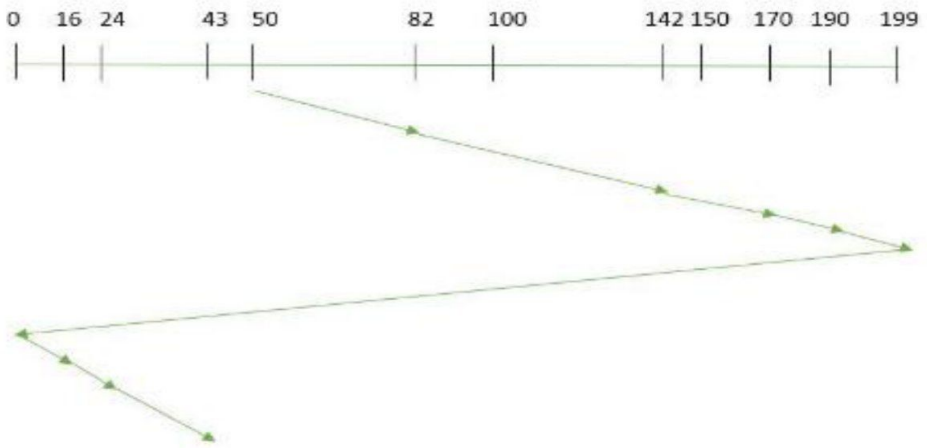
Therefore, the **seek time** = (199-50) + (199-16) = 332

CSCAN Disk Scheduling:

- 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).

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



CSCAN Disk Scheduling

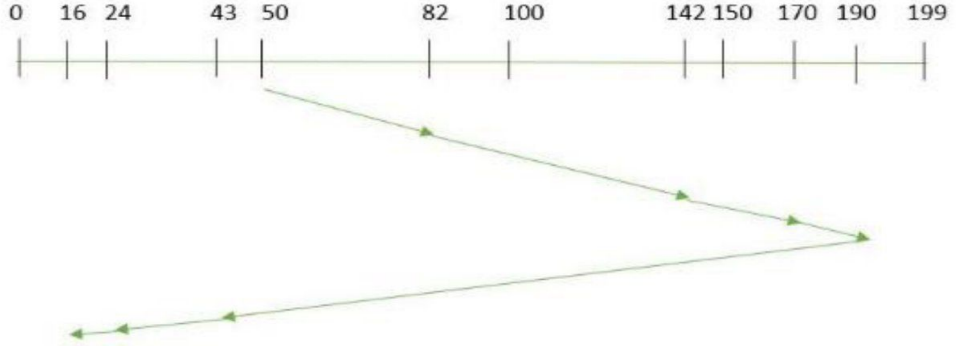
Seek time = (199-50) + (199-0) + (43-0) = 391

LOOK Disk Scheduling:

- 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:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



LOOK Disk Scheduling

So, the seek time = (190-50) + (190-16) = 314

Selection of a Disk Scheduling Algorithm:

- SSTF is commonly used and it increases performance over FCFS.
- SCAN and C-SCAN algorithm is better for a heavy load on disk.
- SCAN and C-SCAN have less starvation problem.
- Disk scheduling algorithm should be written as a separate module of the operating system.
- SSTF or Look is a reasonable choice for a default algorithm.
- SSTF is commonly used algorithms has it has a less seek time when compared with other algorithms.
- SCAN and C-SCAN perform better for systems with a heavy load on the disk, (ie. more read and write operations from disk).
- Selection of disk scheduling algorithm is influenced by the file allocation method, if contiguous file allocation is choosen, then FCFS is best suitable, because the files are stored in contiguous blicks and there will be limited head movements required.
- A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.
- The location of directories and index blocks is also important. Since every file must be opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently.
- Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder.
- The disk head has to move the entire width of the disk. If the directory entry were on the middle cylinder, the head would have to move, at most, one-half the width.
- Caching the directories and index blocks in main memory can also help to reduce the disk-arm movement, particularly for read requests.