Chapter 5: Storage Management

- Basic Concepts of File System
- File Access Methods
- Directory Structure
- File-System Implementation
- Allocation Methods
- Free Space Management
- Overview of Mass Storage Structure
- Disk Structure
- Disk Scheduling
- RAID Structure
- Introduction to I/O Systems

➢ Basic Concepts of File System

In operating systems, a file system is a way of organizing and storing files and directories on a storage device, such as a hard disk or solid-state drive. The file system provides a structure for organizing and accessing data, and it typically includes methods for creating, modifying, deleting, and accessing files and directories.

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From the user's perspective, a file is the smallest allotment of logical secondary storage.

➢ Files attributes

1. **Name**: Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.

2. **Identifier**: Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** A video file can have the extension **.mp4.**
3. **Type**: In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.
4. **Location**: In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.
5. **Size**: The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.
6. **Protection**: The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.
7. **Time and Date**
8. Every file carries a time stamp which contains the time and date on which the file is last modified.

➢ Files Operations

The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system. Some common operations are as follows:

1. **Create operation:** This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. **Open operation:** This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When

the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. **Write operation:** This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. **Read operation:** This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. **Re-position or Seek operation:** The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

6. **Delete operation:** Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

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

8. **Close operation:** When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

9. **Append operation:** This operation adds data to the end of the file.
10. **Rename operation:** This operation is used to rename the existing file.

- **Files types and its functions**

File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, 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, rrf, doc	Various word processor formats

Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a ,so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing a ASCII or binary file.

- **Files Structures**

  File types also can be used to indicate the internal structure of the file. Source and object files have structures that match the expectations of the programs that read them. Further, certain files must conform to a required structure that is understood by the operating system. For example, the operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is. Some operating systems extend this idea into a set of system-supported file structures, with sets of special operations for manipulating files with those structures.

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. Some systems provide only one access method for files. Others (such as mainframe operating systems) support many access methods, and choosing the right one for a particular application is a major design problem.

1. Sequential Access method

Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.

In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.

Modern word systems do provide the concept of direct access and indexed access but the most used method is sequential access due to the fact that most of the files such as text files, audio files, video files, etc need to be sequentially accessed.

2. Direct Access

The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.

Suppose every block of the storage stores 4 records and we know that the record we needed is stored in 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record.

Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.

3. **Indexed Access**
   If a file can be sorted on any of the filed then an index can be assigned to a group of certain records. However, A particular record can be accessed by its index. The index is nothing but the address of a record in the file.

   In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.

➢ Directory Structure

The directory can be viewed as a symbol table that translates file names into their file control blocks. If we take such a view, we see that the directory itself can be organized in many ways. The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory.
When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:

1. **Search for a file**: We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.
2. **Create a file**: New files need to be created and added to the directory.
3. **Delete a file**: When a file is no longer needed, we want to be able to remove it from the directory. Note a delete leaves a hole in the directory structure and the file system may have a method to defragment the directory structure.
4. **List a directory:** We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
5. **Rename a file**: Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
6. **Traverse the file system**: We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals. Often, we do this by copying all files to magnetic tape, other secondary storage, or across a network to another system or the cloud.

Type of Directory Structure

1. **Single-level directory**

   The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.

   A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name. if two users call their dataset test, then the unique name rule violated.

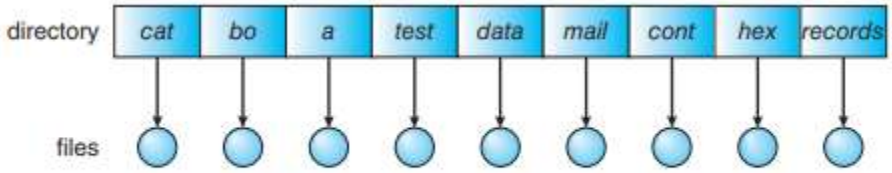


Figure 13.7 Single-level directory.

2. **Two-Level Directory**

   In a two-level directory structure, there is a master node that has a separate directory for each user. Each user can store the files in that directory. It can be practically thought of as a folder that contains many folders, each for a particular user, and now each user can store files in the allocated directory just like a single level directory.

   The pictorial representation of a two-level directory is shown below. For every user, there is a separate directory. At the next level, every directory stores the files just like a single-level directory. Although not very efficient, the two-level directory is better than a single-level directory structure.

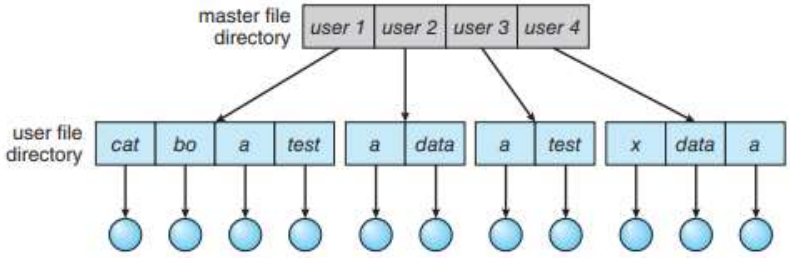


Figure 13.8 Two-level directory structure.

3. **Hierarchical / Tree structure Directory**

   This type of directory is used in our PCs. The biggest disadvantage of a two-level directory was that one could not create sub-directories in a directory. The tree-structured directory solved this problem. In a tree-structured directory, there is a root directory at the peak. The root directory contains directories for each user. The users can, however, create subdirectories inside their directory and also store the files.

This is how things work on our PCs. We can store some files inside a folder and also create multiple folders inside a folder.

The pictorial representation of a tree-structured directory is shown below. The root directory is highly secured, and only the system administrator can access it. We can see how there can be subdirectories inside the user directories. A user cannot modify the root directory data. Also, a user cannot access another user's directory.

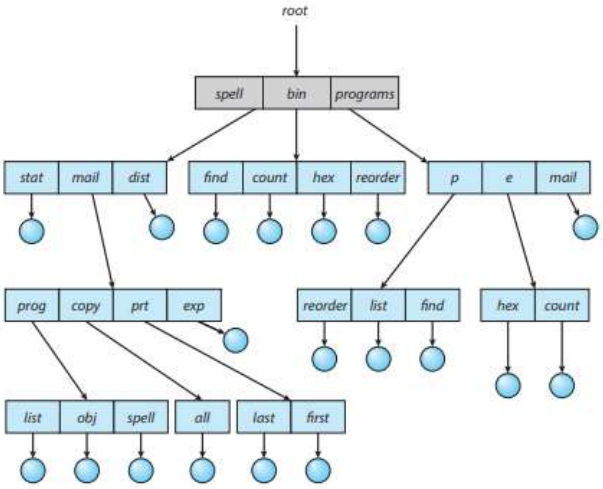


Figure 13.9 Tree-structured directory structure.

4. Acyclic-Graph Directories

Suppose there is a file abcd.txt. Out of the three types of directories we studied above, none of them provide the flexibility to access the file abcd.txt from multiple directories, i.e., we cannot access a particular file or subdirectory from two or more directories. The file or the subdirectory can be accessed only by the directory it is present inside.

The solution to this problem is presented by the acyclic-graph directory. In this type of directory, we can access a file or a subdirectory from multiple directories. Hence files can be shared between directories. It is designed in such a way that multiple directories point to a particular directory or file with the help of links.

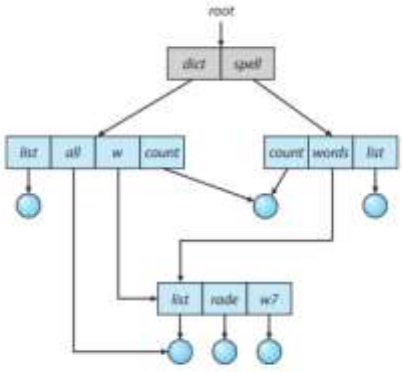


Figure 13.10 Acyclic-graph directory structure.

➢ File-System Implementation

File system is organized into many layers:

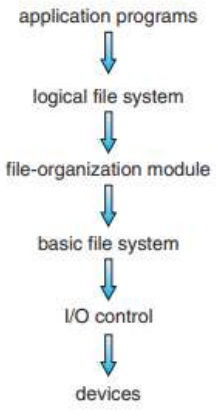


Figure 14.1 Layered file system.

1. **I/O Control level –** Device driver's acts as interface between devices and OS, they help to transfer data between disk and main memory. It takes block number as input and as output it gives low level hardware specific instruction.
2. **Basic file system –** It Issues general commands to device driver to read and write physical blocks on disk. It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.
3. **File organization Module –** It has information about files, location of files and their logical and physical blocks. Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.
4. **Logical file system –** It manages metadata information about a file i.e includes all details about a file except the actual contents of file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

File-System Operations

Several on-storage and in-memory structures are used to implement a file system. These structures vary depending on the operating system and the file system, but some general principles apply.

On storage, the file system may contain information about how to boot an operating system stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.

1. A boot control block (per volume) can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block

can be empty. It is typically the first block of a volume. In UFS, it is called the boot block. In NTFS, it is the partition boot sector.

2. A volume control block (per volume) contains volume details, such as the number of blocks in the volume, the size of the blocks, a free-block count and free-block pointers, and a free-FCB count and FCB pointers. In UFS, this is called a superblock. In NTFS, it is stored in the master fil table.
3. A per-file FCB contains many details about the file. It has a unique identifier number to allow association with a directory entry. In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file.

An in-memory mount table contains information about each mounted volume.

1. An in-memory directory-structure cache holds the directory information of recently accessed directories. (For directories at which volumes are mounted, it can contain a pointer to the volume table.)
2. The system-wide open-fil table contains a copy of the FCB of each open file, as well as other information.
3. The per-process open-fil table contains pointers to the appropriate entries in the system-wide open-file table, as well as other information, for all files the process has open.
4. Buffers hold file-system blocks when they are being read from or written to a file system.

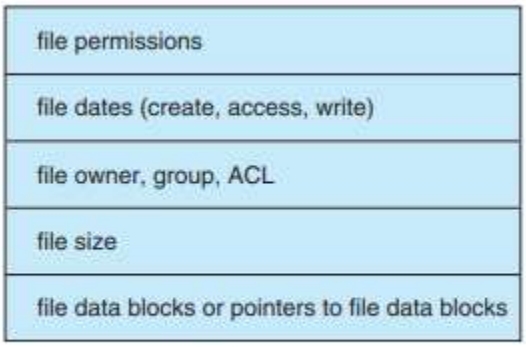
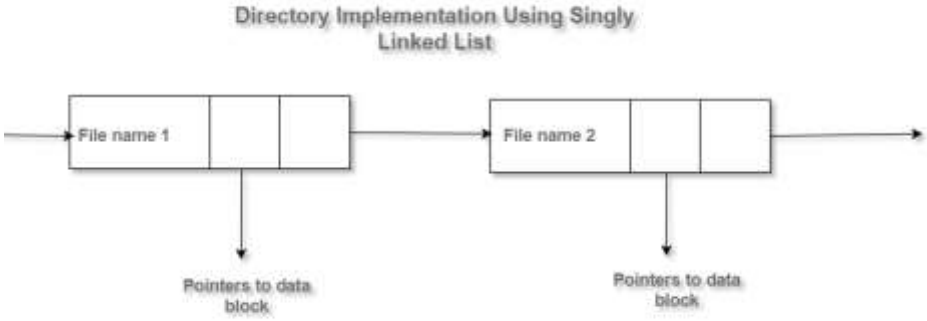


Figure 14.2 A typical file-control block.

Directory Implementation:

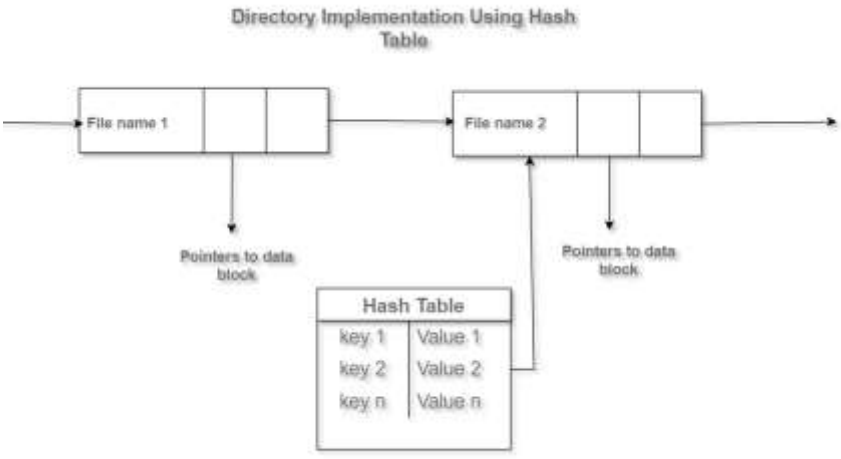
Directory implementation in the operating system can be done using Singly Linked List and Hash table. The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory-allocation and directory-management algorithms. There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.

1. **Directory Implementation using Singly Linked List:** The implementation of directories using a singly linked list is easy to program but is time-consuming to execute. Here we implement a directory by using a linear list of filenames with pointers to the data blocks.



- To create a new file the entire list has to be checked such that the new directory does not exist previously.
- The new directory then can be added to the end of the list or at the beginning of the list.
- In order to delete a file, we first search the directory with the name of the file to be deleted. After searching we can delete that file by releasing the space allocated to it.
- To reuse the directory entry we can mark that entry as unused or we can append it to the list of free directories.
- To delete a file linked list is the best choice as it takes less time.

2. **Directory Implementation using Hash Table:** An alternative data structure that can be used for directory implementation is a hash table. It overcomes the major drawbacks of directory implementation using a linked list. In this method, we use a hash table along with the linked list. Here the linked list stores the directory entries, but a hash data structure is used in combination with the linked list.



In the hash table for each pair in the directory key-value pair is generated. The hash function on the file name determines the key and this key points to the corresponding file stored in the directory. This method efficiently decreases the directory search time as the entire list will not be searched on every operation. Using the keys the hash table entries are checked and when the file is found it is fetched.

Partitions and Mounting

- The layout of a disk can have many variations, depending on the OS. A disk can be sliced into multiple partitions, or a volume can span multiple partitions on multiple disks (RAID).
- Each partition can be either ``raw'', containing no file system (UNIX swap space can use a raw partition), or ``cooked'', containing a file system.
- Multiple OSs can be installed. How does the system know which one to boot?
    - A boot loader that understands multiple file systems and multiple OSs can occupy the boot space.
    - Once loaded, it can boot one of the OSs available on the disk.
- The **root partition**, which contains the OS kernel and sometimes other system files, is mounted at boot time.
- Other volumes can be automatically mounted at boot or manually mounted later, depending on the OS.
- As part of a successful mount operation, the OS verifies that the device contains a valid file system. If the format is invalid, the partition must have its consistency checked and possibly corrected, either with or without user intervention.
- Finally, the OS notes in its in-memory mount table structure that a file system is mounted, along with the type of the file system.
- Microsoft Windows-based systems mount each volume in a separate name space, denoted by a letter and a colon (F:).
- On UNIX, file systems can be mounted at any directory. Mounting is implemented by setting a flag in the in-memory copy of the inode for that directory. The flag indicates that the directory is a mount point.
- The mount table entry contains a pointer to the superblock of the file system on that device.

Virtual File system

A virtual file system (VFS) is an abstraction layer in an operating system (OS) that allows different types of file systems to be supported without the need for modifications to the core OS. It provides a common interface to various file systems, hiding the details of specific file system implementations.

The VFS layer sits between the file system drivers and the user applications, and it is responsible for translating file system requests from applications into the appropriate format for the specific file system being used. It also performs caching of frequently used files to improve performance and manages access to files and directories to ensure security.

The advantages of using a virtual file system include the ability to support different file systems, such as network file systems or compressed file systems, without modifying the core OS. Additionally, it allows applications to access files and directories in a consistent manner, regardless of the underlying file system being used.

Overall, the virtual file system plays a critical role in the functioning of modern operating systems, enabling users to interact with files and directories in a seamless and intuitive manner while also providing a flexible and extensible platform for developers to build on.

➢ Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

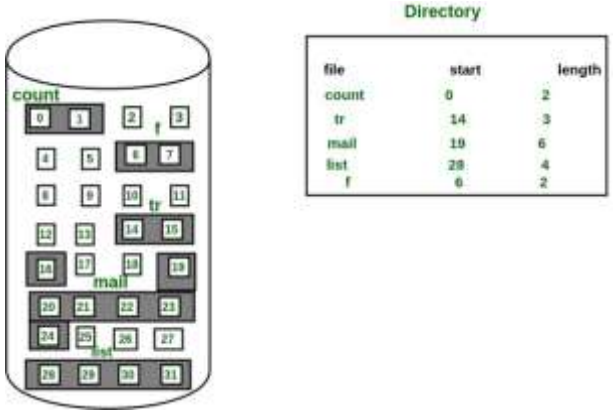
The main idea behind these methods is to provide efficient disk space utilization and fast access to the file blocks.

a. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

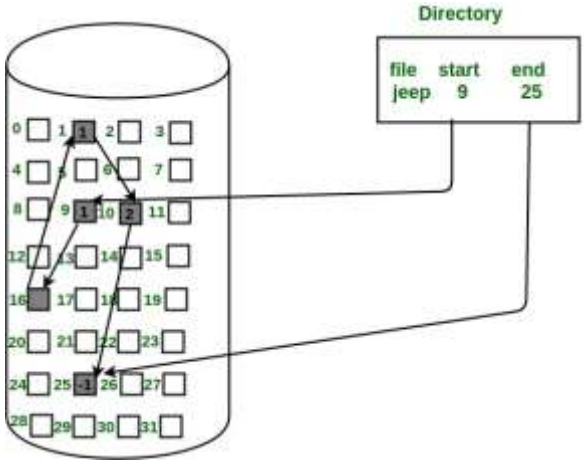
The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.



b. Linked List Allocation

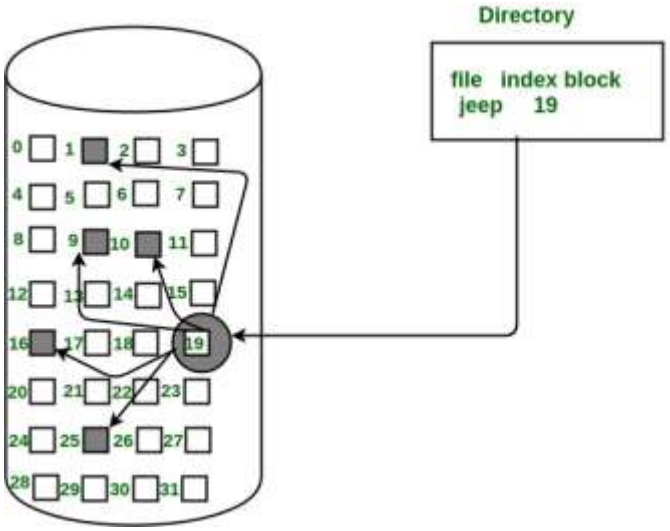
In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



c. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



➢ Free Space Management

Free space management is a critical aspect of operating systems as it involves managing the available storage space on the hard disk or other secondary storage devices. The operating system uses various techniques to manage free space and optimize the use of storage devices. Here are some of the commonly used free space management techniques:

1. Linked Allocation: In this technique, each file is represented by a linked list of disk blocks. When a file is created, the operating system finds enough free space on the disk and links the blocks of the file to form a chain. This method is simple to implement but can lead to fragmentation and wastage of space.

2. Contiguous Allocation: In this technique, each file is stored as a contiguous block of disk space. When a file is created, the operating system finds a contiguous block of free space and assigns it to the file. This method is efficient as it minimizes fragmentation but suffers from the problem of external fragmentation.

3. Indexed Allocation: In this technique, a separate index block is used to store the addresses of all the disk blocks that make up a file. When a file is created, the operating system creates an index block and stores the addresses of all the blocks in the file. This method is efficient in terms of storage space and minimizes fragmentation.

4. File Allocation Table (FAT): In this technique, the operating system uses a file allocation table to keep track of the location of each file on the disk. When a file is created, the operating system updates the file allocation table with the address of the disk blocks that make up the file. This method is widely used in Microsoft Windows operating systems.

5. Volume Shadow Copy: This is a technology used in Microsoft Windows operating systems to create backup copies of files or entire volumes. When a file is modified, the operating system creates a shadow copy of the file and stores it in a separate location. This method is useful for data recovery and protection against accidental file deletion.

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. **Bitmap or Bit vector –** A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block. The given instance of disk blocks on the disk in Figure 1 (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

Figure - 1

Advantages –

a. Simple to understand.
b. Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

2. **Linked List –** In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

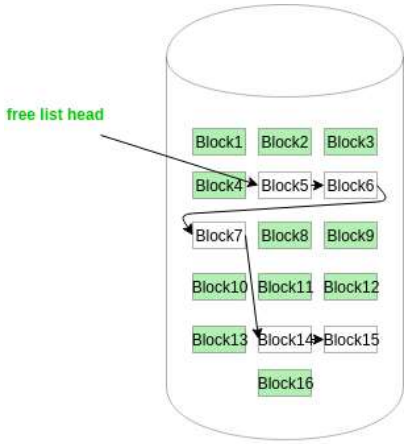


Figure - 2

In Figure-2, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list. A drawback of this method is the I/O required for free space list traversal.

3. **Grouping –** This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.

4. **Counting –** This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block. Every entry in the list would contain:
    a. Address of first free disk block
    b. A number n

➢ Overview of Mass Storage Structure

The bulk of secondary storage for modern computers is provided by hard disk drives (HDDs) and nonvolatile memory (NVM) devices.

1. Hard Disk Drive

Conceptually, HDDs are relatively simple (Figure 11.1). Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters, and we read information by detecting the magnetic pattern on the platters.

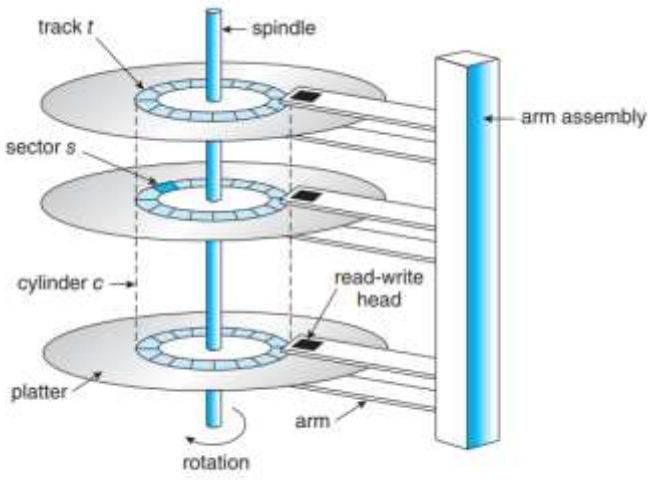
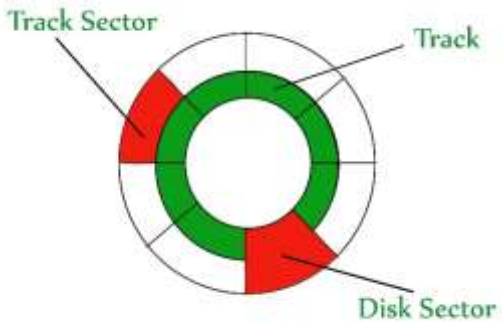


Figure 11.1 HDD moving-head disk mechanism.

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 at a given arm position make up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. Each sector has a fixed size and is the smallest unit of transfer.

➢ Disk Structure

A hard disk is a memory storage device that looks like this:



The disk is divided into **tracks**. Each track is further divided into **sectors**. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity. This is because the storage density is high in sectors of the inner tracks whereas the bits are sparsely arranged in sectors of the outer tracks. Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity.

Read-Write(R-W) head moves over the rotating hard disk. It is this Read-Write head that performs all the read and writes operations on the disk and hence, the position of the R-W head is a major concern. To perform a read or write operation on a memory location, we need to place the R-W head over that position. Some important terms must be noted here:

- Seek time – The time taken by the R-W head to reach the desired track from its current position.
- Rotational latency – Time is taken by the sector to come under the R-W head.
- Data transfer time – Time is taken to transfer the required amount of data. It depends upon the rotational speed.
- Controller time – The processing time taken by the controller.
- Average Access time – seek time + Average Rotational latency + data transfer time + controller time.

➢ 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:

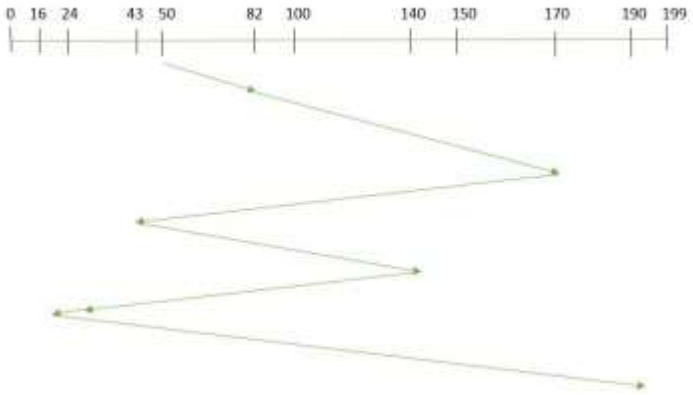
Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

Two or more requests 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.

Following are the Disk Scheduling Algorithms

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.

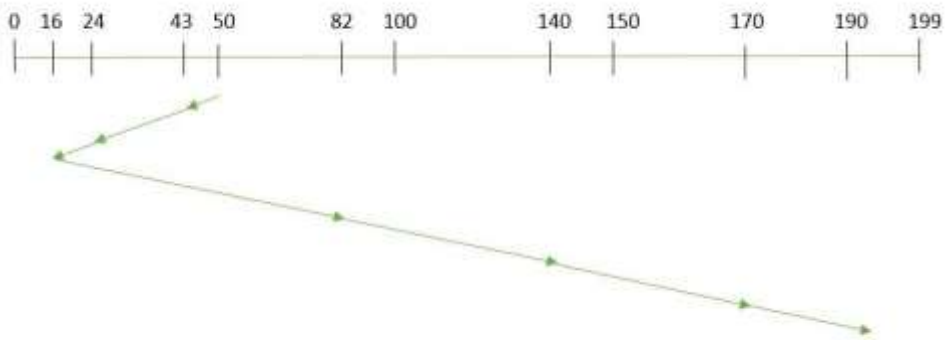
   Example: Suppose the order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is: 50



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

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.

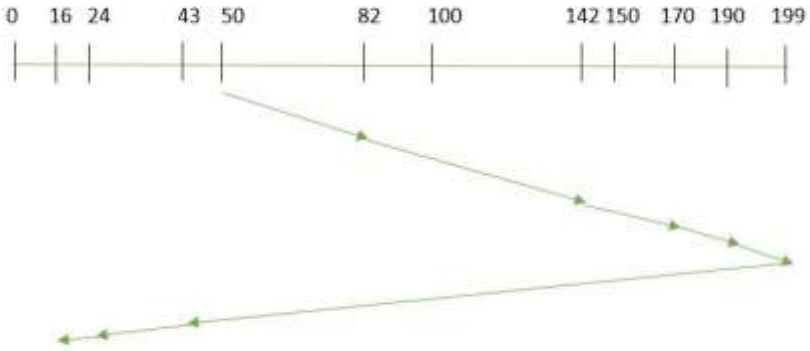
Example: Suppose the order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is : 50



total seek time = (50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

3. SCAN: In SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an 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”.



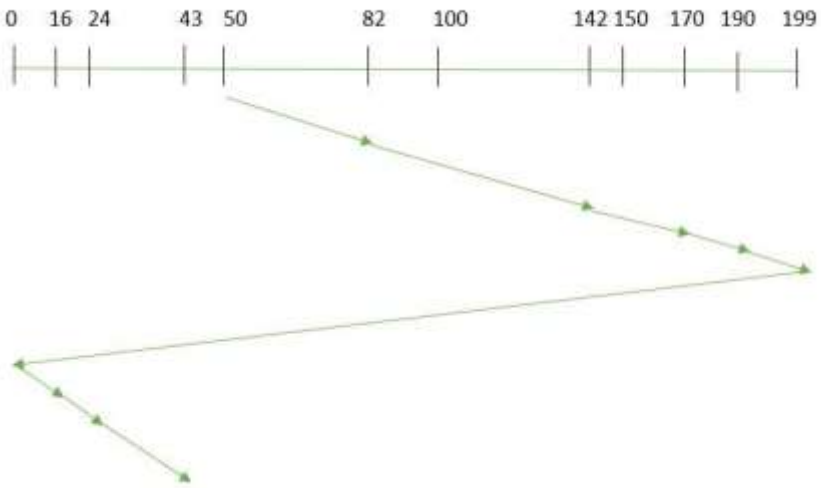
   Therefore, the seek time is calculated as: =(199-50)+(199-16) =332

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.

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”.

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”.**



Seek time is calculated as: =(199-50)+(199-0)+(43-0) =391