RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Operating System**

Subject Code: **CS-405**

Semester: **4th**

CS-405 Operating System

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.

File Structure

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 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:

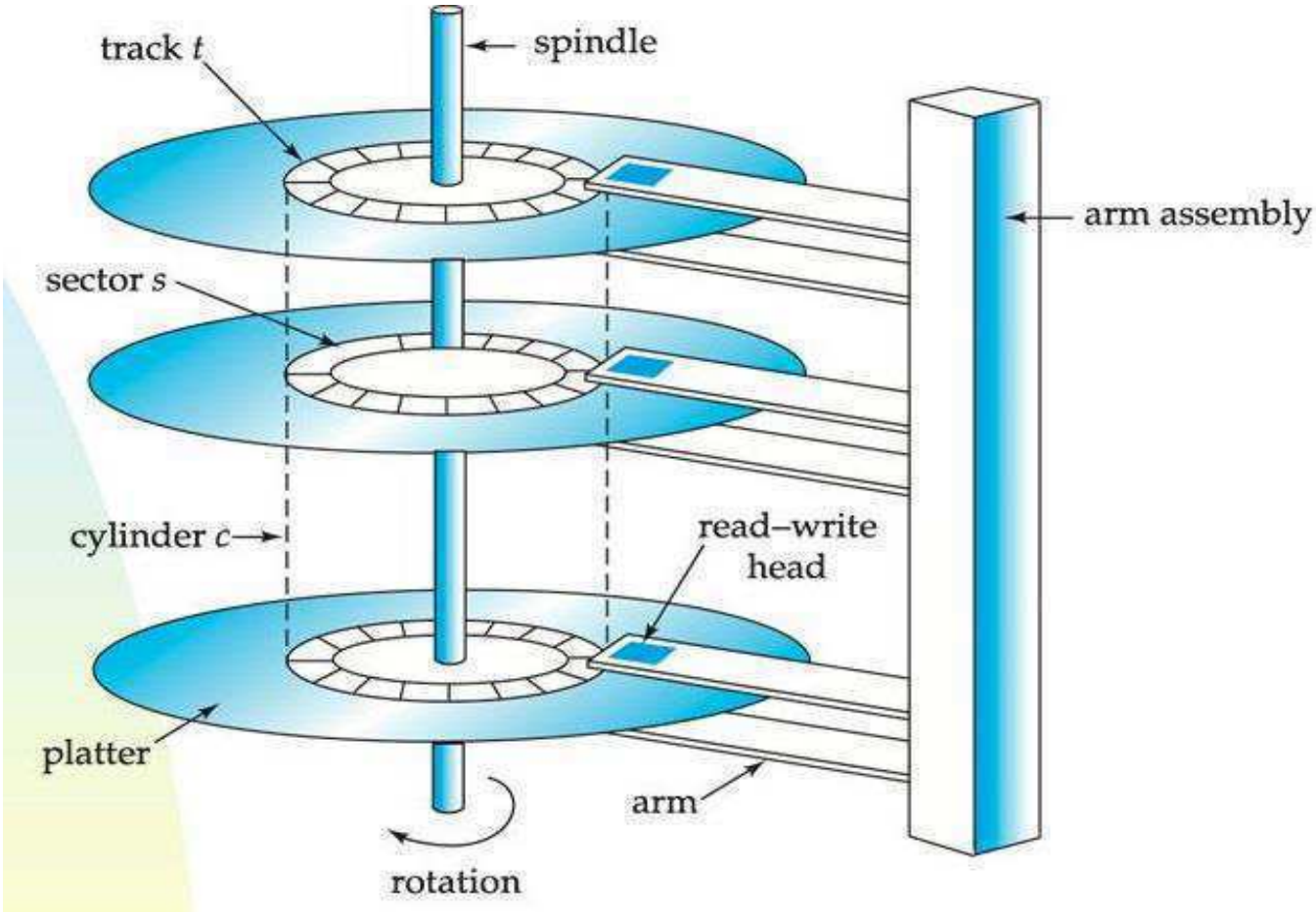


Fig 2.0 Hard disk

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 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.
2. **Rotational latency –** Time taken by the sector to come under the R-W head.
3. **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
4. **Controller time –** The processing time taken by the controller.
5. **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 .

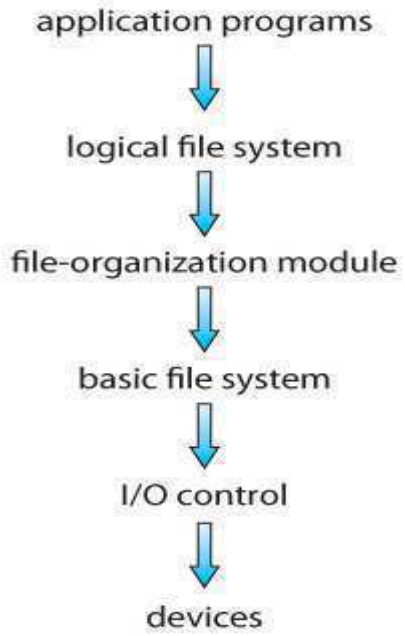


Fig 2.1 - Layered file system.

File system in Linux & Windows

Linux

Linux is one of popular version of UNIX operating 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.

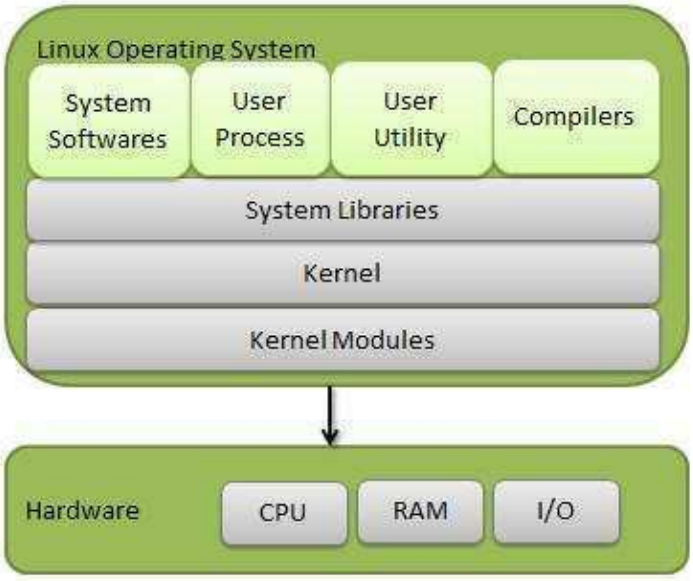


Fig 2.2 Linux Operating System

Directory systems

A directory is a location for storing files on your computer. Directories are found in a hierarchical file system, such as Linux, MS-DOS, OS/2, and Unix.

- A collection of nodes containing information about all files
- A directory system can be classified in to single level and hierarchical directory system:

Single level directory system: In this type of directory system, there is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. This type of directory system is used in cameras and phones.

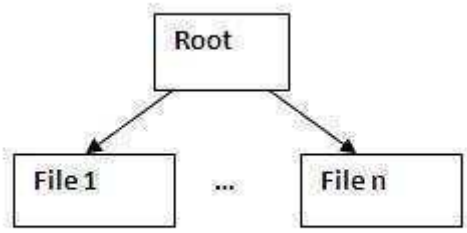


Fig 2.3 Directory System

Hierarchical directory system: In a hierarchical directory system, files are grouped together to form a sub directory at the top of the hierarchy is the root directory and then there are sub directories which has files. Advantage of hierarchical directory system is that users can be provided access to a sub directory rather than the entire directory. It provides a better structure to file system. Also, managing millions of files is easy with this architecture. Personal computers use hierarchical directory system for managing files.

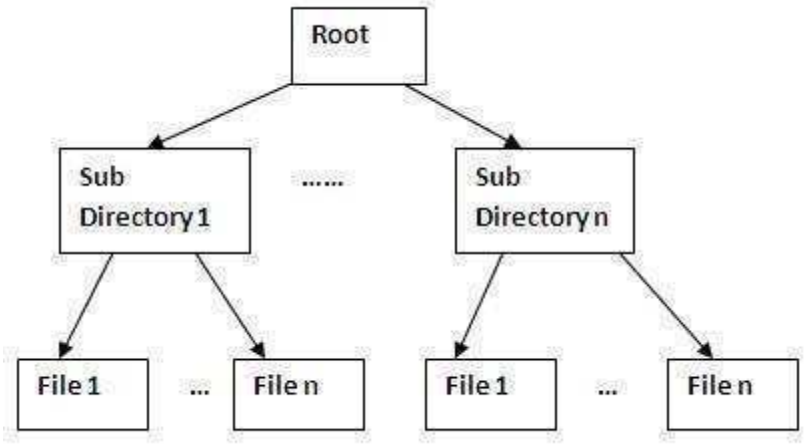


Fig 2.4 Hierarchical directory system

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:
    - Over-estimation of the file's final size increases external fragmentation and wastes disk space.
    - Under-estimation may require that a file be moved or a process aborted if the file grows beyond its originally allocated space.
    - If a file grows slowly over a long time period and the total final space must be allocated initially, then a lot of space becomes unusable before the file fills the space.
- A variation is to allocate file space in large contiguous chunks, called ***extents.*** When a file outgrows its original extent, then an additional one is allocated. (For example an extent may be the size of a complete track or even cylinder, aligned on an appropriate track or cylinder boundary.) The high-performance files system Veritas uses extents to optimize performance.

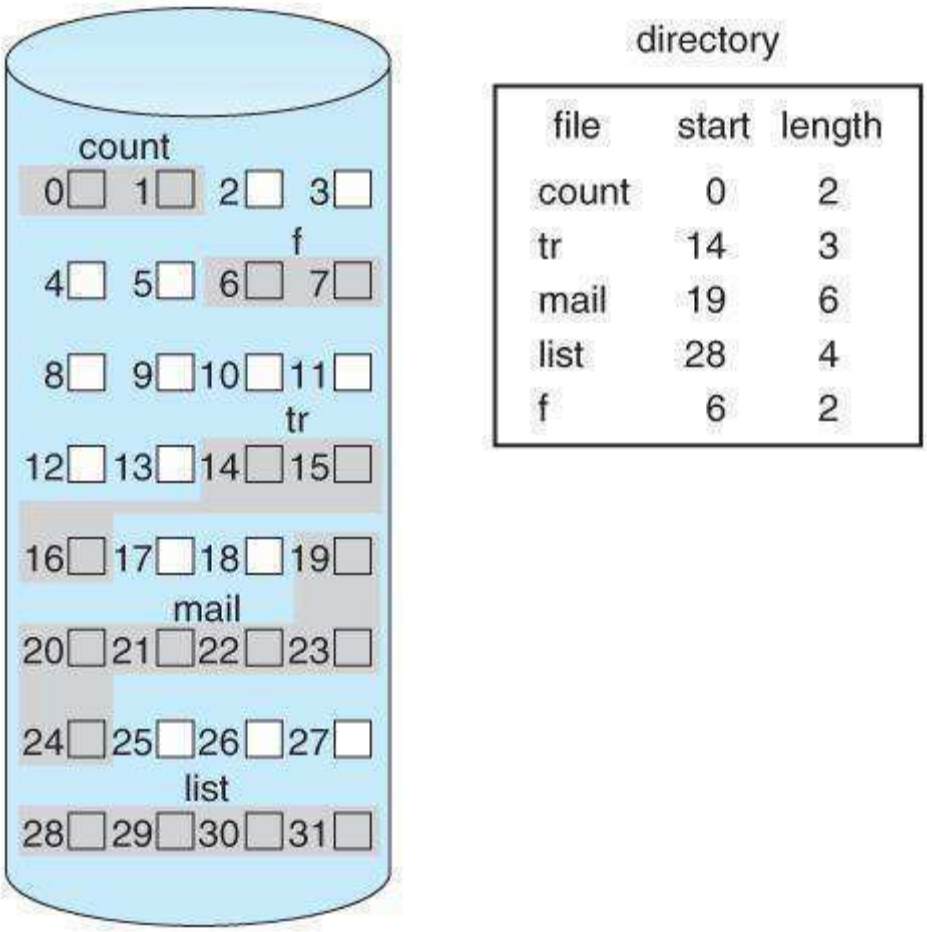


Fig 2.5 Contiguous allocation of disk space.

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.

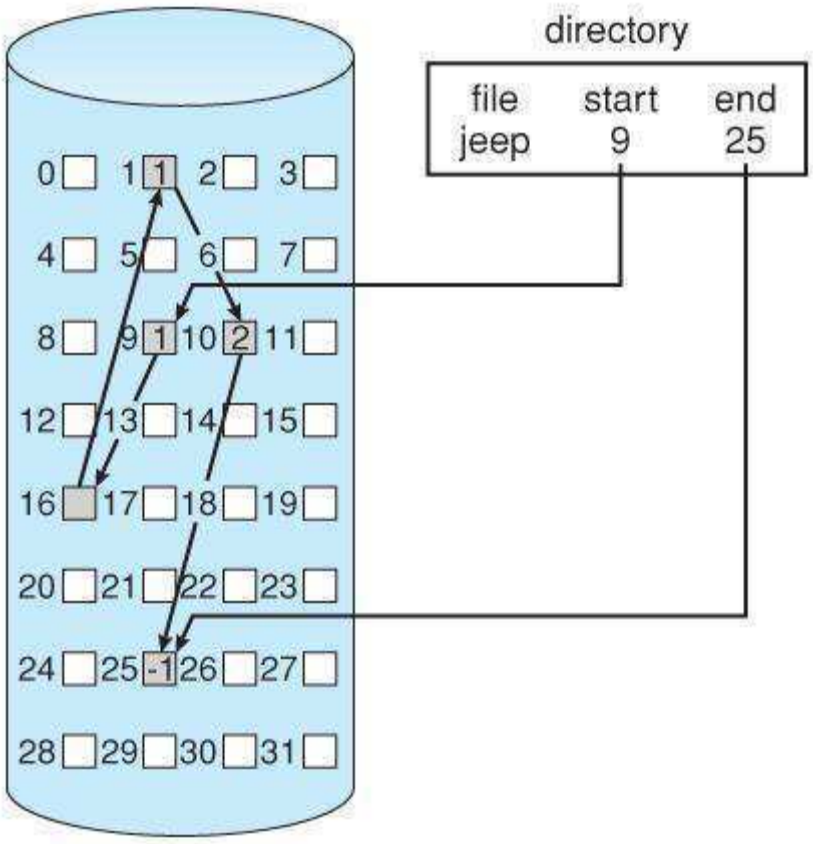


Fig 2.6 - Linked allocation of disk space.

- 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.

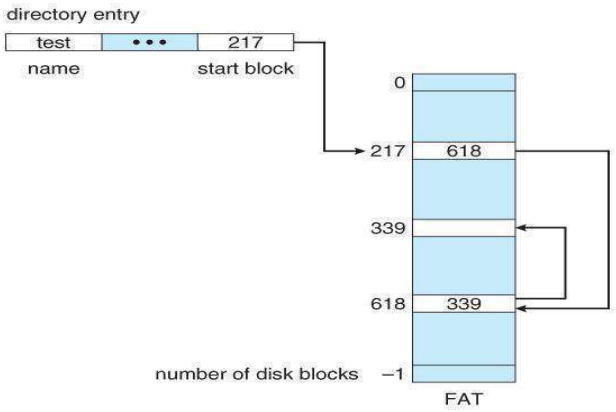


Fig 2.7 File-allocation table.

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 all over the disk or storing them in a FAT table.

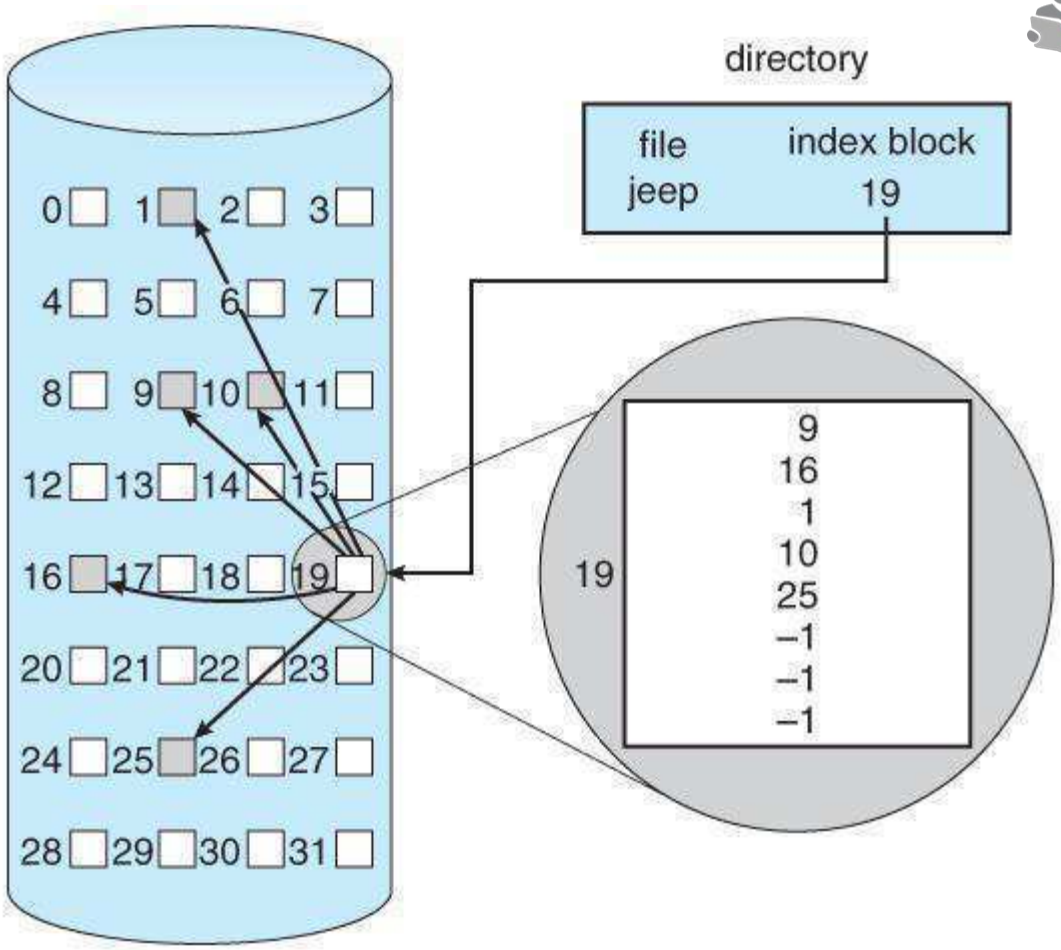


Fig 2.8 - Indexed allocation of disk space.

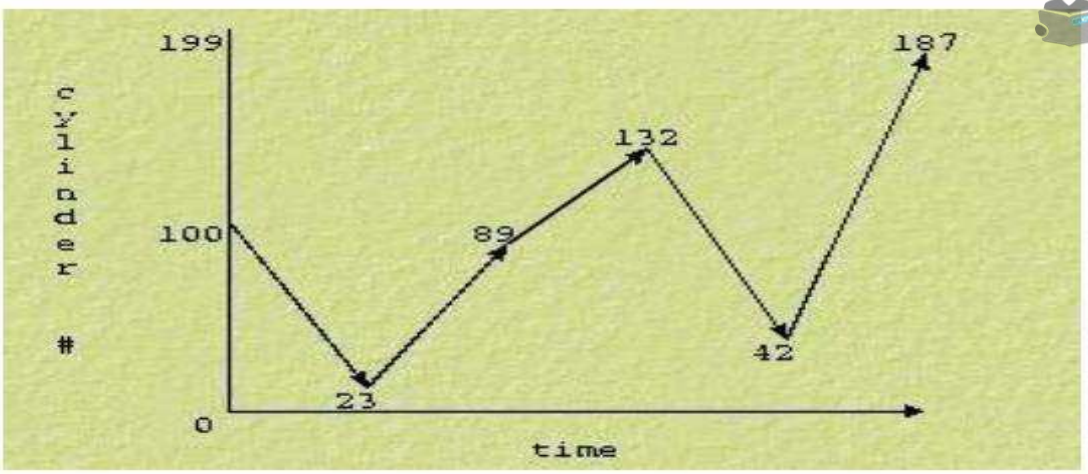
Disk Scheduling Algorithms

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



77+66+43+90+145=421

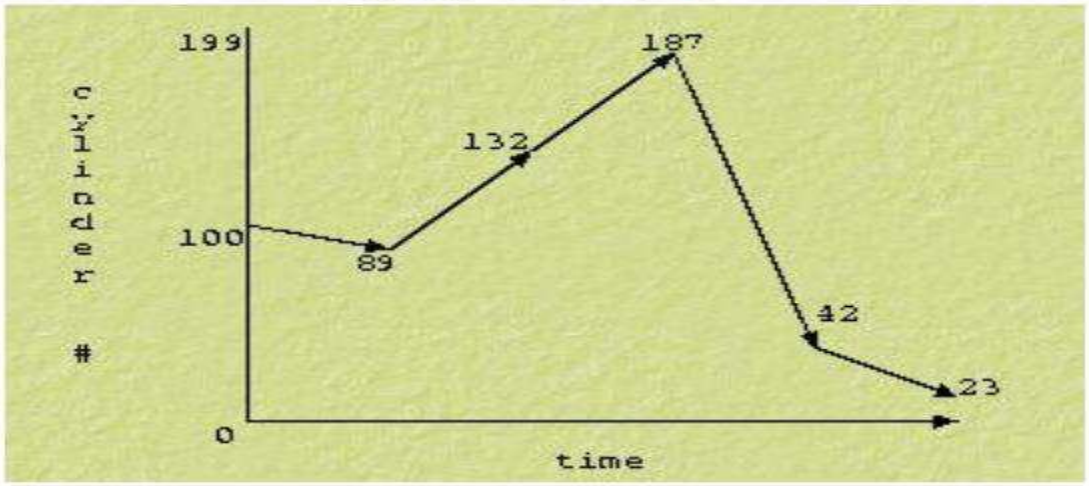
Fig 2.9 FCFS Scheduling

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.

23, 89, 132, 42, 187



11+43+55+145+19=273

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

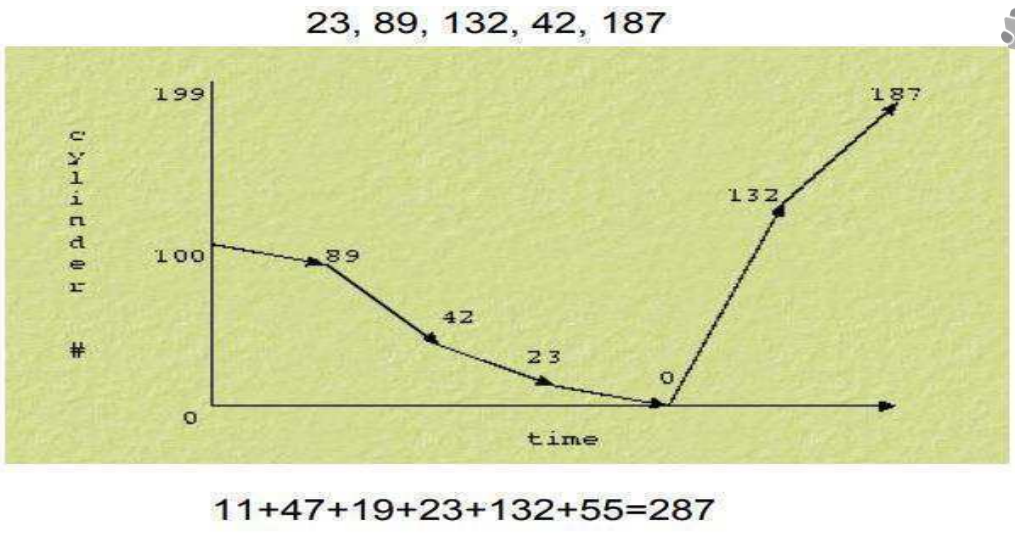


Fig 2.11 SCAN Scheduling

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.

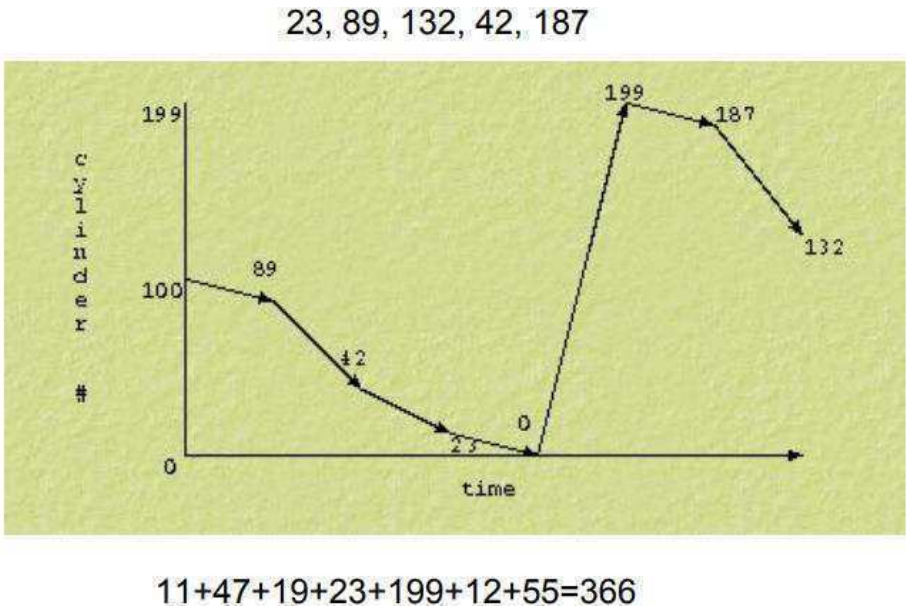


Fig 2.12 C-SCAN Scheduling

LOOK

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

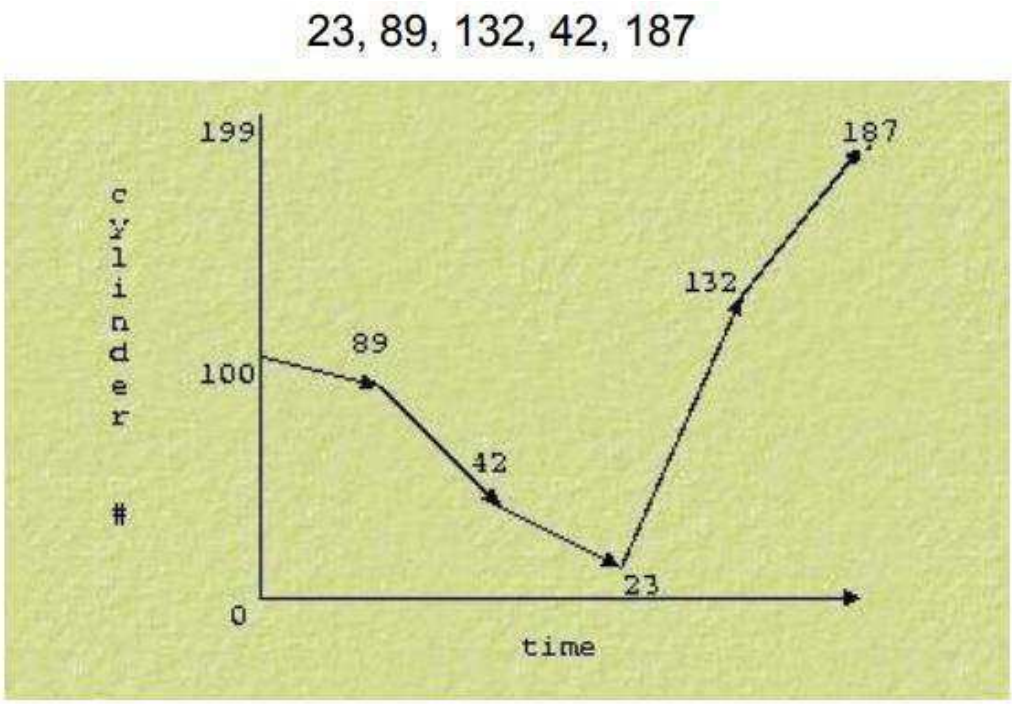


Fig 2.13 LOOK Scheduling

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, const char* newpath);
```

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in