V 16BIA05A1 CSE-II RGAN U.S N.V. Srikanth Reddy

# Unit 5
# File System

## 1. File Concept:

Computer stores information on various storage media like magnetic disks, tapes and optical disks. Files are mapped by OS onto physical devices. File is named collection of related information that is recorded or stored on secondary storage. Commonly, files represents programs and data, where Data may be numeric, alphabetic or binary. Many different types of information may be stored in file: source, object, executable programs and so on

- Source file is sequence of sub-routines and functions each of which further organized as declarations followed by statements
- Object file is sequence of bytes organized into blocks understandable by system linker
- Executable file is series of code sections that loader can bring into memory and execute

### 1.1 File Attributes:

File attributes vary from one OS to another but typically consist of these:

- **Name** – symbolic file name is only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system, non human readable name
- **Type** – needed for systems that support different types of files
- **Location** – pointer to device and to the location of file on that device
- **Size** – current file size
- **Protection** – Access Control information that define who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring

### 1.2 File Operations:

- File is an **abstract data type**, operation that can be performed on files must be known to define file properly, these operations are provided by OS
- **Operations are:**
  - **Creating file**: first space in file system must be found for file and then entry for new file must be made in directory to create file
  - **Writing file**: make a system call specifying both name of file and information to be written, with given name system searches directory to find location and keeps write pointer to write into file
  - **Reading file**: make as system call that specifies name of file , with given name system reaches directory to find location and keeps read pointer to read contents from file
  - **Repositioning within file**: directory is searched for appropriate entry and current file position pointer is re-positioned to given value which is known as Seek
  - **Deleting file**: search directory for named file, releases all file space so that it can be reused by other files and erase directory entry

- **Truncating file**:user may want to erase contents of file but keep its attributes rather delete, this option lets file be resets to length zero and its file space released

- Other common operations:
  - Appending new information at the end of file
  - Renaming existing file
  - Copying file to another I/O device
  - Get & Set various attributes of file
- Some OS provide facilities for locking files, useful for files shared by several process
- File locks allows one process to lock file & prevent other process form granting access to it
- File locks provide functionality:
  - **Shared lock**: is reader lock in that several processes can acquire lock concurrently
  - **Exclusive lock**: is writer lock , only one process at a time can acquire lock
- OS may provide:
  - **Mandatory**: access is denied depending on locks held and requested
  - **Advisory** :processes can find status of locks and decide what to do

### 1.3 File Types:

- OS should recognize and support file types, then only it can operate on file in reasonable way.
- Common technique used is to include type as part of file name
- Name is split into two parts; **Name & Extension** separated by **dot**
- MS DOS recognizes only few extensions
- Application programs also use extensions to indicate file types: *.doc*

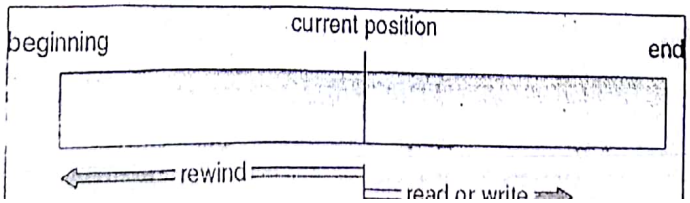
| file type | usual extension | function |
|---|---|---|
| executable | exe, com, bin or none | ready-to-run machine-language program |
| object | obj, o | compiled, machine language, not linked |
| source code | c, cc, java, pas, asm, a | source code in various languages |
| batch | bat, sh | commands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

## 2. Access Methods

### 2.1 Sequential Access:

Simplest access method, file is accessed in order one record after other. Reads & Writes make up bulk of operations on file.

- **Read Next**: reads next portion of file, automatically advances file pointer
- **Write Next**: appends to end of file & advances to end of newly written data
- Tape model of file works as sequential access



delete,

## 2 Direct/ Relative Access:

ile is made up of **fixed length logical records** that allows to read or write records rapidly in no particular order

It is based on disk model of file, since disk allow random access to any file. Direct access files are used for immediate access of large amount information. Databases are often of this type, answer may contains in block of given query.

Ex: Airline reservation system, might store all info. of flight in particular block

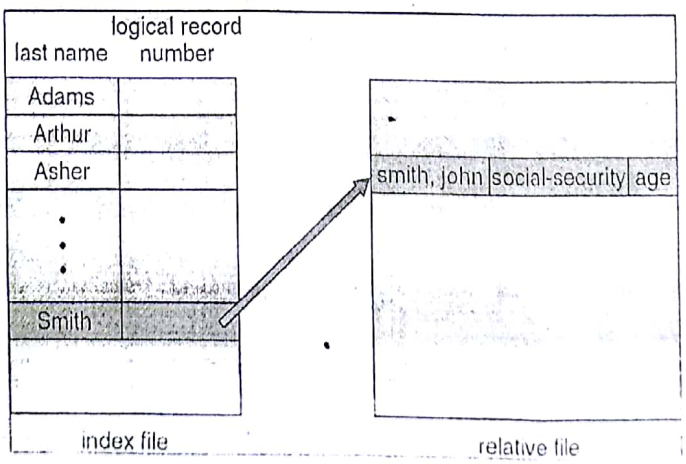
File operations includes block number N: **read N, write N**

| sequential access | implementation for direct access |
|---|---|
| *reset* | *cp* = 0; |
| *read next* | *read cp*;<br>*cp* = *cp* + 1; |
| *write next* | *write cp*;<br>*cp* = *cp* + 1; |

## 2.3 Other Access Methods:

Built on top of Direct access method, they involve contraction of index for file. **Index** contains pointer to various blocks in file like index in back of book. To find record in file, First search in index & then use pointer to access file directly

- Ex: retail price file might list Universal Product Codes (UPC) for items, with prices
- With large files, index file itself may become too large to kept in memory
  - One solution is to create index for index file
  - Primary index file will contain pointer to secondary index file, which points to actual data item
- Ex: IBM's indexed sequential access method (ISAM) uses small master index that points to disk blocks of secondary index
- Secondary index blocks point to actual file blocks
- Binary search is used to find block containing desired record
- Finally block is searched sequentially



## 3. Directory Structure:

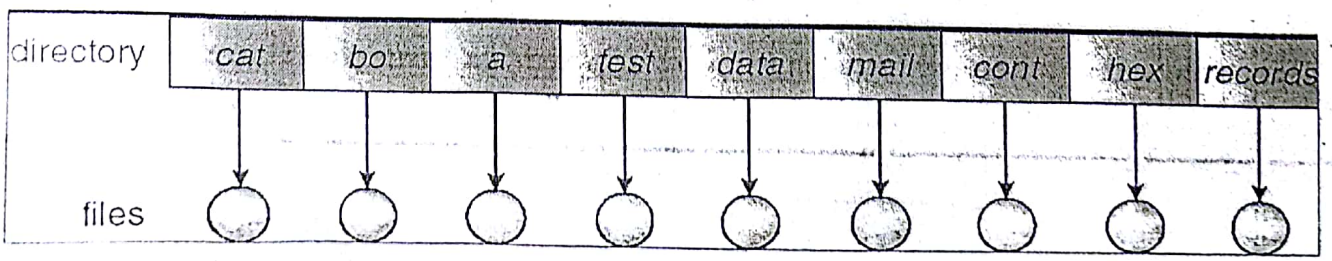
- Systems may have zero or more file systems & may be of varying types
- Ex: Solaries have few UFS file system, VFS file system, NFS file systems
- File system can be extensive, some systems store millions of files on tera bytes
- Organization of these data involves use of directories

### 3.1 Directory Overview:

- It can be viewed as symbol table that translates file names into their directory entries
- Directory itself can be organized in many ways: insert, delete entries, search form named entry, list all entries etc.
- Operations that are to be performed on directory:
  - **Search for a file**: search directory to find entry for particular file, files have symbolic names & similar names may indicate relationship between files
  - **Create a file**: new files need to create & add to directory
  - **Delete a file**: when file is no longer needed, remove it from directory
  - **List a directory**: able to list files in directory & contents of directory entry
  - **Rename a file**: file name represents its contents, able to change name when contents or use of file changes, it may allow to change its position in directory
  - **Traverse the file system**: for reliability, save contents & structure of entire file system at regular intervals, if file is no longer in use file can be copied to tape & disk space of that file released for reuse by another file .

### 1. Single Level Directory:

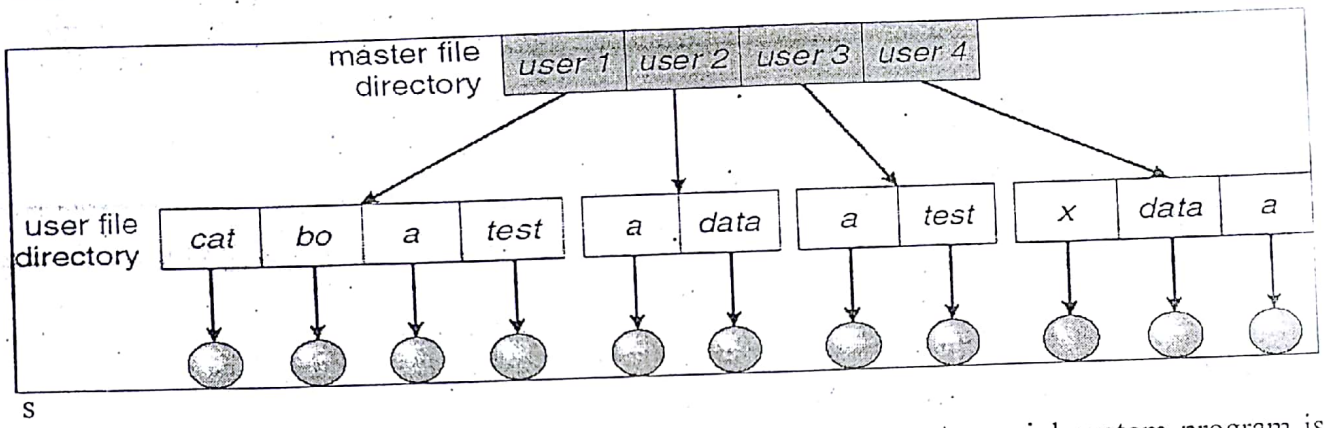
- All files are contained in same directory, which is easy to support & understand
- When number of files increases or when system has more than one user, it has significant limitations
- Since all files are in same directory, they must have unique names
- Length of file name is also limited
  - Ex: MS-DOS allows only 11 characters, UNIX allows 255 characters
- Single user on single level directory may find difficult to remember names of all files as number of files increases



## 2. Two Level Directory:

- Single level directory often leads to confusion of file names among different users.
- One solution is to create **Separate directory** for each user
- In two level directory structure

  Each user has his own **User File Directory (UFD).** When user job starts or user logs in, systems **Master File Directory (MFD)** is searched. MFD is indexed by user name or account number & each entry points to UFD of user. When user refers to particular file, only his own UFD is searched. Different users may have files with same name. To create or delete file OS searches in users UFD




s

er directories themselves must be created or deleted as necessary. A special system program is run with appropriate user name & account information. It creates new UFD & adds entry for it to MFD

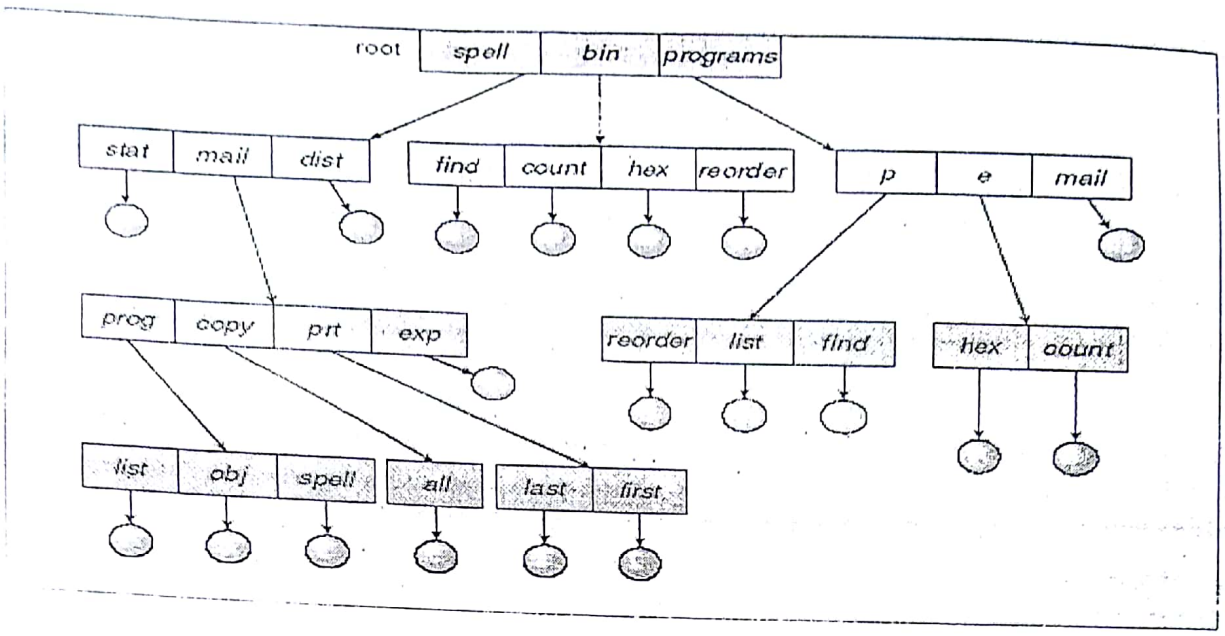
- Although two level directory solves name collision problem, it has some drawbacks
- **Advantages & Disadvantages:**
  - Isolation:
    - This structure effectively isolates one user from another
    - It is useful when users are completely independent
    - When users want to cooperate on some task by sharing files, it do not allows
    - If access it to be permitted, one user must have ability to name file in another users directory, we must give both user name & file name
- Two level directory can be thought of tree of height 2, root is MFD & Childs are UFD's
- Childs of UFD's are files, files are leaves of tree.

## 3. Tree Structured Directories:

Two level directory structure can be extended to tree of arbitrary height. This generalization allows users to create their own sub directories & to organize their files accordingly. Tree has root directory, every file in system has unique path name. Directory contains set of file or subdirectories, directory is simply another file but is treated in special way, All directories has same internal format

One bit in each directory entry define entry as **file (0) or sub directory (1).** special system call are used to create & delete directories Each process has **Current Directory**, contains most of files that are of

current interest When reference is made, it will be searched in current directory first If is not found, then user must either specify path name or change current directory.



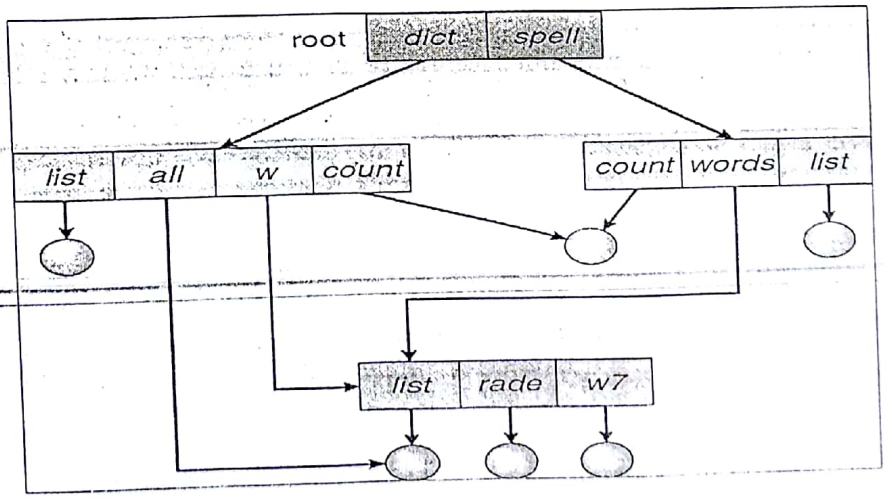

- Path names can be of tow types:
  - Absolute Path name:
  - Relative path name
- Difficulty in tree structured directory is **how to handle deletion of directory.** If directory is empty, its entry in directory that contains it can simply deleted. If directory is not empty then:
  - Do not delete until directory is empty Ex: MS-DOS
  - Provide option to delete all files & sub directories to be deleted or not Ex: UNIX
- With tree structure, users can be allowed to access in addition to their files of other users, path in tree structure is longer than two lever structure

## 4. Acyclic-Graph Directories:

- Consider two programmers who are working on joint project
  - Files associated with that project can be stored in sub directory, separating them from other projects & files of two programmers
  - Since both programmers are equally responsible for project, both want sub directory to be in their own directories
  - Common sub directory should be shared, Shared directory or file will exist in file system in two places at once
- A tree structure prohibits sharing of files or directories.
- An Acyclic Graph is a graph with no cycles allows directories to share sub directories & files. Acyclic graph is natural generalization of tree structured directory scheme
- With two copies, each programmer can view copy rather than original

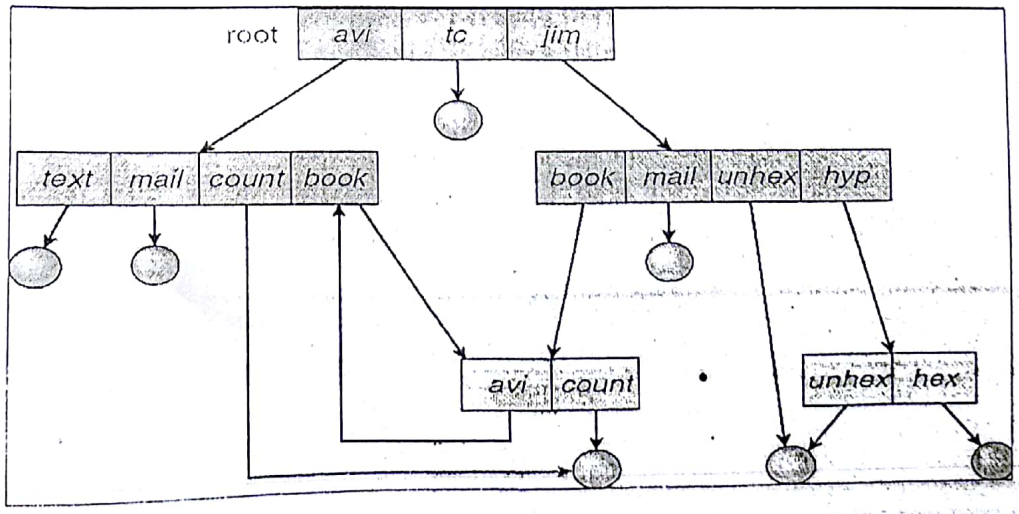
- If one programmer changes file, changes will not appear in others copy
- With shared files only one actual file exists, so any changes made by one person are immediately visible to other

- Even in single user, users file organization may require that some file be placed in different sub directories, Ex: program written for particular project should both in directory of all programs
- Simple ways is:
  - In UNIX common way to create new directory entry called Link, link is effectively pointer to another file or sub directory Ex: link may be absolute or relative path name
  - If directory entry is marked as link, then name of real file is included in link information
  - We resolve link by using that path name to locate real file
  - Links are easily identified by their format in directory entry & effectively named indirect pointers
- Another ways is:
  - To implement shared files is simply to duplicate all information about them in both sharing directories, both entries are identical & equal
  - Duplicate directory entries, make original & copy indistinguishable
  - Major problem is maintaining consistency when file is modified.
- Acyclic graph structure is more flexible than simple tree structure, but is more complex
  - A file may now have multiple absolute path names, distinct file names may refer to same file
  - 
  - When can space allocated to shared file be de-allocated & reused?
  - If file entry itself is deleted, space for file is de-allocated, leaving links dangling



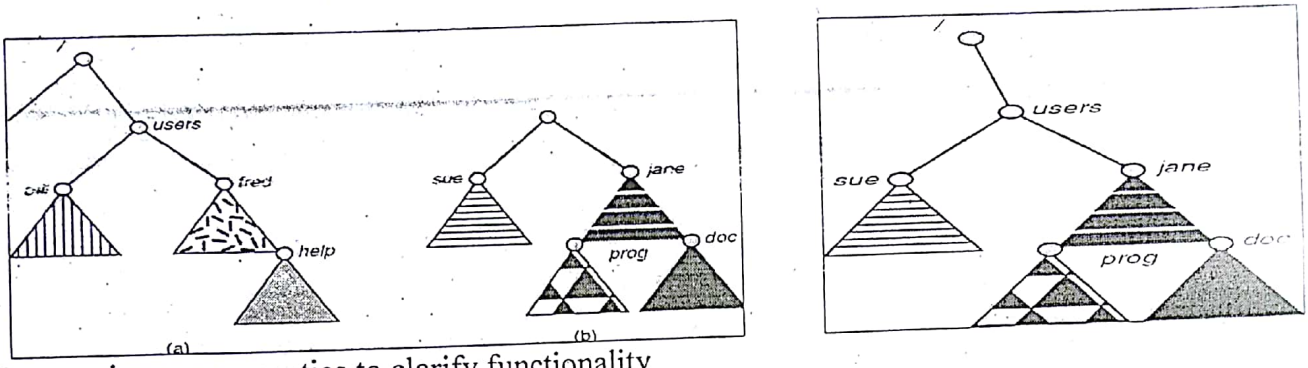
## 5. General Graph Directory

- Serious problem with acyclic graph structure is ensuring that there are no cycles. If we start with two level directory & allow users to create sub directories, a tree structured directory results. When we add links to existing tree structured directory, tree structure is destroyed resulting in simple graph structure
- Primary advantages of acyclic graph is Relative simplicity of algorithms to traverse graph & determine when there are no more references to file. If cycles are allowed to exist in directory, we likewise want to avoid searching any component twice, for reasons of correctness as well as performance
- Poorly designed algorithm might results in infinite loop continually searching through cycle & never terminating.
- one solution to limit arbitrarily number of directories that will be accessed during search Similar problem exists when we are trying to determine when file can be deleted. With acyclic graph directory structures, value of 0 in reference count means that there are no more references to file or directory & file can be deleted.
- When cycle exist, reference count may not be 0 even when it is no longer possible to refer to director or file
- This anomaly results from possibility of self referencing in directory structure
- In this case, we need to use garbage collection scheme to determine when last reference has been deleted & disk space can be reallocated. Garbage collection involves traversing entire file system making everything that can be accessed. Garbage collection for disk based file system is extremely time consuming
- Difficulty is to avoid cycles as new links are added to structure, how do we known when new link will complete cycle?
- There are algorithms to detect cycles in graphs, are computationally expensive when graph is on disk storage
- Simpler algorithm in special case of directories & links is to bypass links during directory traversal



## File System Mounting:

- A file-system must be **mounted** before it can be accessed
- OS is given name of device and **Mount Point** – location within file structure where file system is to be attached, Typically mount point is an empty directory EX: in UNIX
- Next OS verifies that device contains valid file system by asking device driver to read device directory and verify the expected format
- Finally, OS notes in its directory structure that file system is mounted at the specified mount point
- It will enables OS to traverse its directory structure
- Consider the example:




- System impose semantics to clarify functionality
  - Ex: system may disallow mount over directory that contains files

## 5. File Sharing:

User oriented OS must accommodate need to share files. Once multiple users are allowed to share file. The challenge is to extend sharing to multiple file systems including remote

### Multiple Users:

- When OS accommodate multiple users, following issues are permitted
  - File sharing, File naming, File protection
- System must mediate file sharing, if we allow files to be shared by users
- For **Sharing & Protection**, it must maintain more file & directory attributes
- Most systems evolved to use concept of file **Owner & Group**
  - Owner can change attributes, grant access & have most control over it
  - Group defines sub set of users who can share access of file
- Owner & Group ID of given file are stored with other file attributes
- If a user requests file, User ID is checked with Owner attribute followed by Group attribute to grant permissions

### Remote File Systems:

- Networking allows sharing of resources spread over network
- Remote file sharing methods are changed with evolution of network:
  - **Manual transfer**: of file between machines via programs like FTP

- **Distributed File System (DFS)**: remote directories are visible from local system
- World Wide Web: browser is needed to gain access to remote files

- FTP is used for both **Anonymous & Authenticated** access
- Anonymous access allows user to transfer files without having account on remote system Ex: WWW
- DFS involves much tighter integration between local & remote machines

Client Server Model:

- Remote file system allows computer to mount one or more file systems from one or more remote machines
- Machine containing file is **Server**, Machine seeking access is **Client**
- Server usually specifies available files on volume or directory
- Client can be specified by Network name or other identifier like IP address
- **Spoofing** is done, unauthorized client could be allowed to access server
- Secured encrypted keys are used to provide more security
- In Unix, Network File System (NFS) takes authentication via client networking
- In this user ID's of client & server should match other wise access denied
- Once remote file system is mounted, file access is allowed on behalf of user
- Server then apply standard access checks to determine user credential access
- If it is allowed, file handle will be returned to client

**Distributed Information Systems:**

- Known as Distributed Naming Services(DNS), provide unified access to info.
- It provides **Host name to Network address** translations for entire internet
- It provide user name, password, user ID, group ID for distributed facility
- Sun micro systems introduced yellow pages known as **Network Info. Service**
  - It centralizes storage of user, host names, printer information etc.
  - It uses unsecure authentication methods including pass words identify by host IP
  - NIS+ is much more secure replacement for NIS
- One distributed LADP directory is used by organization to store all user & resource information of other computers to result **Secure single sign in**

Failure Modes:

- Local file system can fail for variety of reasons
  - Failure of disk containing file system
  - Corruption of directory structure or disk management info. (meta data)
  - Disk, Cable, Host adapter failure
- Remote file system have even more failure modes
  - Complexity of network, more problems with interface interactions

- Interrupts between hosts cause hardware failure, network implementation issues

**onsistency Semantics:**

- Consistency Semantics are
  - important criterion for evaluating any file system that supports file sharing
  - Used to specify how multiple users of system are to access shared file simultaneously
  - Typically implemented as code with file system
  - Directly related to Process Synchronization algorithms
- EX: performing atomic transaction to remote disk involve several network communications, disk reads & writes or both.

## 6. Protection:

- When information is stored in system, keep it safe from:
  - Physical Damage (Reliability)
  - Improper Access (Protection)
- Reliability is provided by duplicate copy of files
  - Many systems have programs that automatically copy disk files to tape
  - File System can be damaged by hardware problems, power failures, crashes, dirt, temperature extremes etc.
  - Files may be deleted accidentally
  - Bugs in file systems can also cause file content to lost
- Protection can be provided in many ways:
  - In Single user system, protect by physical remove of disk & locking
  - In Multi user system, other mechanisms are needed

**Types of Access:**

- Systems that do not permit access to files of other users do not need protection
- Complete protection is provided by Prohibiting access with **Controlled access**
- In Controlled access, access is permitted or denied depending on several factors one of which is type **of access** requested
- Types of operations may be controlled:
  - **Read**: read from file
  - **Write**: write or rewrite to file
  - **Execute**: load file into memory & execute it
  - **Append**: write new data at the end of file
  - **Delete**: delete file and free its space for possible reuse
  - **List**: list name & attributes of file

- Other operations like rename, copy, edit may also be controlled
- Protection is provided at only **Lower level**, all system may not have same type of protection ex: research group & large corporate computers

Access Control:

- Common approach is to make **access dependent on identity of user**
- Different users may need different types of access to file or directory
- Access Control List(ACL), most general scheme to implement identity dependent access to each file & directory
- ACL specifies user names & type of access allowed for each user
- When user request access to particular file:
  - OS checks access list associated with that file
  - If user is listed for the requested access, access is allowed
  - Otherwise, protection violation occurs and access denied
- **Advantage**: enables complex access methodologies
- **Problem**: with access lists is their length
  - If we want to allow everyone to read file, must list all users with read access
  - It requires list of users in system, constructing such list is waste
  - In general Directory is fixed size, now it must be variable size
- It can be resolved by **Condensed version of access list**
- Classification of users in connection with each file:
  - **Owner**: creator of file
  - **Group**: set of users who share the file
  - **Universe**: all other users
- Most common recent approach is combine access control lists with above classes
- Ex: User 1 able to use all operations, User2, 3 able to use only Read & Write, all others able to read only
- This can be achieved by creating a Group associated with the file, set access permissions as required
- Permissions & Access lists must be maintained tightly
- In case of UNIX:
  - Groups can be created or modified by the managers of system
  - In case of limited protection, only three fields are needed with 3 bits – RWX
  - More protection, can be included with 9 bits for 3 class of 3 fields each
- Windows XP users mange access control lists via GUI.

Other Protection Approaches:

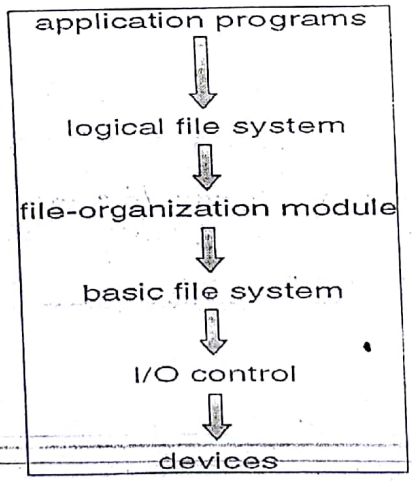
- Associate password with each file like computer password
- Disadvantages:

- Number of passwords that user needs to remember become large
- If only one password is used for all files & is discovered all files are accessible

- Some system allow user to associate password for sub directory rather than file
- Old OS's like DOS provide limited file protection
- In multilevel directory structure, we need to protect not only individual file but also collection of files in sub directories
- **Directory protection** is also required by restricting directory creation, deletion.

## 7. File System Structure:

- Disk provides bulk of secondary storage on which file system is maintained
- They have two characteristics:
  - Disk can be written in place
  - Disk can access directly any given block of information it contains
- Disk are performed in units of **Blocks** to improve I/O efficiently, transfer
- File system poses two quite different design problems:
  - Defining how file system should look to user i.e. defining file & its attributes, operations
  - Creating algorithms & data structures to map logical file system to physical storage
- File system itself composed of many different levels




- **I/O Control level**: consists of **Device drivers** and **Interrupt handlers** to transfer data between memory & disk
- **Basic File System**: issues only generic commands to appropriate device driver to read & write physical blocks on disk
- **File Organization Module**: knows about file and their logical & physical blocks
- **Logical File System**: manages meta data information, it includes all file system structure except actual data

- Logical file system manages directory structure to provide file organization module with information of given symbolic file name
- It maintains file structure via **File Control Block (FCB)**
- FCB contains information about file including ownership, permissions, and location of file contents
- Most OS support more than one file systems
- EX: Unix uses **UNIX file system (UFS)** which is based on Berkeley Fast File System (BFFS)
- Windows support **FAT, FAT32, NTFS**
- Linux file system is known as **External File System**

## 8. Allocation Methods:

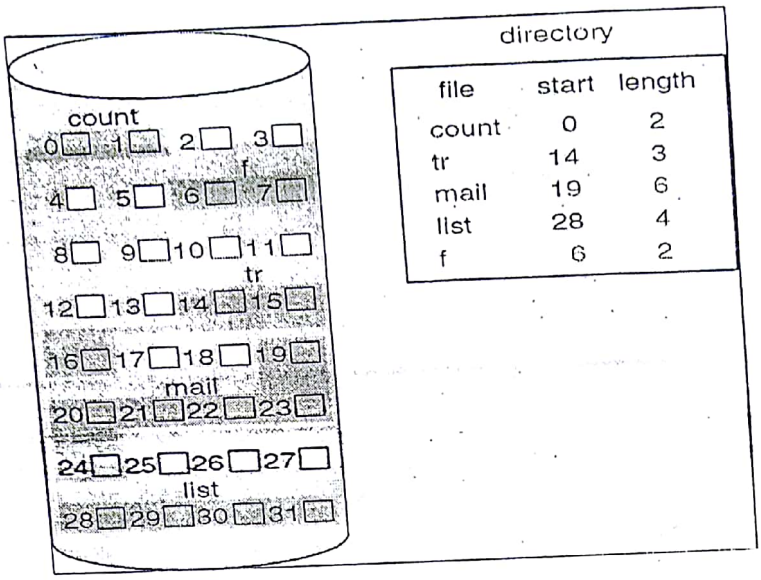
- Many files are stored on same disk. Main problem is how to allocate space to files for effective utilization.
  - For effective utilization of disk space
  - Quick access of files
- Three major methods are as follows: contiguous, linked and indexed

### 8.1 Contiguous Allocation:

- Each file occupy set of contiguous blocks on disk, disk addresses define linear ordering on disk.
  Ex: IBM VM/CMS uses contiguous allocation
- Contiguous allocation of files is defined by **disk address & length of block.** Directory entry for each file indicates **address of starting block & length of area allocated for this file**
- Accessing is easy:
  - Sequential Access, disk address of last reference must be remembered
  - Direct Access, block $i$ of file start at block $b$ can access $b+i$ directly
- Problems:
  - Finding space for new file
  - Dynamic storage allocation
  - External fragmentation
- Another problem is determining how much space is need for file.
- When file is created, total amount of space it will need must be found
- How does creator know size of file to be created?
  - Size of output file may be difficult to estimate
  - If we allocate too little space to file, file cannot be extended
  - With Best fit allocation, space on both sides of file may be in use & cannot make file longer
  - Two possibilities exist:
    - Terminate program, allocate more space, run program again, these repeated runs are costly

- Find larger hole, copy contents to file to new space, release previous space, time consuming

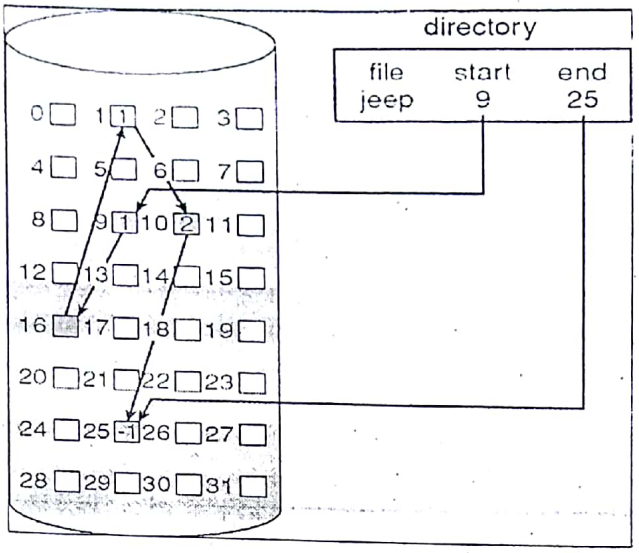
- Even if total amount of space need for file is known in advance, pre-allocation may inefficient for **slowly growing files**
  - To overcome external fragmentation compaction is used.




## 8.2 Linked Allocation:

- It solves all problems of contiguous allocation.
- Each file is linked list of disk blocks, disk blocks may be scattered any where on disk
- Directory contains pointer to first & last block of file
- Each block contains pointer to next block, pointers are not visible to user
- To create new file, simply create new entry in directory with **pointer to first disk block** initialized with **NIL, size is set to 0** to indicate **empty file**
- **Write to file** cause free space management to find free block & new block is written and linked to end of file
- **Reading file**, is simple by reading blocks with pointers from block to block
- There is no **External fragmentation**, any free block on free space is used. No need of **compaction**, file can grow as long as free blocks are available
- **Disadvantages:**
  - Inefficient for Direct access
    - It can be effectively used for **sequential access** files
    - To find $i^{th}$ block of file for **Direct access**, start at beginning of that file and follow pointers until we got $i^{th}$ block
  - Another disadvantage is **Space required for pointers.**
  - Reliability
    - If pointers were lost or damaged due to bug in OS or disk hardware failures
    - This error will result in linking into free space list or another file

- One solution is use **Doubly linked list** and store file name & relative block number in each block .

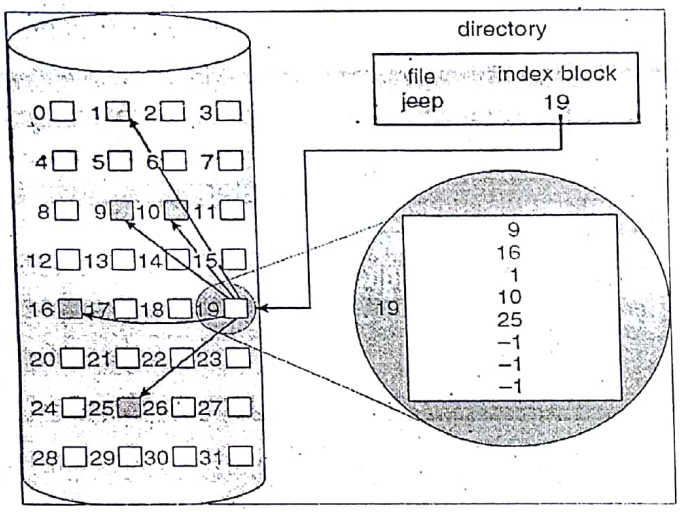



## 8.3 Indexed Allocation:

- Linked allocation solves External fragmentation, Size declaration problems. In absence of FAT, linked allocation can not support efficient direct access.
- Indexed allocation solves this problem by bringing all pointers together into one location known Index block. Each file has its **own index block**, which is **array of disk block addresses.** $i^{th}$ entry in index block points to $i^{th}$ block of file.
- Directory contains address of index block. To find or read $i^{th}$ block, use pointer in $i^{th}$ index block entry like paging.
- When pointer is created, all pointers in index block are set to NIL When $i^{th}$ block is first written, block is obtained form free space and its address is put in $i^{th}$ index block entry.
- Supports direct access with out suffering from external fragmentation
- Indexed allocation suffer from **wasted space**
- Pointer overhead of index block is generally greater than pointer overhead of linked allocation
- An entire index block must be allocated, even if only one or two pointers will be non NIL
- Every file must have index block, so index block to be as small as possible
- If index block is too small, it will not be able to hold enough pointers for large file, mechanism for this purpose include following:
- **Linked Scheme:**
  - Index block is normally one disk block, can read & write directly by it self
  - To allow for large files, we may link together several index blocks
  - Index block contains small header giving name of file & set of first 100 disk block address
  - Next address is pointer to another index block
- **Multilevel Index:**
  - Use first level index block to point to set of second level index blocks

- Which in turn point to file blocks
- To access block OS uses first level index to find second level index to find desired block
- N levels of index can be implemented

- **Combined Scheme:**
  - UFS uses this approach, which will keep first **15 pointers of index block** in files inode
  - First **12 pointers** point to **direct blocks**, contains address of data block
  - Next **3 pointers** points to **indirect blocks**
    - First indirect block pointer is address of **Single indirect block**
    - Single indirect block is an index block, containing addresses of block
  - Combined Scheme: UNIX (4K bytes per block)




## 9. Free Space Management:

Since disk space is limited, we need to reuse space from deleted files for new files if possible. To keep track of free disk space, system maintains **Free space list.** It records all free disk blocks those are not allocated to some file or directory. To create file, search free space list for required amount of space & allocate that space to new file

### 9.1 Bit Vector:

- Frequently, Free space list is implemented as Bit map or Bit vector
- Each block is represented by 1 bit, if block is free, the bit is 1, it block is allocated, bit is 0
- Main advantage of this approach is
  - Its relative simplicity
  - its efficiency in finding first free block or N consecutive free blocks on disk
- Many computers supply bit manipulation instructions that can be used effectively for that purpose
- One technique for finding first free block on system that uses bit vector to allocate disk space is to sequentially check each word in bit map to see whether that value is not 0
- First non 0 word is scanned for first 1 bit, which is location of first free block

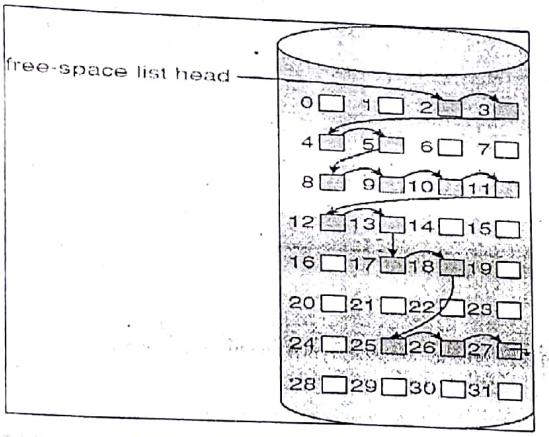
- Calculation of block number is:

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

- Hardware features driving software functionality
  - Bit vectors are inefficient unless entire vector is kept in main memory
  - Keeping it in main memory is possible for smaller disks but not for larger disks.

## 9.2 Linked List:

- Linking together all free disk blocks, keeping pointer to first free block in special location on disk & caching it in memory
- First block contain pointer to next free disk block & so on
- However, it is not efficient
  - To traverse list, must read each block
  - Which requires substantial I/O time
- Usually OS simply needs to free block so that it can allocate that block to file, so first block in free list is used
- FAT method incorporates free block accounting into allocation data structures, no separate method is needed




## 9.3 Grouping:

- Modification of free list approach is to store address of N free blocks in first free block
- First N-1 of these blocks are actually free, last block contains addresses of another N free blocks
- Address of large number of free blocks can now be found quickly

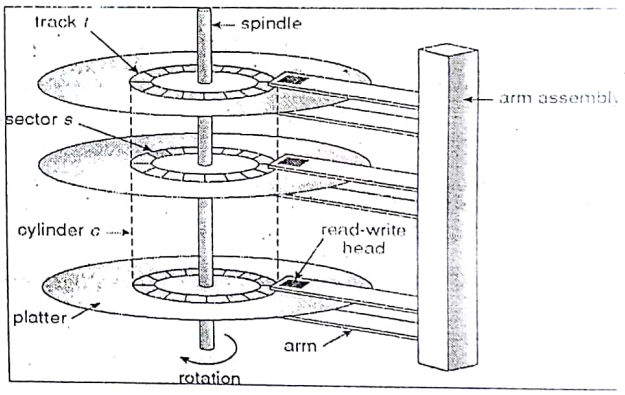
## 9.4 Counting:

- Several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with contiguous allocation algorithm or through clustering
- Rather than keeping list of N free disk addresses, we can keep address of first free block & number N of free contiguous blocks that follow first block. Each entry in free space list then consists of disk address & count. Although each entry requires more space than simple disk address, overall list will be shorter as long as count is greater than 1.

## erview of Mass Storage Structure:

### agic Disks:

- Magnetic disks provide bulk of secondary storage of modern computers
  - **Platter** is flat circular disk with two surfaces covered with magnetic material
  - Data is stored by recording it magnetically on these platters
  - **Read-Write head** flies just above each surface of ever platter, they are attached to **Disk arm** that moves all heads as unit
  - Surface of platter is logically divided into circular **Tracks**, which are sub divided into **sectors**, Set of tracks that are at one arm position make up **Cylinder**
  - Most drives rotate at 60 to 200 times per second
  - Disk speed has two parts(typically in MB per Second):
    - **Transfer rate** is rate at which data flow between drive and computer
    - **Positioning time (random-access time)** is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - Disk platter is coated with protective layer to avoid contact between head & disk
- Sometimes head will damage magnetic surface called as **Head crash,** head crash normally cannot be repaired
- Disks can be **removable**, removable disks have one platter like **Floppy disk**.
- Floppy disk is inexpensive with **soft plastic case & flexible platter**
- Head of Floppy disk sits directly on disk surface, disk will be rotate slowly




- Disk Drive is attached to computer by set of wires known as **I/O bus**
  - Busses vary, including **EIDE, ATA, SATA, USB, Fibre Channel, SCSI**
  - **Controllers**: special processors used for Data transfer on bus
    - **Host controller** is controller at computer end of bus
    - **Disk controller** is built into each disk drive for Disk I/O
  - Computer places command on host controller, then sent to disk controller
  - Disk controller operates disk drive hardware to carry out command

### Magnetic Tapes:

- It is relatively permanent & can hold large quantities of data with slow access
- In addition, random access on magnetic tape is hundred times slower than magnetic disk
- Tapes are used mainly for backup for storage of infrequently used information
- Moving to correct spot on tape can take minutes, it can write data fastly

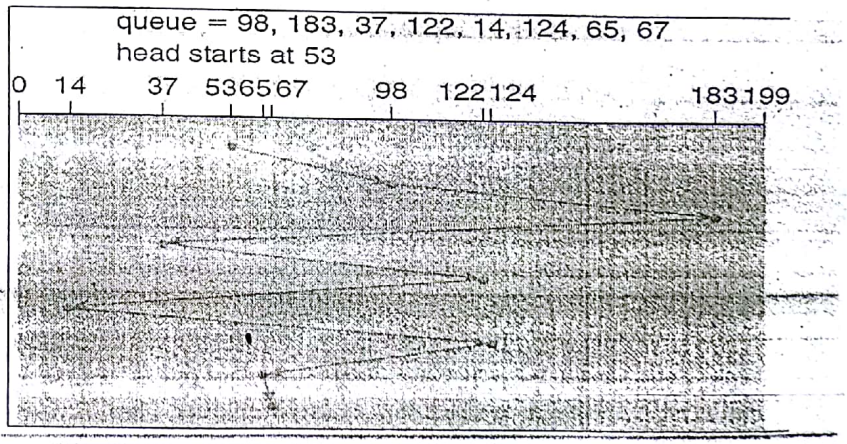
- Tape capabilities vary generally, depending on particular kind of tape drive
- Types & their drivers are usually categorized by width

## Disk Scheduling:

- One of responsibilities of OS is to use hardware efficiently
- For disk drives, **fast access time & large bandwidth** should be achieved
- Access time has two components:
  - **Seek time**: is time for disk arm to move head to cylinder containing desired sector
  - **Rotation Latency**: is time for disk to rotate desired sector to disk head
- Disk bandwidth is total number bytes transferred divided by total time between first request of service & completion of last transfer
- Disk Scheduling is used to improve both Access time & Bandwidth
- I/O is done by **system call.** it specifies several pieces of information:
  - Whether this operation is input or output
  - What is the disk address for transfer
  - What is the memory address for transfer
  - What is the number of sector to be transfer
- If the desired disk drive & controller are available, request can be serviced
- If it is busy, request will be placed in queue of pending requests of drive

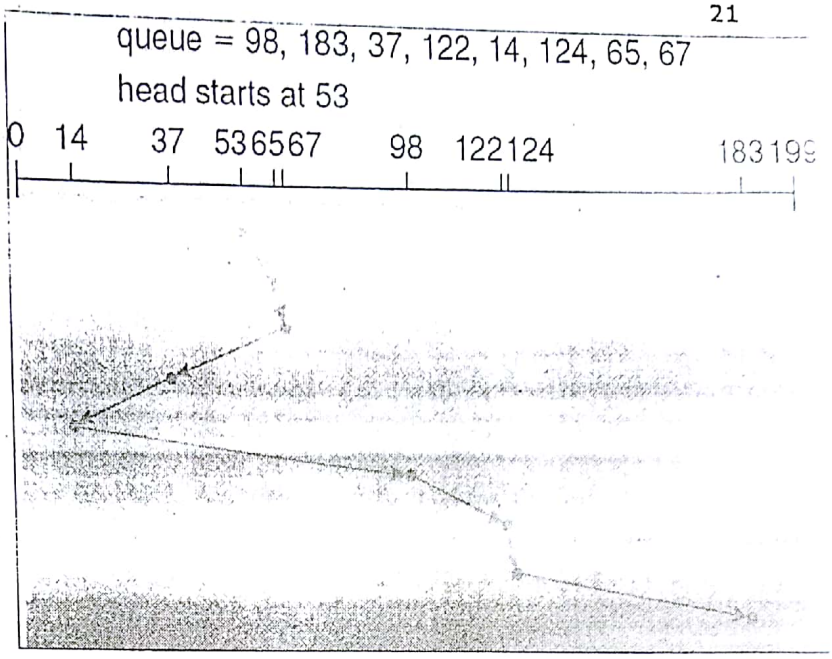
### FCFS Scheduling:

- Simplest form of disk scheduling is **First Come First Served(FCFS)** algorithm
- It is fair, but does not provide fastest service
- Ex: consider disk queue with request for I/O for blocks on cylinders
  98, 183, 37, 122, 14, 124, 65, 67



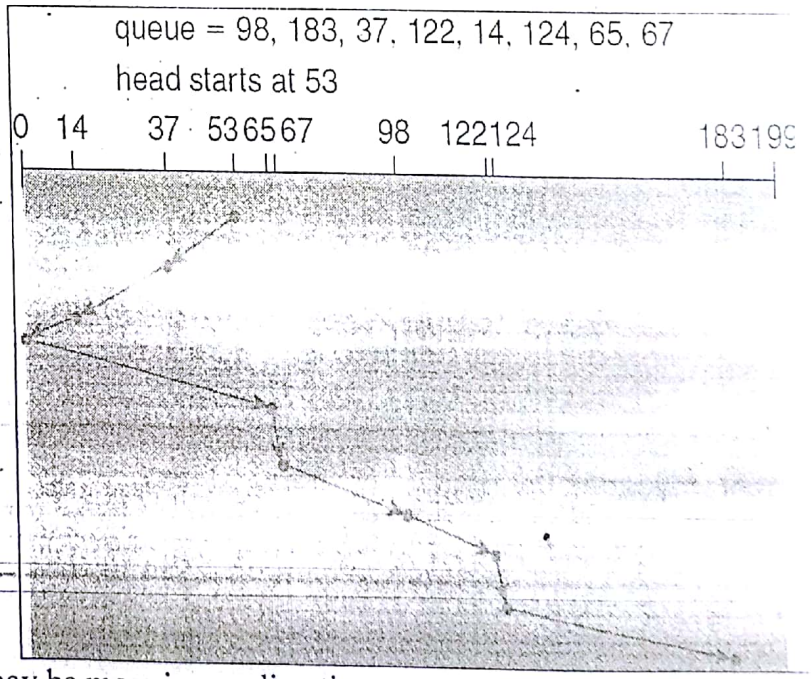

- **Problem:**
  - Wild swing from 122 to 14 & then back to 124
  - Will performance of head movement

**F Scheduling:**




- It serves all requests close to the **current head position**
- **Shortest Seek Time First (SSTF)** algorithm will select request with **minimum seek time** from current head position
- It chooses pending request closes to current head position
- It gives improved performance than FCFS algorithm
- SSTF is essentially a form of SJF scheduling
- Ex: 98, 183, 37, 122, 14, 124, 65, 67
- **Problem:**
  - Similar as SJF it also had **Starvation** problem
  - the request with largest seek time will always for the shortest seek time requests

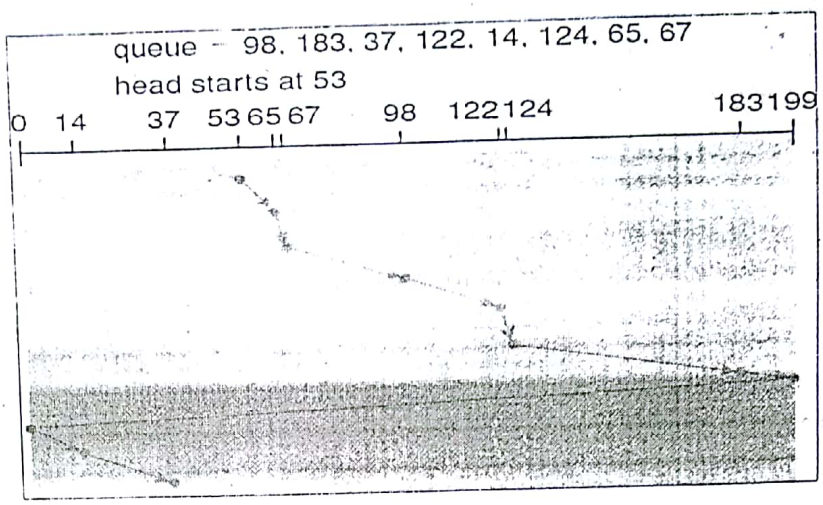
**SCAN Scheduling:**




- Disk arm starts at one end of the disk & moves toward other end
- It serves requests as it reaches each cylinder until it get to other end of disk
- At other end, direction of head movement is reversed & continues serving requests
- Head continuously scans back & forth across disk
- It is known **Elevator Algorithm**
- Ex: 98, 183, 37, 122, 14, 124, 65, 67
- **Problem:**
  - Density of requests in one direction may be more in one direction
  - If heaviest density requests are at other end of disk, they have to wait for longest time

C-SCAN Scheduling:

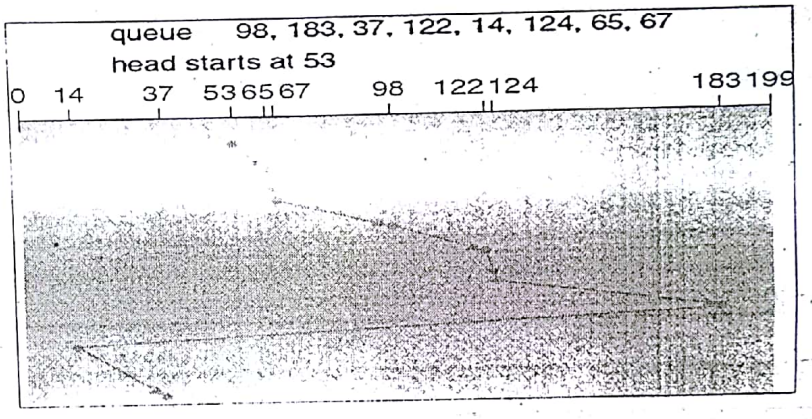
- Circular SCAN is designed to provide more **uniform wait time**
- In this head moves from one end to other end of disk serves requests along the way
- When head reaches other end, it immediately returns to beginning of disk without serving any request on return trip



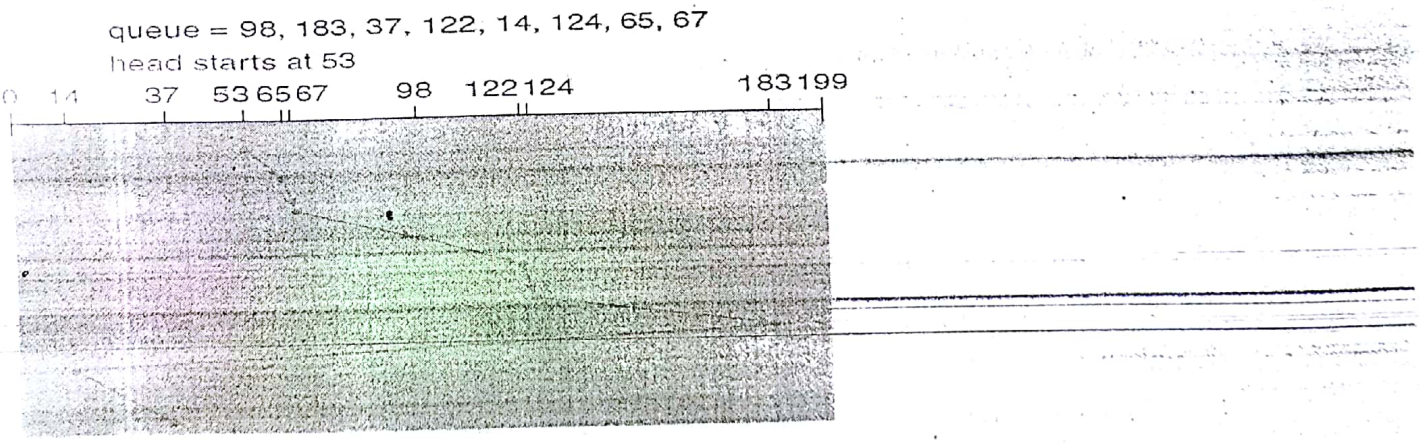

- It will treat cylinders as circular list that wraps around from final cylinder to first one

LOOK Scheduling:

- Both SCAN & C-SCAN move disk arm across full width of disk
- In LOOK, Arm goes only as far as final request in each direction
- It reverses direction immediately, without first going all way to end of disk




C-LOOK Scheduling




Selection of Disk scheduling algorithm:

- How do we choose best one?
  - SSTF is common and has natural appeal
  - SCAN & C-SCAN perform better for systems that place heavy load on disk
- Performance depends heavily on **number & types of requests**
- Requests for disk service greatly **influenced by file allocation method**