**UNIT-II**

**File Concept: -**A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

Following are some common file attributes:

**File StructName:** File name is the name given to the file. A name is usually a string of characters.

**Identifier:** Identifier is a unique number for a file. It identifies files within the file system. It is not readable to us, unlike file names.

**Type:** Type is another attribute of a file which specifies the type of file such as archive file (.zip), source code file (.c, .java), .docx file, .txt file, etc.

**Location:** Specifies the location of the file on the device (The directory path). This attribute is a pointer to a device.

**Size:** Specifies the current size of the file (in Kb, Mb, Gb, etc.) and possibly the maximum allowed size of the file.

**Protection:**Specifies information about Access control (Permissions about Who can read, edit, write, and execute the file.) It provides security to sensitive and private information.

**Time, date, and user identification:** This information tells us about the date and time on which the file was created, last modified, created and modified by which user, etc.

A File Structure should be according to a required format that the operating system can understand.

* A file has a certain defined structure according to its type.
* A text file is a sequence of characters organized into lines.
* A source file is a sequence of procedures and functions.
* An object file is a sequence of bytes organized into blocks that are understandable by the machine.
* When operating system defines different file structures, it also contains the code to support these file structure. UNIX, MS-DOS support minimum number of file structure.

**File Type**

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

## Ordinary files

* These are the files that contain user information.
* These may have text, databases or executable program.
* The user can apply various operations on such files like add, modify, delete or even remove the entire file.

## Directory files

* These files contain list of file names and other information related to these files.

## Special files

* These files are also known as device files.
* These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

* **Character special files**− data is handled character by character as in case of terminals or printers.
* **Block special files**− data is handled in blocks as in the case of disks and tapes.

## User’s and System Programmer’s view of File System

**User View**

The user view of the computer refers to the interface being used. Such systems are designed for one user to monopolize its resources, to maximize the work that the user is performing. In these cases, the operating system is designed mostly for ease of use, with some attention paid to performance, and none paid to resource utilization.

## System View

Operating system can be viewed as a resource allocator also. A computer system consists of many resources like - hardware and software - that must be managed efficiently. The operating system acts as the manager of the resources, decides between conflicting requests, controls execution of programs etc.

## Disk Organization

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

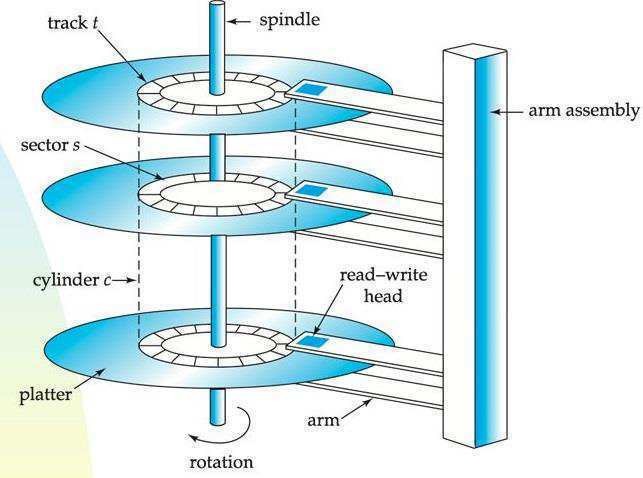


Fig 2.0 Hard disk

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 where as 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](https://be.rgpvnotes.in/) 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 write operations on the disk and hence, 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:

1. **Seek time –** The time taken by the R-W head to reach the desired track from it’s current

position.

1. **Rotational latency –** Time taken by the sector to come under the R-W head.
2. **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
3. **Controller time –** The processing time taken by the controller.
4. **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time.

## Different Modules of a File System:

* + **The basic file system level** works directly with the device drivers in terms of retrieving and storing raw blocks of data, without any consideration for what is in each block. Depending on the system, blocks may be referred to with a single block number or with head-sector-cylinder combinations.



* + **The file organization module** knows about files and their logical blocks, and how they map to physical blocks on the disk. In addition to translating from logical to physical blocks, the file organization module also maintains the list of free blocks, and allocates free blocks to files as needed.
  + **The logical file system** deals with all of the meta data associated with a file ( UID, GID, mode, dates, etc ), i.e. everything about the file except the data itself. This level manages the directory structure and the mapping of file names to file control blocks, FCBs, which contain all of the Meta data as well as block number information for finding the data on the disk.
  + The layered approach to file systems means that much of the code can be used uniformly for a wide variety of different file systems, and only certain layers need to be file system specific. Common file systems in use include the UNIX file system, UFS, the Berkeley Fast File System, FFS, Windows systems FAT, FAT32, NTFS, CD-ROM systems ISO 9660, and for Linux the extended file systems ext2 and ext3 .

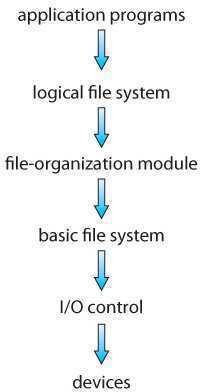


Fig 2.1 - Layered file system.

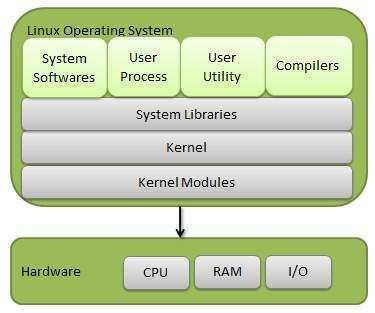
## File system in Linux & Windows Linux

Linux is one of popular version of UNIX oerating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

## Linux Operating System has primarily three components

* **Kernel**− Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
* **System Library**− System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.
* **System Utility**− System Utility programs are responsible to do specialized, individual level tasks.



## Fig 2.2 Linux Operating System

**Directory structure**

The collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

**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.



**Advantages:**

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.

**Increased Efficiency:** Directory structures can increase the efficiency of the file system by reducing the time required to search for files.

**Improved Security**: Directory structures can provide better security for files by allowing access to be restricted at the directory level. This helps to prevent unauthorized access to sensitive data and ensures that important files are protected.

**Facilitates Backup and Recovery**: Directory structures make it easier to backup and recover files in the event of a system failure or data loss. By storing related files in the same directory, it is easier to locate and backup all the files that need to be protected.

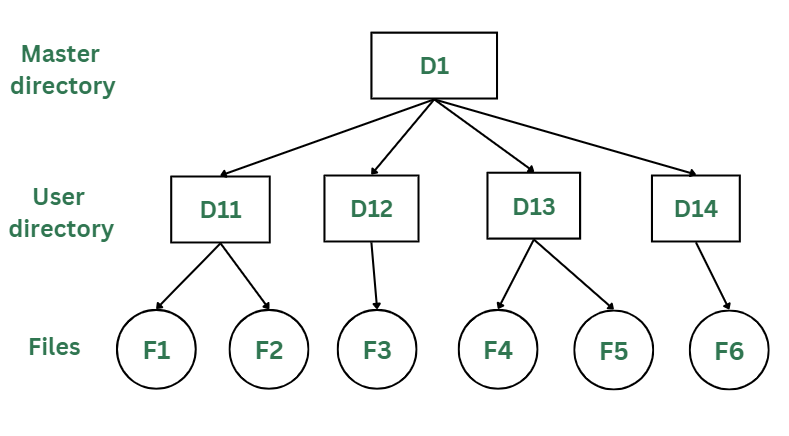
**Disadvantages:**

* There may chance of name collision because two files can 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:**

As we have seen, a single level directory often leads to confusion of files names among different users. The solution to this problem is to create a **separate directory for each user**.

In the two-level directory structure, each user has their own **user files directory (UFD).** The UFDs have similar structures, but each lists only the files of a single user. System’s **master file directory (MFD)** is searched whenever a new user id is created.



**Advantages:**

* The main advantage is there can be more than two files with same name, and would be very helpful if there are multiple users.
* A security would be there which would prevent user to access other user’s files.
* Searching of the files becomes very easy in this directory structure.

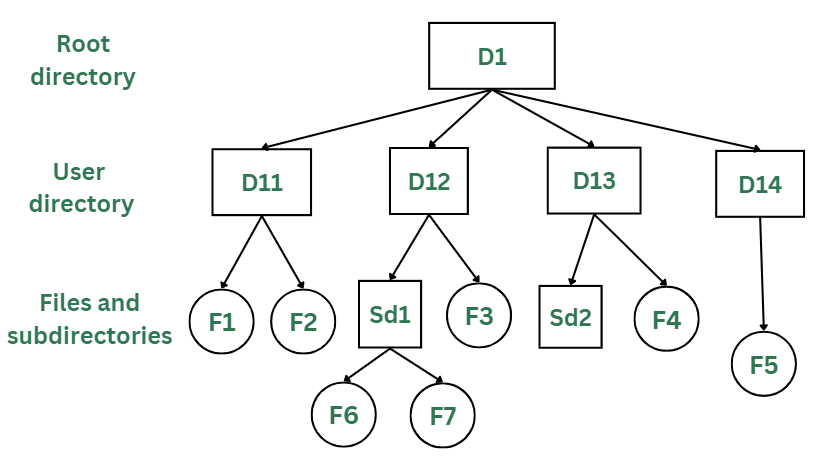
**Disadvantages:**

* As there is advantage of security, there is also disadvantage that the user cannot share the file with the other users.
* Unlike the advantage users can create their own files, users don’t have the ability to create subdirectories.
* Scalability is not possible because one use can’t group the same types of files together.

**3) Tree Structure/ Hierarchical Structure:**

Tree directory structure of operating system is most commonly used in our **personal computers**. User can create files and subdirectories too, which was a disadvantage in the previous directory structures.

This directory structure resembles a real tree upside down, where the **root directory** is at the peak. This root contains all the directories for each user. The users can create subdirectories and even store files in their directory.



**Advantages:**

* This directory structure allows subdirectories inside a directory.
* The searching is easier.
* File sorting of important and unimportant becomes easier.
* This directory is more scalable than the other two directory structures explained.

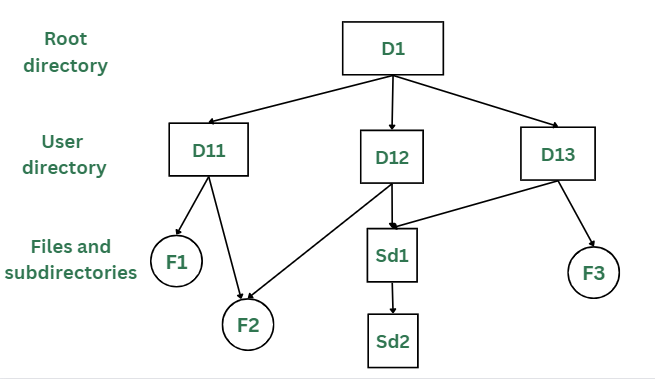
**Disadvantages:**

* As the user isn’t allowed to access other user’s directory, this prevents the file sharing among users.
* As the user has the capability to make subdirectories, if the number of subdirectories increase the searching may become complicated.
* Users cannot modify the root directory data.
* If files do not fit in one, they might have to be fit into other directories.

**4) Acyclic Graph Structure:**

As we have seen the above three directory structures, where none of them have the capability to access one file from multiple directories. The file or the subdirectory could be accessed through the directory it was present in, but not from the other directory.

This problem is solved in acyclic graph directory structure, where a file in one directory can be accessed from multiple directories. In this way, the files could be shared in between the users. It is designed in a way that multiple directories point to a particular directory or file with the help of links.



**Advantages:**

* Sharing of files and directories is allowed between multiple users.
* Searching becomes too easy.
* Flexibility is increased as file sharing and editing access is there for multiple users.

**Disadvantages:**

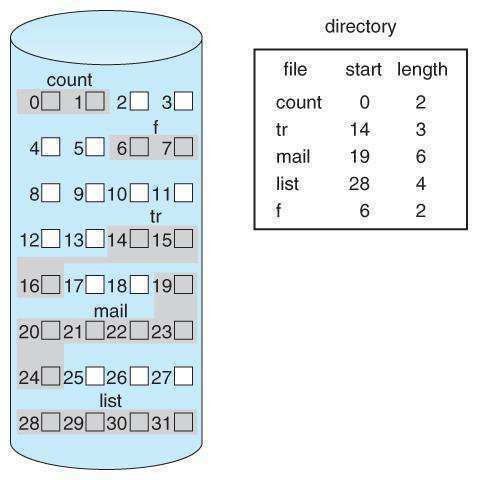
* Because of the complex structure it has, it is difficult to implement this directory structure.
* The user must be very cautious to edit or even deletion of file as the file is accessed by multiple users.
* If we need to delete the file, then we need to delete all the references of the file inorder to delete it permanently.

**Disk Space Allocation Methods:**

There are three major methods of storing files on disks: contiguous, linked, and indexed.

**Contiguous Allocation**

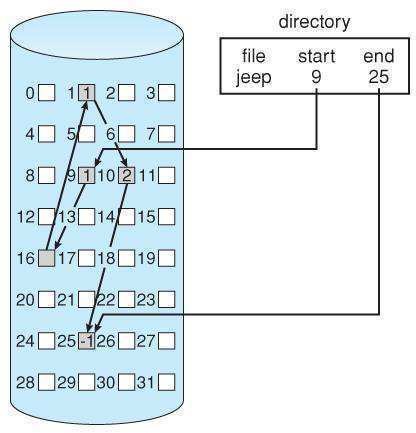
* + ***Contiguous Allocation*** requires that all blocks of a file be kept together contiguously.
  + Performance is very fast, because reading successive blocks of the same file generally requires no movement of the disk heads, or at most one small step to the next adjacent cylinder.
  + Storage allocation involves the same issues discussed earlier for the allocation of contiguous blocks of memory (first fit, best fit, fragmentation problems, etc.) The distinction is that the high time penalty required for moving the disk heads from spot to spot may now justify the benefits of keeping files contiguously when possible.
  + (Even file systems that do not by default store files contiguously can benefit from certain utilities that compact the disk and make all files contiguous in the process.)
  + Problems can arise when files grow, or if the exact size of a file is unknown at creation time:



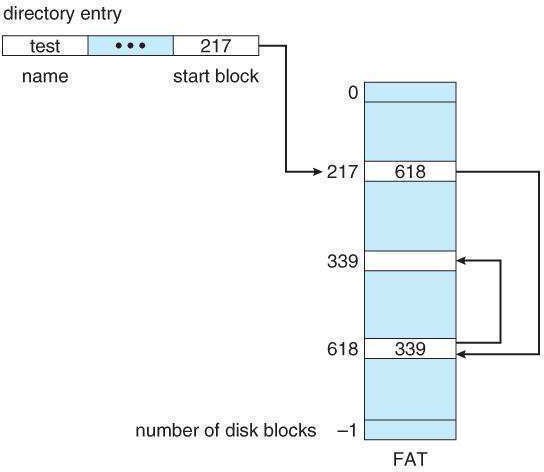
**Fig 2.5 Contiguous allocation of disk space.**

## Linked Allocation

* + Disk files can be stored as linked lists, with the expense of the storage space consumed by each link. ( E.g. a block may be 508 bytes instead of 512. )
  + Linked allocation involves no external fragmentation, does not require pre-known file sizes, and allows files to grow dynamically at any time.
  + Unfortunately linked allocation is only efficient for sequential access files, as random access requires starting at the beginning of the list for each new location access.
  + Allocating ***clusters*** of blocks reduces the space wasted by pointers, at the cost of internal fragmentation.
  + Another big problem with linked allocation is reliability if a pointer is lost or damaged. Doubly linked lists provide some protection, at the cost of additional overhead and wasted space.



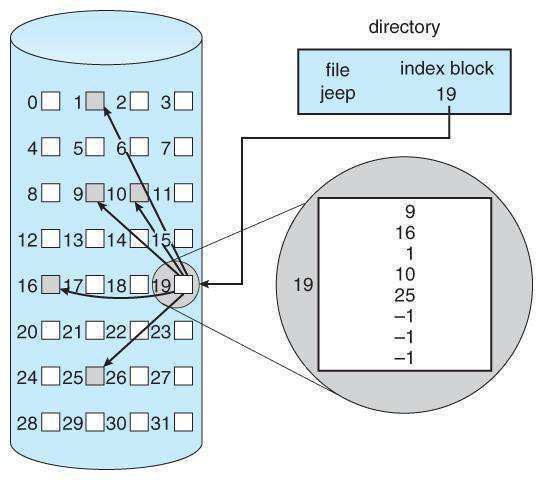
## Fig 2.6 - Linked allocation of disk space.

* + The ***File Allocation Table, FAT,*** used by DOS is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the disk. The benefit of this approach is that the FAT table can be cached in memory, greatly improving random access speeds.

## Fig 2.7 File-allocation table.

**Indexed Allocation**

**Indexed Allocation** combines all of the indexes for accessing each file into a common block (for that file), as opposed to spreading them a



**.**

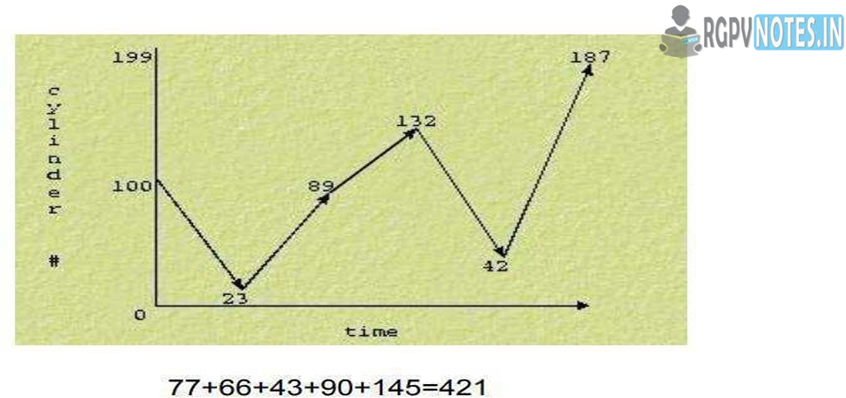
**Disk Scheduling Algorithms**

I/O request issues a system call to the OS. If desired disk drive or controller is available, request is served immediately. If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the OS has to choose which pending request to service next.

**FCFS Scheduling**

Simplest, perform operations in order requested no reordering of work queue „ no starvation: every request is serviced „ Doesn’t provide fastest service Ex: a disk queue with requests for I/O to blocks on cylinders 23, 89, 132, 42, 187 with disk head initially at 100

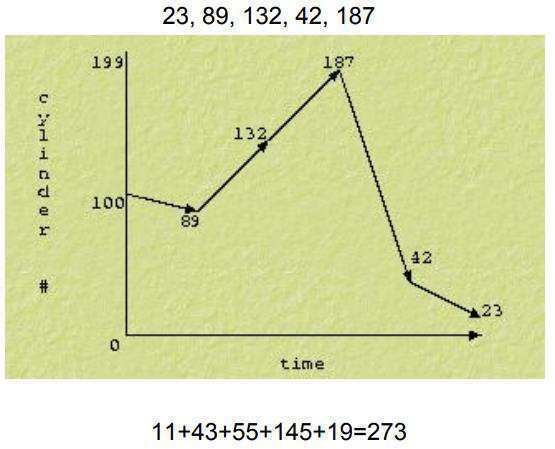
FCFS 23, 89, 132, 42, 187



If the requests for cylinders 23 and 42 could be serviced together, total head movement could be decreased substantially.

## SSTF Scheduling

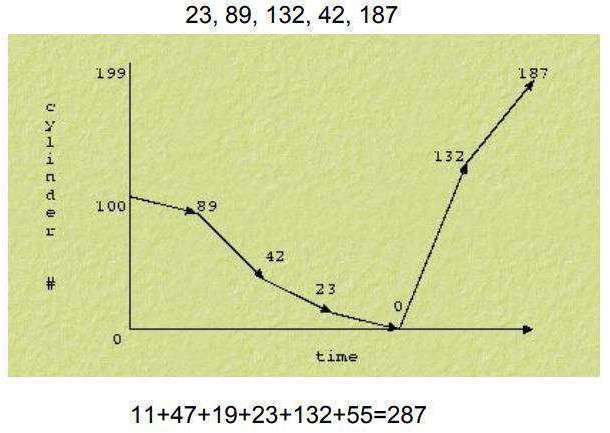
Like SJF, select the disk I/O request that requires the least movement of the disk arm from its current position, regardless of direction reduces total seek time compared to FCFS. Disadvantages starvation is possible; stay in one area of the disk if very busy switching directions slow things down not the most optimal.



## Fig 2.10 SSTF Scheduling

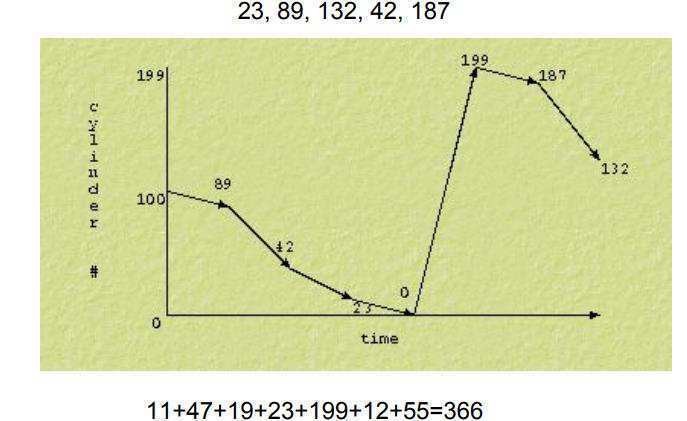
**SCAN**

Go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests. Sometimes called the elevator algorithm Reduces variance compared to SSTF. If a request arrives in the queue just in front of the head ‰ Just behind



**C-SCAN**

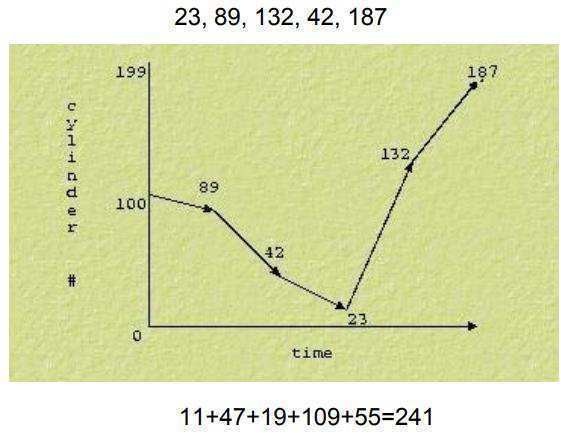
Circular SCAN „ moves inwards servicing requests until it reaches the innermost cylinder; then jumps to the outside cylinder of the disk without servicing any requests. „ Why C-SCAN? ‰ Few requests are in front of the head, since these cylinders have recently been serviced. Hence provides a more uniform wait time.



## Fig 2.12 C-SCAN Scheduling

**LOOK**

Like SCAN but stops moving inwards (or outwards) when no more requests in that direction exist.



Compared to SCAN, LOOK saves going from 23 to 0 and then back. Most efficient for this sequence of requests

## File Protection

When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection). Reliability is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed. File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally. Bugs in the file-system software can also cause file contents to be lost. Protection can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multiuser system, however, other mechanisms are needed.

## System calls for File Management

* **System call OPEN**

Opening or creating a file can be done using the system call open. The syntax is:

#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h>

**int** open(**const char** \*path,

**int** flags,... /\* mode\_t mod \*/);

## System call CREAT

A new file can be created by:

#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h>

**int** creat(**const char** \*path, **mode\_t** mod);

## System call READ

When we want to read a certain number of bytes starting from the current position in a file, we use the *read* call. The syntax is:

#include <unistd.h>

**ssize\_t** read(**int** fd, **void\*** buf, **size\_t** noct);

## System call WRITE

For writing a certain number of bytes into a file starting from the current position we use the *write* call. Its syntax is:

#include <unistd.h>

**ssize\_t** write(**int** fd, **const void\*** buf, **size\_t** noct);

## System call CLOSE

For closing a file and thus eliminating the assigned descriptor we use the system call *close*.

#include <unistd.h>

**int** close(int fd);

## System call LSEEK

To position a pointer that points to the current position in an absolute or relative way can be done by calling the *lseek* function. Read and write operations are done relative to the current position in the file. The syntax for *lseek* is:

#include <sys/types.h> #include <unistd.h>

**off\_t** lseek(**int** fd, **off\_t** offset, **int** ref);

## System call LINK

To link an existing file to another directory (or to the same directory) link can be used. To make such a link in fact means to set a new name or a path to an existing file. The *link* system call creates a hard link. Creating symbolic links can be done using *symlink* system call. The syntax of link is:

#include <unistd.h>

**int** link(**const char\*** oldpath, **const char\*** newpath);

**int** symlink(**const char\*** oldpath,

**File access method**

There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.

**Sequential Access –**   
It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file. A read operation *-read next-* read the next position of the file and automatically advance a file pointer, which keeps track I/O location. Similarly, for the -write*next-* append to the end of the file and advance to the newly written material.

**Key points:**

Data is accessed one record right after another record in an order.

When we use read command, it move ahead pointer by one

When we use write command, it will allocate memory and move the pointer to the end of the file

Such a method is reasonable for tape. 

**Advantages of Sequential Access Method :**

It is simple to implement this file access mechanism.

It uses lexicographic order to quickly access the next entry.

It is suitable for applications that require access to all records in a file, in a specific order.

It is less prone to data corruption as the data is written sequentially and not randomly.

It is a more efficient method for reading large files, as it only reads the required data and does not waste time reading unnecessary data.

It is a reliable method for backup and restore operations, as the data is stored sequentially and can be easily restored if required.

**Disadvantages of Sequential Access Method :**

If the file record that needs to be accessed next is not present next to the current record, this type of file access method is slow.

Moving a sizable chunk of the file may be necessary to insert a new record.

It does not allow for quick access to specific records in the file. The entire file must be searched sequentially to find a specific record, which can be time-consuming.

It is not well-suited for applications that require frequent updates or modifications to the file. Updating or inserting a record in the middle of a large file can be a slow and cumbersome process.

Sequential access can also result in wasted storage space if records are of varying lengths. The space between records cannot be used by other records, which can result in inefficient use of storage.

**2.Direct Access –**   
Another method is *direct access method* also known as *relative access method*. A fixed-length logical record that allows the program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59, and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.   
A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on. 

**Advantages of Direct Access Method :**

The files can be immediately accessed decreasing the average access time.

In the direct access method, in order to access a block, there is no need of traversing all the blocks present before it.

**3.Index sequential method –**   
It is the other method of accessing a file that is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index, and then by the help of pointer we access the file directly.

**Key points:**

It is built on top of Sequential access.

It control the pointer by using index.