Program : **B.Tech**

Subject Name: **Operating System**

Subject Code: **IT-501**

Semester: **5th**

Department of Information Technology
Class Notes

Unit 5
## Disk Management: Physical Disk Structure

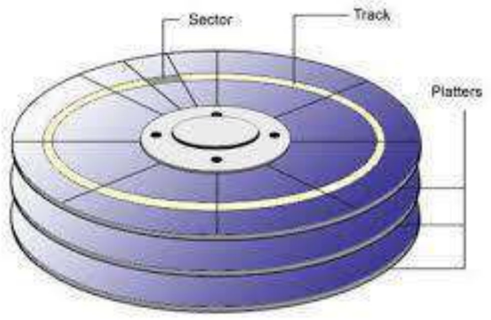


Figure 5.1 Disk Structure

### Disk Structure
- Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track (top platter) on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to inner most.

### Disk Access Time
Two major components
- Seek time is the time for the disk to move the heads to the cylinder containing the desired sector
  - Typically 5-10 milliseconds
- Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head
  - Typically, 2-4 milliseconds
- One minor component
  - Read/write time or transfer time– actual time to transfer a block, less than a millisecond

### Disk Scheduling
- Should ensure a fast access time and disk bandwidth
- Fast access
  - Minimize total seek time of a group of requests
  - If requests are for different cylinders, average rotation latency has to be incurred for each anyway, so minimizing it is not the primary goal (though some scheduling possible if multiple requests for same cylinder is there)
- Seek time =seek distance
- Main goal : reduce total seek distance for a group of requests
- Auxiliary goal: fairness in waiting times for the requests
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
- Several algorithms exist to schedule the servicing of disk I/O requests.

### TYPES OF DISK SCHEDULING ALGORITHMS
Although there are other algorithms that reduce the seek time of all requests, we will only concentrate on the following disk scheduling algorithms:

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

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

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

1. **First Come -First Serve (FCFS):** All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

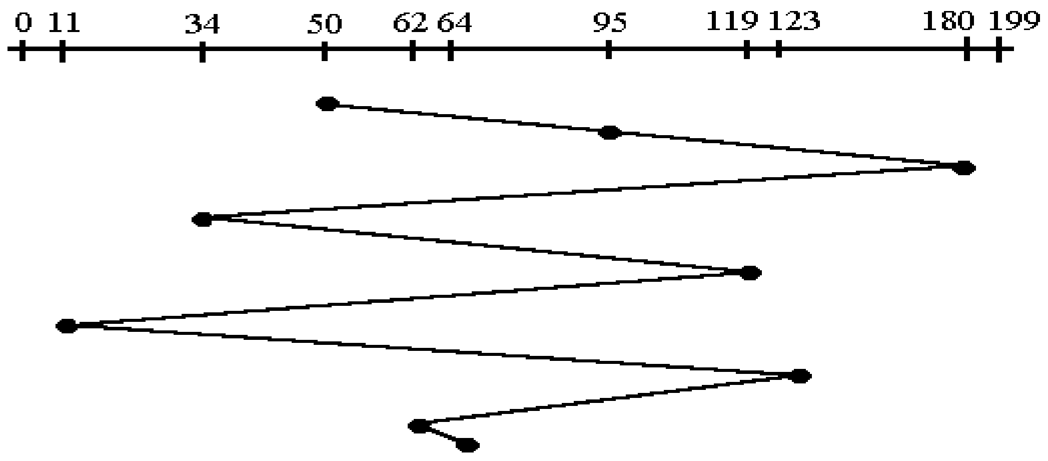


Figure 5.2 FCFS

**2. Shortest Seek Time First (SSTF)**

In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one.

There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.

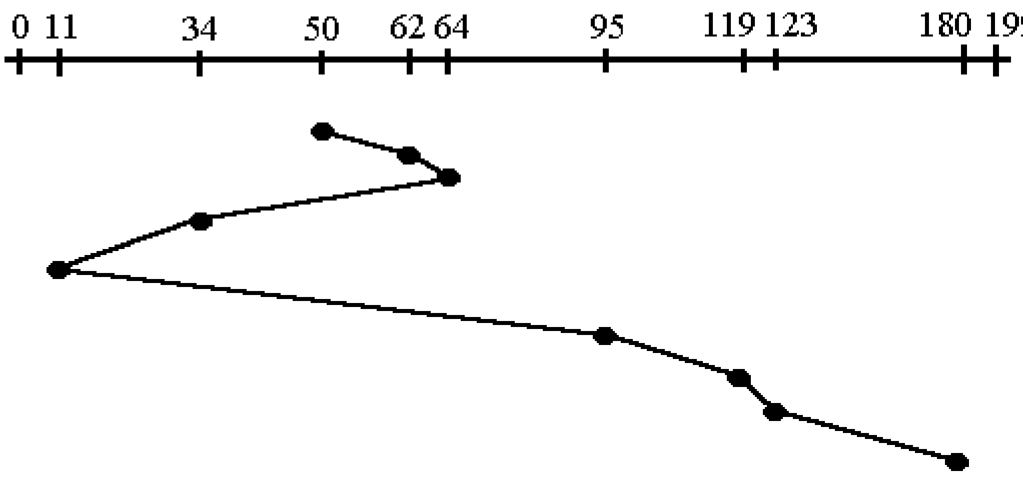


Figure 5.3 SSTF

3.Elevator (SCAN)

This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.

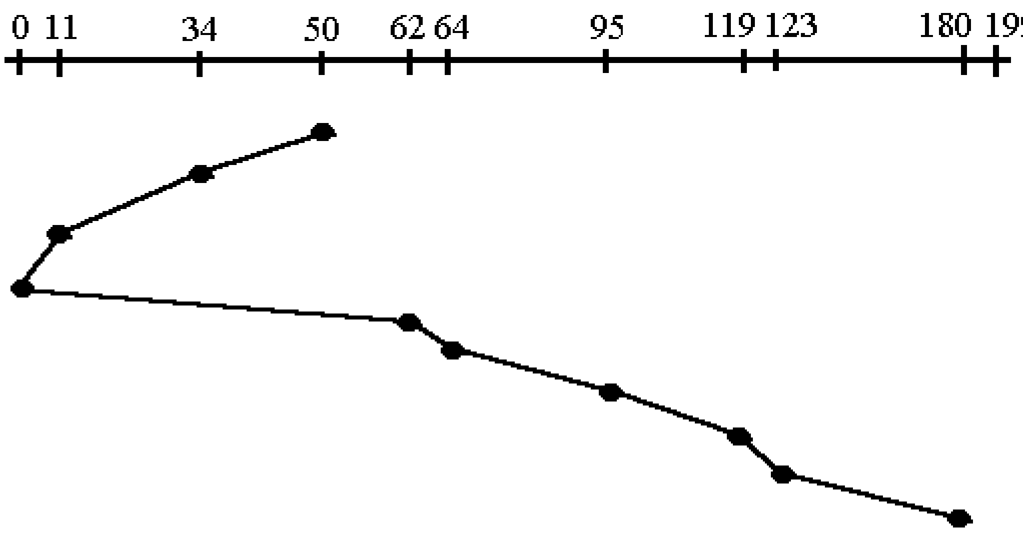


Figure 5.4 Scan

4. Circular Scan (C-SCAN)
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the most sufficient.

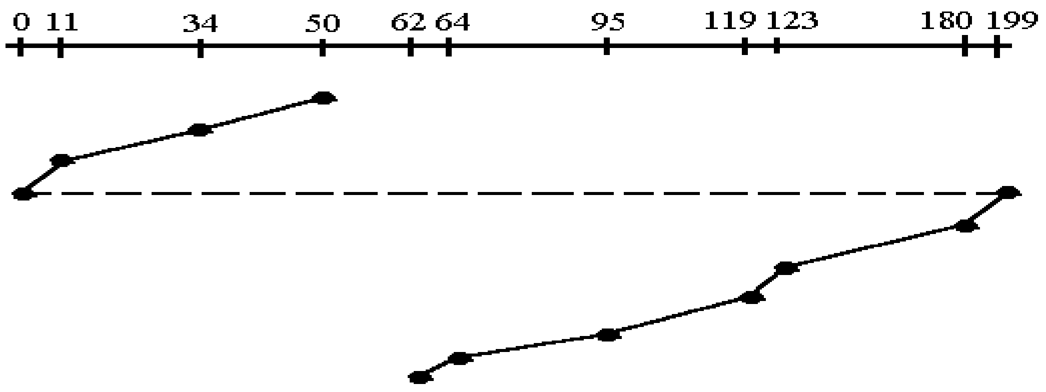


Figure 5.5 C - Scan

5. LOOK: It is similar to the SCAN disk scheduling algorithm except 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.

- Work Queue: 23, 89, 132, 42, 187
- there are 200 cylinders numbered from 0 - 199
- the diskhead stars at number 100

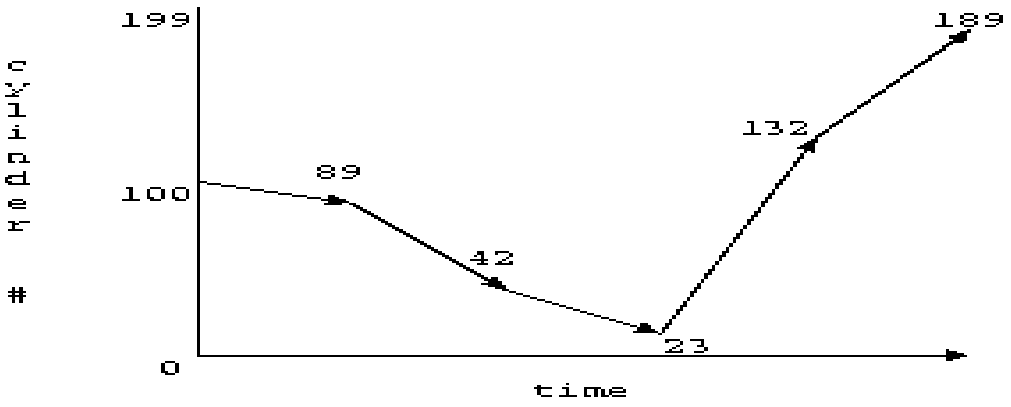


Figure 5.6 LOOK

6. C-LOOK

This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.
From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

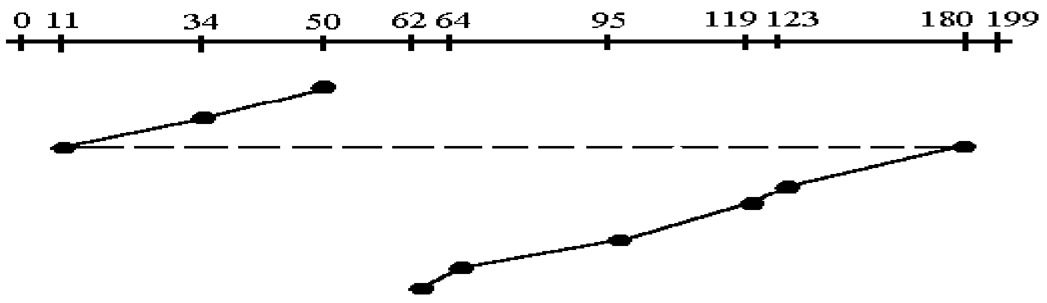


Figure 5.7 C-LOOK

**File System**
A file can be "free formed", indexed or structured collection of related bytes having meaning only to the one who created it. Or in other words an entry in a directory is the file. The file may have attributes like name, creator, date, type, permissions etc.

**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

A file has various kinds of structure. Some of them can be:

- Simple Record Structure with lines of fixed or variable lengths.
- Complex Structures like formatted document or re loadable load files.
- No Definite Structure like sequence of words and bytes etc.

**File and Directory concept:**
**FILE DIRECTORIES:**
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.
**Information contained in a device directory is:**

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed

- Date last updated
- Owner id
- Protection information

**Attributes of a File**

Following are some of the attributes of a file:

- **Name**: It is the only information which is in human-readable form.
- **Identifier**: The file is identified by a unique tag(number) within file system.
- **Type**: It is needed for systems that support different types of files.
- **Location:** Pointer to file location on device.
- **Size**: The current size of the file.
- **Protection:** This controls and assigns the power of reading, writing, executing.
- **Time, date, and user identification:** This is the data for protection, security, and usage monitoring.

**Operation on File**

There are many file operations that can be perform by the computer system. Here are the list of some common file operations:

- File Create operation
- File Delete operation
- File Open operation
- File Close operation
- File Read operation
- File Write operation
- File Append operation
- File Seek operation
- File Get attribute operation
- File Set attribute operation
- File Rename operation

Now let's describe briefly about all the above most common operations that can be performed with files.

**File Create Operation:**

- The file is created with no data.
- The file create operation is the first step of the file.
- Without creating any file, there is no any operation can be performed.

**File Delete Operation:**

- File must have to be deleted when it is no longer needed just to free up the disk space.
- The file delete operation is the last step of the file.
- After deleting the file, it doesn't exist.

**File Open Operation:** The process must open the file before using it.

**File Close Operation:** The file must be closed to free up the internal table space, when all the accesses are finished and the attributes and the disk addresses are no longer needed.

**File Read Operation:**

The file read operation is performed just to read the data that are stored in the required file.

**File Write Operation:**

The file write operation is used to write the data to the file, again, generally at the current position.

**File Append Operation:**

The file append operation is same as the file write operation except that the file append operation only add the data at the end of the file.

**File Seek Operation:**

For random access files, a method is needed just to specify from where to take the data. Therefore, the file seek operation performs this task.

**File Get Attribute Operation:**
The file get attributes operation are performed by the processes when they need to read the file attributes to do their required work.
**File Set Attribute Operation:**
The file set attribute operation used to set some of the attributes (user settable attributes) after the file has been created.
**File Rename Operation:**
The file rename operation is used to change the name of the existing file.

**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.

**File Access Methods**
The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods. some sre:
- Sequential access
- Direct/Random access
- Indexed sequential access

**Sequential Access**
- Data is accessed one record right after another is an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.

**Direct Access**
- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read / written;
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

**Indexed Sequential Access**
- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

**File 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.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

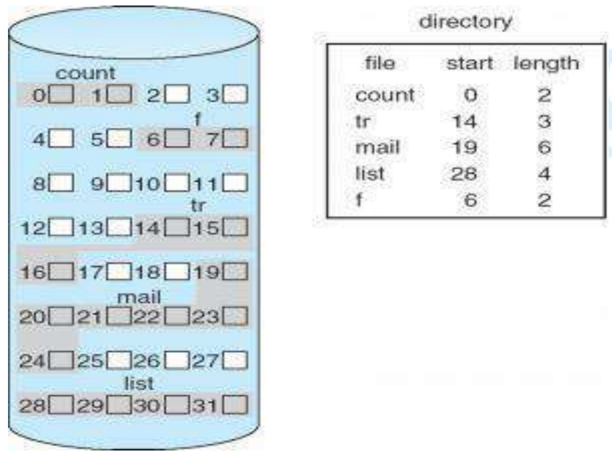
**1. Contiguous Allocation**

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.



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

Figure 5.8 contiguous allocation

**Advantages:**

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

**Disadvantages:**

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

**2. 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.

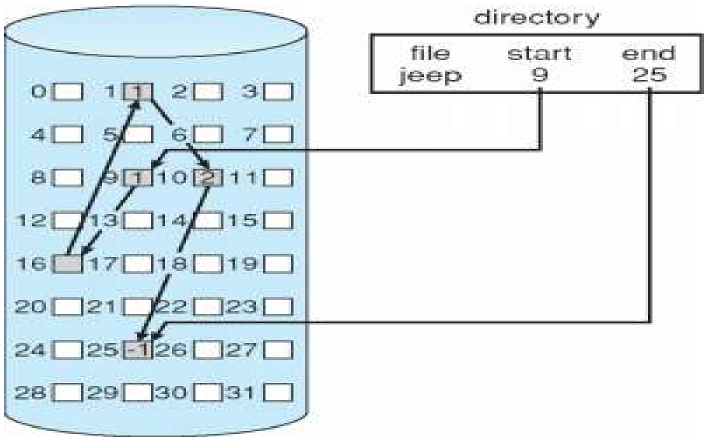


Figure 5.9 Linked List Allocations

**Advantages:**

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

**Disadvantages:**

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We cannot directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

**3. 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:

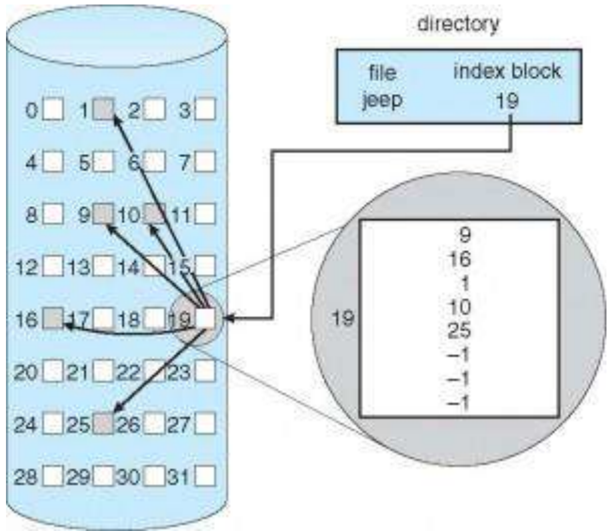


Figure 5.10 Indexed Allocations

**Advantages:**

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

**Disadvantages:**

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

**Free Space Management**

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks – those not allocated to some file or directory.
- To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file.
- This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list.

**1. Bit Vector**

- The free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be 001111001111110001100000011100000 …
- The main **advantage** of this approach is it's relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

**Example**

The Intel family starting with the 80386 and the Motorola family starting with the 68020 (processors that have powered PCs and Macintosh systems, respectively) have instructions that return the offset in a word of the first bit with the value 1. In fact, the Apple Macintosh operating system uses the bit-vector method to allocate disk space.

The calculation of the block number is

**(Number of bits per word) x (number of 0-value words) + offset of first 1 bit.**

Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory (and is written to disk occasionally for recovery needs). Keeping it in main memory is possible for smaller disks, such as on microcomputers, but not for larger ones.
A 1.3-GB disk with 512-byte blocks would need a bitmap of over 332 KB to track its free blocks. Clustering the blocks in groups of four reduces this number to over 83 KB per disk.

**2. Linked List**

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.
- In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.
- The FAT method incorporates free-block accounting data structure. No separate method is needed.

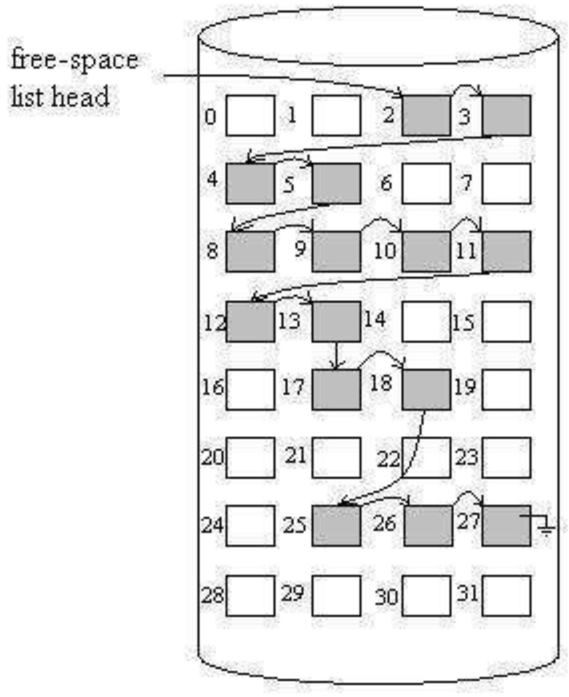


Figure 5.11 Link list space managements

**3. Grouping**

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- The first n-1 of these blocks is actually free.
- The last block contains the addresses of another n free block, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

**4. Counting**

- We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than one.

**Directory**

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory:

- **Name**: The name visible to user.
- **Type**: Type of the directory.
- **Location**: Device and location on the device where the file header is located.
- **Size**: Number of bytes/words/blocks in the file.
- **Position**: Current next-read/next-write pointers.
- **Protection**: Access control on read/write/execute/delete.
- **Usage**: Time of creation, access, modification etc.
- **Mounting**: When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.

**Operations performed on directory are:**

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

**Advantages of maintaining directories are:**

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

**Directory Structure**

**SINGLE-LEVEL DIRECTORY**

In this a single directory is maintained for all the users.

- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.

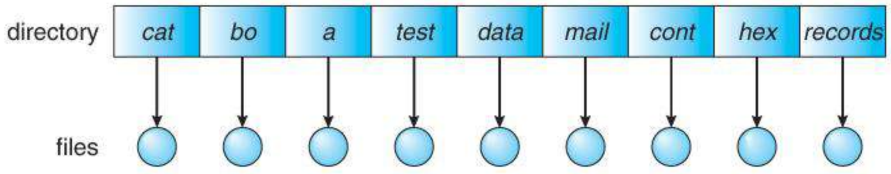


Figure 5.12 SINGLE-LEVEL DIRECTORY

**Two-Level Directory**

- Each user gets their own directory space
- File names only need to be unique within a given user's directory.
- A master file directory is used to keep track of each user’s directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system (executable) files.
- Systems may or may not allow users to access other directories besides their own
    - If access to other directories is allowed, then provision must be made to specify the directory being accessed.
    - If access is denied, then special consideration must be made for users to run programs located in system directories. A search path is the list of directories in which to search for executable programs, and can be set uniquely for each user.

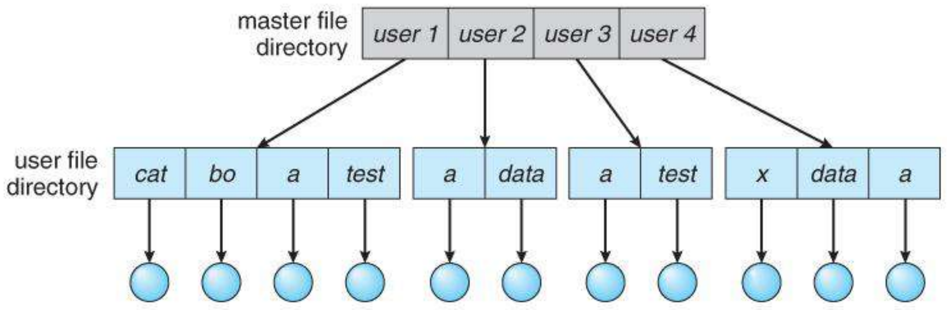


Figure 5.13 Two-Level Directory

**Tree-Structured Directories**

- An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar.
- Each user / process has the concept of a **current directory** from which all (relative) searches take place.
- Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory. )
- Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.
- One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

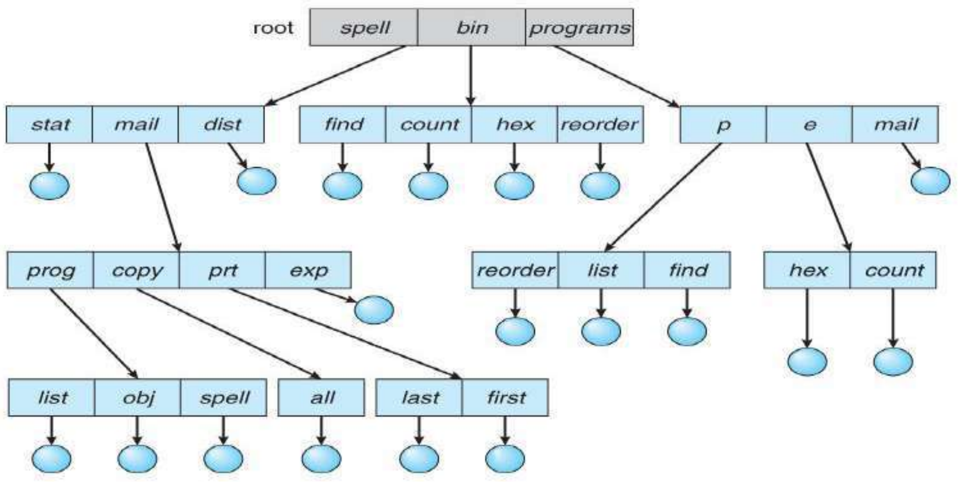


Figure 5.14 Tree-Structured Directories

Protection:

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List
- Access Lists and Groups
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux: RWX
  - owner access 7 $\Rightarrow$ 1 1 1
  - group access 6 $\Rightarrow$ 1 1 0
  - public access 1 $\Rightarrow$ 0 0 1

**File Sharing Implement Issue:** Sharing of files on multi-user systems is desirable

- Sharing may be done through a protection scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system

- User IDs identify users, allowing permissions and protections to be peruser
- Group IDs allow users to be in groups, permitting group access rights
- Owner of a file / directory
- Group of a file / directory

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using distributed file systems
  - Semi automatically via the world wide web
- Client-server model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - NFS is standard UNIX client-server file sharing protocol
  - CIFS is standard Windows protocol
  - Standard operating system file calls are translated into remote calls

File Sharing – Failure Modes

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called metadata
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

**Linux File System**

A file is a named collection of related information that is a 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 .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 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.

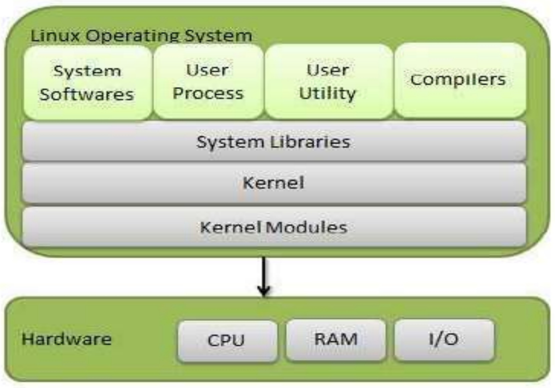


Figure 5.15 Component of Linux

**Kernel Mode vs. User Mode**

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each process and provides system services to processes, provides protected access to hardware to processes.

Support code, which is not required to run in kernel mode, is in System Library. User programs and other system programs works in User Mode, which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low-level tasks.

**Basic Features**

Following are some of the important features of Linux Operating System.

**Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.

**Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

**Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

**Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.

**Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.

**Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It used to do various types of operations, call application programs. Etc.

**Security** − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

**Architecture**

The following illustration shows the architecture of a Linux system −

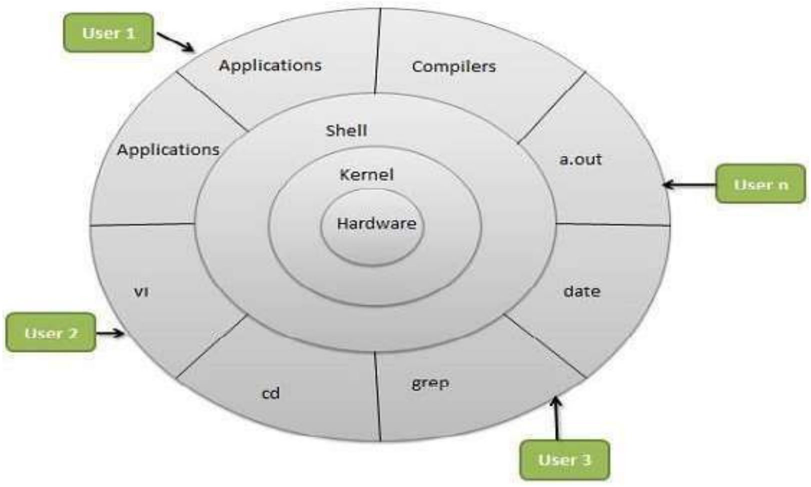


Figure 5.16 Architecture of Linux System

The architecture of a Linux System consists of the following layers −

**Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).

**Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.

**Shell** −An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.

**Utilities** – Utility programs that provide the user most of the functionalities of an operating systems

**FAT**
A file allocation table (FAT) is a table that an operating system maintains on a hard disk that provides a map of the clusters (the basic units of logical storage on a hard disk) that a file has been stored in. When you write a new file to a hard disk, the file is stored in one or more clusters that are not necessarily next to each other; they may be rather widely scattered over the disk. A typical cluster size is 2,048 bytes, 4,096 bytes, or 8,192 bytes. The operating system creates a FAT entry for the new file that records where each cluster is located and their sequential order. When you read a file, the operating system reassembles the file from clusters and places it as an entire file where you want to read it. For example, if this is a long Web page, it may very well be stored on more than one cluster on your hard disk.
A file allocation table (FAT) is a file system developed for hard drives that originally used 12 or 16 bits for each cluster entry into the file allocation table. It is used by the operating system (OS) to manage files on hard drives and other computer systems. It is often also found on in flash memory, digital cameras and portable devices. It is used to store file information and extend the life of a hard drive.
Most hard drives require a process known as seeking; this is the actual physical searching and positioning of the read/write head of the drive. The FAT file system was designed to reduce the amount of seeking and thus minimize the wear and tear on the hard disc.
FAT was designed to support hard drives and subdirectories. The earlier FAT12 had a cluster addresses to 12-bit values with up to 4078 clusters; it allowed up to 4084 clusters with UNIX. The more efficient FAT16 increased to 16-bit cluster address allowing up to 65,517 clusters per volume, 512-byte clusters with 32MB of space, and had a larger file system; with the four sectors it was 2,048 bytes.
FAT16 was introduced in 1983 by IBM with the simultaneous releases of IBM's personal computer AT (PC AT) and Microsoft's MS-DOS (disk operating system) 3.0 software. In 1987 Compaq DOS 3.31 released an expansion of the original FAT16 and increased the disc sector count to 32 bits. Because the disc was designed for a 16-bit assembly language, the whole disc had to be altered to use 32-bit sector numbers.
In 1997 Microsoft introduced FAT32. This FAT file system increased size limits and allowed DOS real mode code to handle the format. FAT32 has a 32-bit cluster address with 28 bits used to hold the cluster number for up to approximately 268 million clusters. The highest level division of a file system is a partition. The partition is divided into volumes or logical drives. Each logical drive is assigned a letter such as C, D or E.

**A FAT file system has four different sections, each as a structure in the FAT partition.**
**Boot Sector:** This is also known as the reserved sector; it is located on the first part of the disc. It contains: the OS's necessary boot loader code to start a PC system, the partition table known as the master boot record (MRB) that describes how the drive is organized, and the BIOS parameter block (BPB) which describes the physical outline of the data storage volume.
**FAT Region:** This region generally encompasses two copies of the File Allocation Table which is for redundancy checking and specifies how the clusters are assigned.
**Data Region:** This is where the directory data and existing files are stored. It uses up the majority of the partition.
**Root Directory Region**: This region is a directory table that contains the information about the directories and files. It is used with FAT16 and FAT12 but not with other FAT file systems. It has a fixed maximum size that is configured when created. FAT32 usually stores the root directory in the data region so it can be expanded if needed.

**Introduction to Distributed System:**

- A distributed operating system is an operating system that runs on several machines whose purpose is to provide a useful set of services, generally to make the collection of machines behave more like a single machine. The distributed operating system plays the same role in making the collective resources of the machines more usable that a typical single-machine operating system plays in making that machine's resources more usable. Usually, the machines controlled by a distributed operating system are connected by a relatively high quality network, such as a high speed local area network. Most commonly, the participating nodes of the system are in a relatively small geographical area, something between an office and a campus.

- Distributed operating systems typically run cooperatively on all machines whose resources they control. These machines might be capable of independent operation, or they might be usable merely as resources in the distributed system. In some architectures, each machine is an equally powerful peer as all the others. In other architectures, some machines are permanently designated as master or are given control of particular resources. In yet others, elections or other selection mechanisms are used to designate some machines as having special roles, often controlling roles.
- Sometimes distinctions are made between parallel operating systems, distributed operating systems, and network operating systems, though the latter term is now a bit archaic. The distinctions are perhaps arbitrary, though they do point out differences in the design space for making operating systems control operations across multiple processing engines.
    - A parallel operating system is usually defined as running on specially designed parallel processing hardware. It usually works on the assumption that elements of the hardware (such as the memory) are tightly coupled. Often, the machine is expected to be devoted to running a single task at very high speed.
    - A distributed operating system is usually defined as runing on more loosely coupled hardware. Unlike parallel operating systems, distributed operating systems are intended to make a collection of resources on multiple machines usable by a set of loosely cooperating users running independent tasks.
    - Network operating systems are sometimes regarded as systems that attempt merely to make the network connecting the machines more usable, without regard for some of the larger problems of building effective distributed systems.