FILE ACCESS METHODS 1a

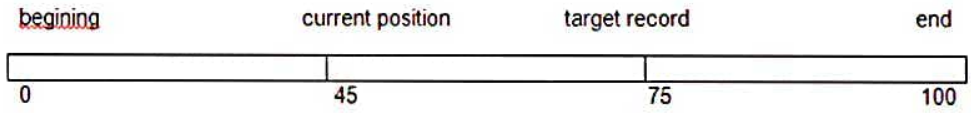
Files stores information, this information must be accessed and read into computermemory. There are so many ways that the information in the file can be accessed.

1. Sequential file access:

Information in the file is processed in order i.e. one record after the other.Magnetic tapes are supporting this type of file accessing.

Eg : A file consisting of 100 records, the current position of read/write head is 45th record, suppose we want to read the 75th record then, it access sequentially from 45, 46, 47

…….. 74, 75. So the read/write head traverse all the records between 45 to 75.



2. Direct access:

Direct access is also called relative access. Here records can read/write randomly without any order. The direct access method is based on a disk model of a file, because disks allow random access to any file block.

Eg : A disk containing of 256 blocks, the position of read/write head is at 95th block. Theblock is to be read or write is 250th block. Then we can access the 250th block directly without any restrictions.

Eg : CD consists of 10 songs, at present we are listening song 3, If we want to listen song 10, we can shift to 10.

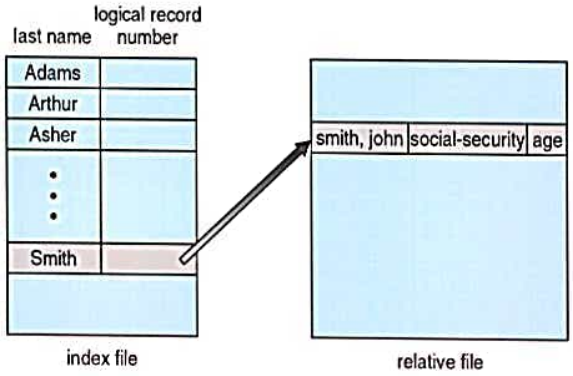
3. Indexed Sequential File access

The main disadvantage in the sequential file is, it takes more time to access a Record

.Records are organized in sequence based on a key field.Eg :

A file consisting of 60000 records,the master index divide the total records into 6 blocks, each block consisiting of a pointer to secondary index.The secondary index divide the 10,000 records into 10 indexes.Each index consisting of a pointer to its orginal

location.Each record in the index file consisting of 2 field, A key field and a pointer field.



1b

Certainly! File system mounting is the process of integrating a file system into the existing directory structure of an operating system (OS). Here's an illustration of how file system mounting works:

1. Initial State:

In the beginning, the operating system has its root file system mounted, which is typically stored on a dedicated partition or device.

2. Mounting Process:

Let's say we have an external storage device, such as a USB drive, that contains a file system (e.g., FAT32, NTFS, ext4). To access the files on this device, we need to mount its file system into the OS.

- The user or an automated process initiates the mount command, specifying the device and the target mount point (an empty directory where the file system will be attached).

- The OS locates the file system on the specified device and performs any necessary checks to ensure its integrity.

- The OS reads the file system's metadata, which includes information about directories, files, permissions, and other attributes.

- The OS links the file system's root directory to the specified mount point, integrating it into the directory structure.

3. Mounted State:

Once the file system is successfully mounted, it becomes accessible to the OS and its users. The mount point acts as an entry point to the file system, allowing users to navigate its directories and access files as if they were part of the local file system.

4. Multiple Mounts:

An operating system can support multiple mounted file systems simultaneously. Each file system is typically mounted at a unique mount point, providing separate access to their respective files and directories.

5. Unmounting Process:

When you're finished working with a mounted file system, it's important to unmount it properly to ensure data integrity. The unmounting process involves:

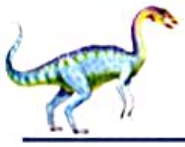
- Initiating the unmount command, specifying the mount point or device to be unmounted.

- The OS ensures that all pending read/write operations are completed and that there are no open files or active processes relying on the file system.

- The OS updates the file system's metadata and detaches it from the mount point, restoring the previous directory structure.

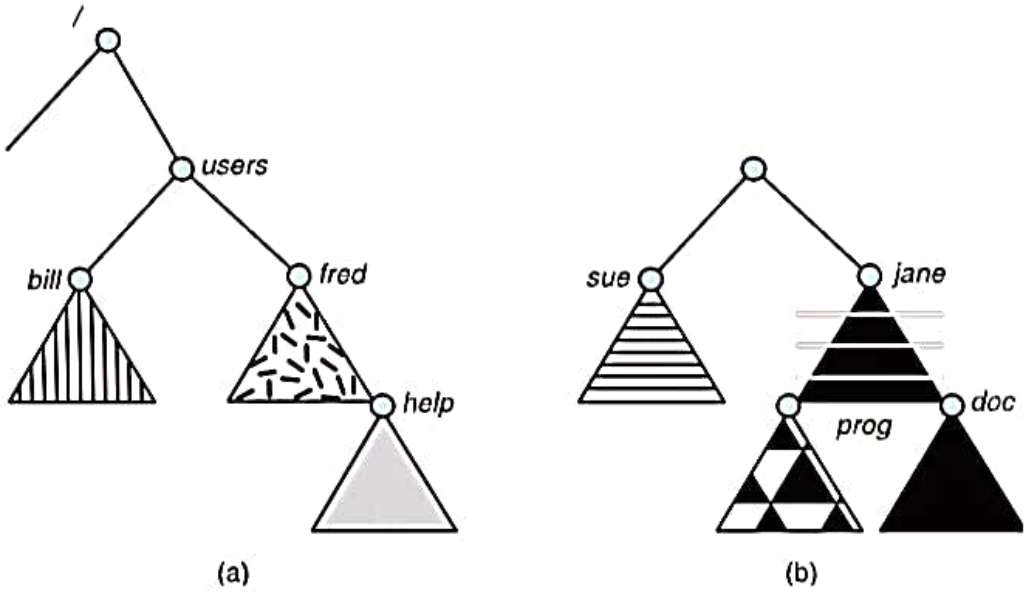
- Once unmounted, the device or file system can be safely removed from the system.

File system mounting is a crucial aspect of operating systems as it allows seamless integration of various storage devices and network resources into the file hierarchy, enabling users to access and manage data efficiently.

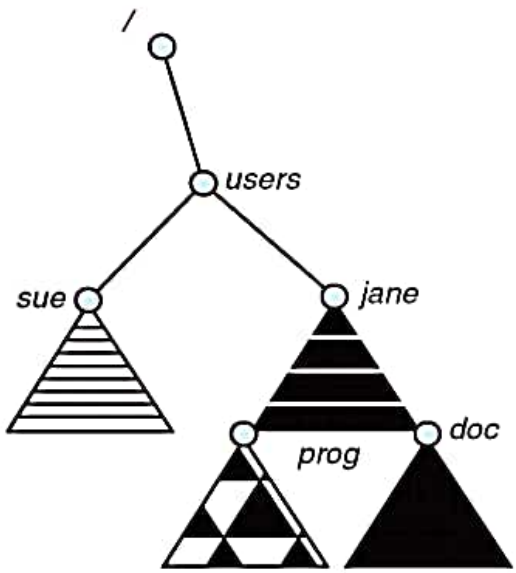


File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**

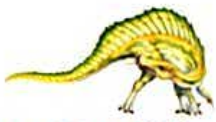


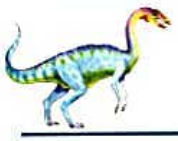
Mount Point



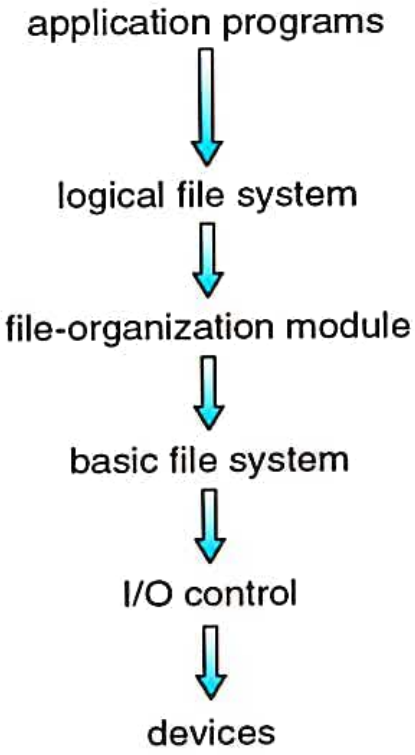
2 File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers





Layered File System



File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation

File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performanceTranslates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Logical layers can be implemented by any coding method according to OS designer

File-System Implementation

- There are 2 structure
- On-disk structure and in-memory structures

1. On-disk structure: this structure maintained on disk.

- Boot control block contains info needed to boot OS.
  - Hard disk has 1stblock i.e., Boot control block.
- Volume control block (superblock, master file table) contains size of block details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table-list of files.

File-System Implementation (Cont.)

- Per-file File Control Block (FCB) contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

File-System Implementation (Cont.)

- 2. In-Memory File System Structures- : this structure maintained on disk .It contains
- Mount table storing file system mounts, mount points.
- Directory structure-information about directories accessed recently
- System-wise open file table- list of files that are opened by processes.
- Per-process open file table- has list of files opened per process.

file structure in que 2

3 Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time- O(n)
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method
  - Takes constant search time O(1).

and counts.

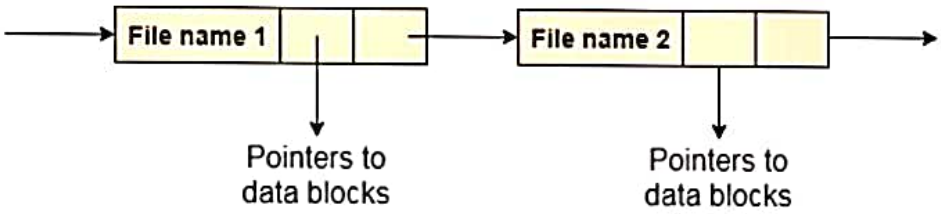
Directory Implementation

1. Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains thepointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics

1. When a new file is created, then the entire list is checked whether the new file name is matching to aexisting file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.



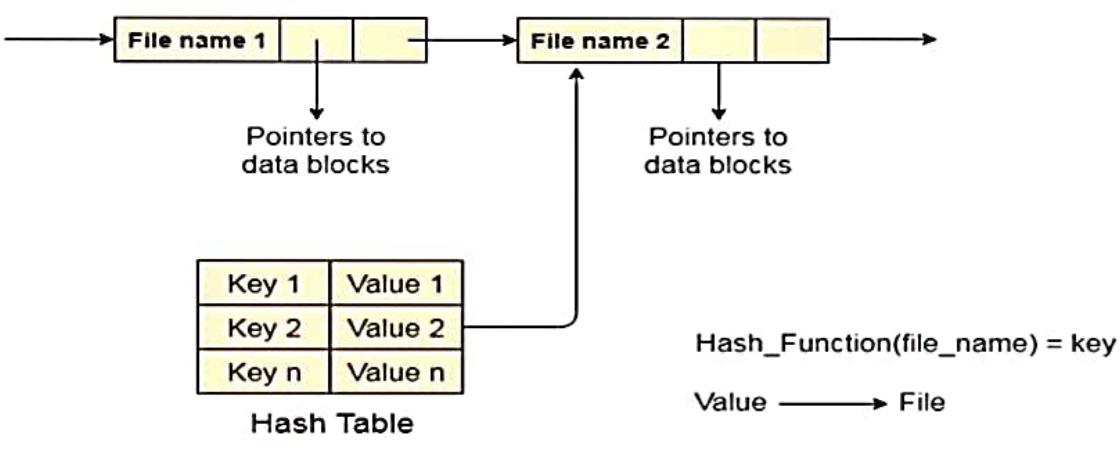
Linear List

2.Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



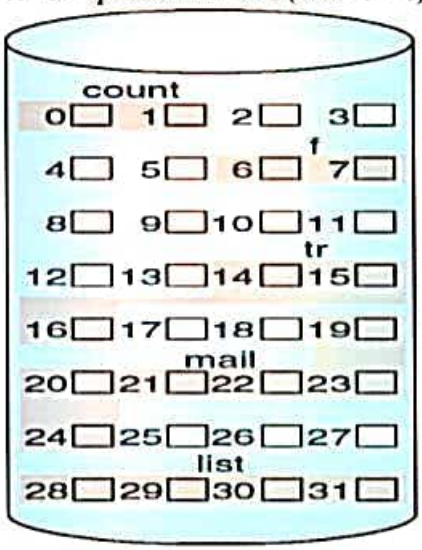
4

FILE ALLOCATION METHODS – [refer ppt for advantages and disadvantages]

An allocation method refers to how disk blocks are allocated for files:

1.Contiguous allocation – each file occupies set of contiguous blocks o Bestperformance in most cases

- o Simple – only starting location (block #) and length (number ofblocks) are required
- o Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line (downtime) or on-line**



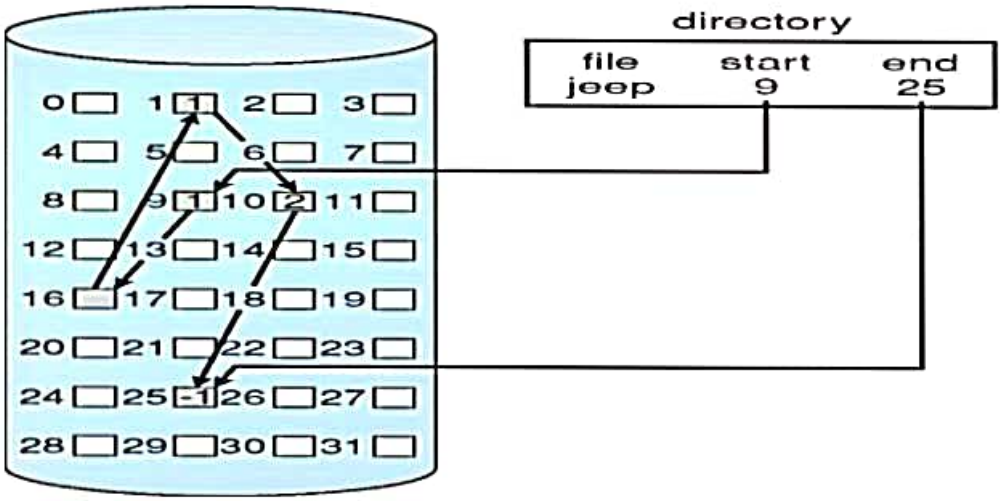
directory

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

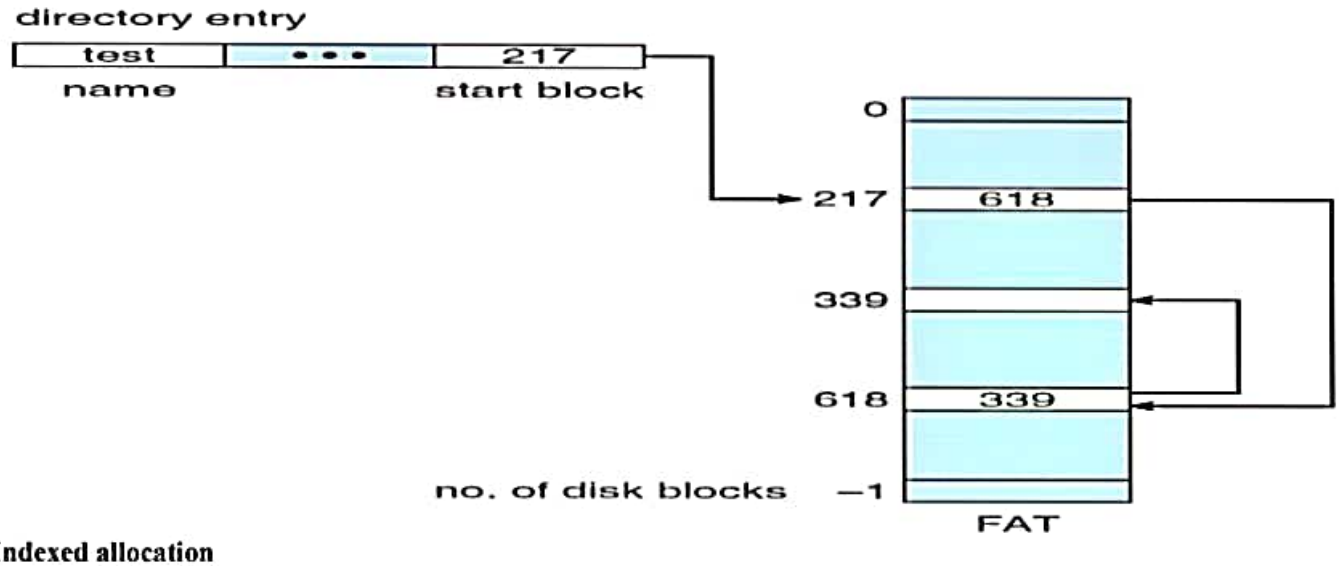
Linked

2.Linked allocation – each file a linked listof blocks o File ends at nil pointer

- o No external fragmentation
- o Each block contains pointer to next block
- o No compaction, external fragmentation
- o Free space management system called when new block needed
- o Improve efficiency by clustering blocks into groups butincreasesinternal fragmentation
- o Reliability can be a problem
- o Locating a block can take many I/Os and disk seeks FAT (File Allocation Table) variation
- o Beginning of volume has table, indexed by block number
- o Much like a linked list, but faster on disk and cacheable

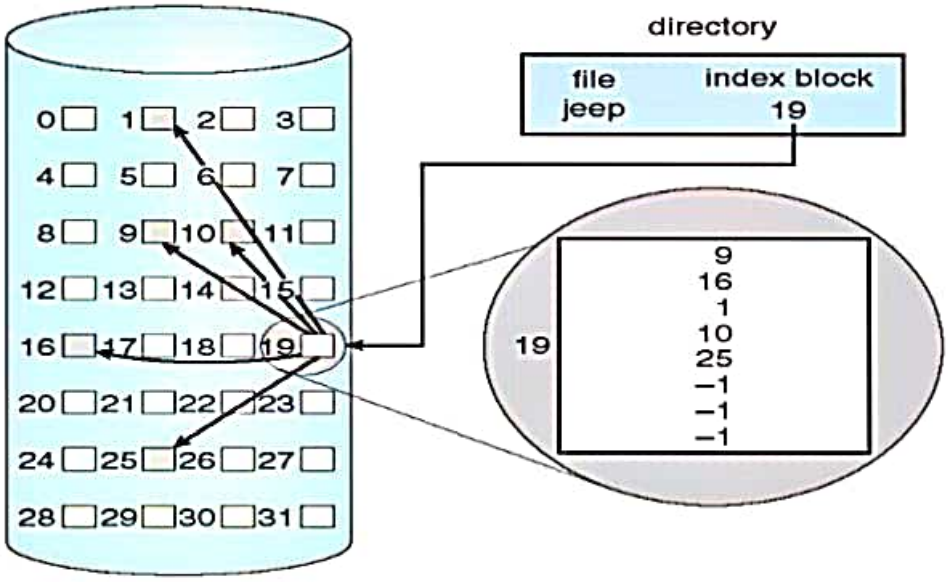


File-Allocation Table



3.Indexed allocation

- Each file has its own **index block**(s) of pointers to its data blocks



5

Free-Space Management

File system maintains **free-space list** to track availableblocks/clusters Linked list (free list)

- Cannot get contiguous space easily
- No waste of space
- No need to traverse the entire list

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

Advantages –

Simple to understand.

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

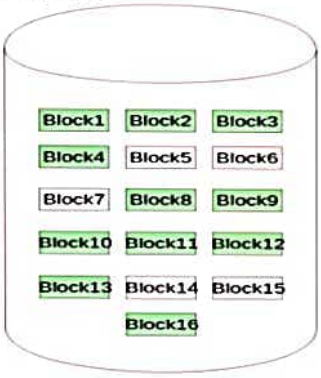
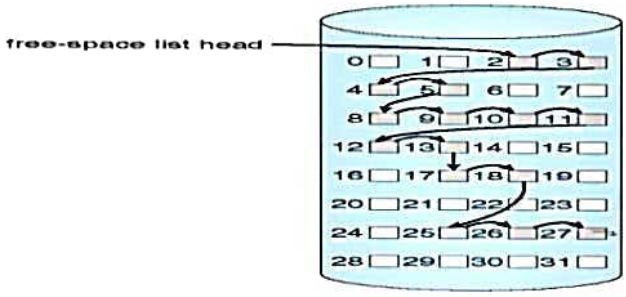


Figure - 1

2.Linked Free Space List on Disk



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

3.Grouping

Modify linked list to store address of next n-1 free blocks in first free block, plusa pointer to next block that contains free-block-pointers (like this one).

An **advantage** of this approach is that the addresses of a group of free disk blockscan be found easily

4.Counting

Because space is frequently contiguously used and freed, with contiguous- allocationallocation, extents, or clustering.

Keep address of first free block and count of following free blocks. Free space listthen has entries containing addresses and counts.

Disk Scheduling Algorithms

Disk scheduling algorithms are used to allocate the services to the I/O requests on the disk . Since seeking disk requests is time consuming, disk scheduling algorithms try to minimize this latency. 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 Operating System has to choose which pending request to service next. The OS relies on the type of algorithm it needs when dealing and choosing what particular disk request is to be processed next. The objective of using these algorithms is keeping Head movements to the amount as possible. The less the head to move, the faster the seek time will be. To see how it works, the different disk scheduling algorithms will be discussed and examples are also provided for better understanding on these different algorithms.

I. **First Come First Serve(FCFS)**

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processes according to their arrival. The request arrives first will be accessed and served first. Since it follows the order of arrival, it causes the wild swings from the innermost to the outermost tracks of the disk and vice versa . The farther the location of the request being serviced by the read/write head from its current location, the higher the seek time will be.

Example: Given the following track requests in the disk queue, compute for theTotal Head Movement (THM) of the read/write head :

95, 180, 34, 119, 11, 123, 62, 64

Consider that the read/write head is positioned at location 50. Prior to this track location199 was serviced. Show the total head movement for a 200 track disk (0-199).

Solution:

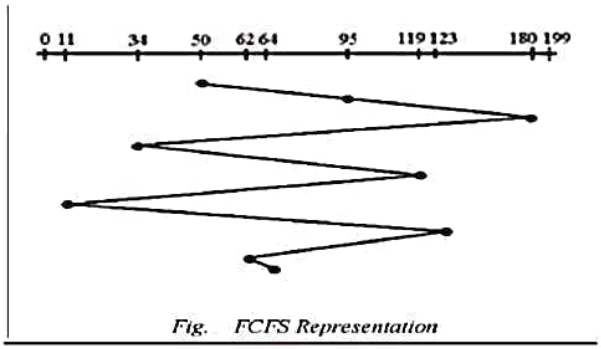


Fig. FCFS Representation

Total Head Movement Computation: (THM) =

(180 - 50) + (180-34) + (119-34) + (119-11) + (123-11) + (123-62) + (64-62) =

130 + 146 + 85 + 108 + 112 + 61 + 2 (THM) = 644 tracks

Assuming a seek rate of 5 milliseconds is given, we compute for the seek timeusing the formula: Seek Time = THM * Seek rate

=644 * 5 ms

Seek Time = 3,220 ms.

2. Shortest Seek Time First(SSTF):

This algorithm is based on the idea that that he R/W head should proceed to the track that is closest to its current position . The process would continue until all the track requests are taken care of. Using the same sets of example in FCFS the solution are as follows:

Solution:

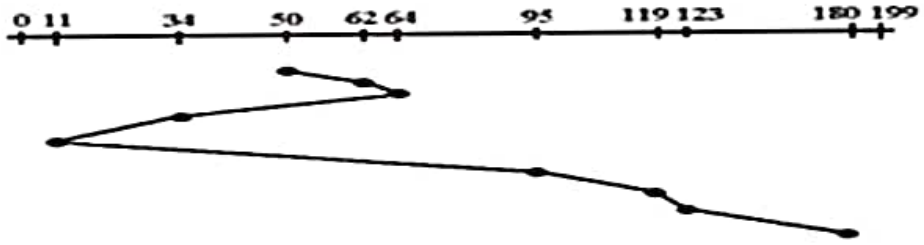


Fig. SSTF Representation

(THM) = (64-50) + (64-11) + (180-11) =

14 + 53 + 169 (THM) = 236 tracks

Seek Time = THM * Seek rate

= 236 * 5ms

Seek Time = 1,180 ms

In this algorithm, request is serviced according to the next shortest distance. Starting at50, 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 up to the last track request. There are a total of 236 tracks and a seek time of 1,180 ms, which seems to be

a better service compared with FCFS which there is a chance that starvation3 would take place. The reason for this is if there were lots of requests closed to each other, the other requests will never be handled since the distance will always be greater.

3. SCAN Scheduling Algorithm

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm. Using thesame sets of example in FCFS the solution are as follows:

Solution:

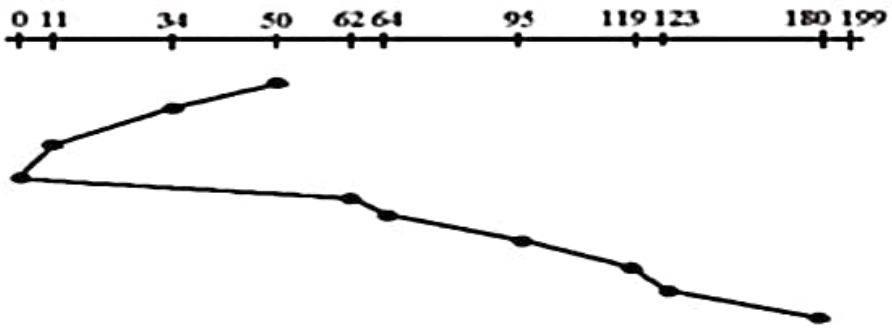


Fig. SCAN Representation

$$(THM) = (50-0) + (180-0)$$
$$= 50 + 180$$
$$\textbf{(THM)} = \textbf{230}$$

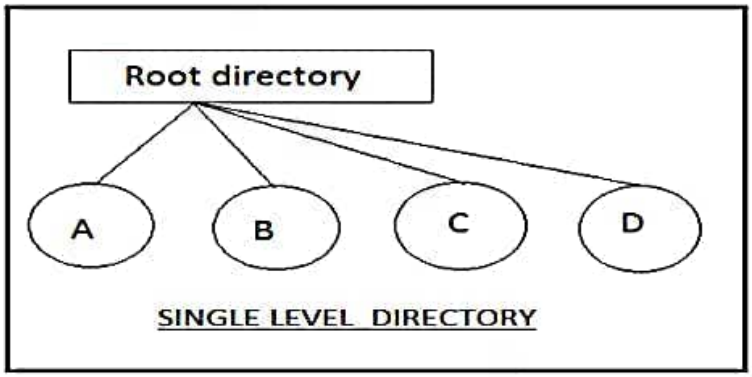
$$Seek\ Time = THM * Seek\ rate$$
$$= 230 * 5ms$$
$$\textbf{Seek Time} = \textbf{1,150 ms}$$

This algorithm works like an elevator does. In the algorithm example, it scans down towards the nearest end and when it reached the bottom it scans up servicing the requests that it did not 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 and a seek time of 1,150. This is optimal than the previous algorithm.

The **VARIOUS DIRECTORY STRUCTURES [refer ppt for -advantages and disadvantages]**

1. Single level directory:

The directory system having only one directory,it consisting ofall files some times it is said to be root directory.



E.g :- Here directory containing 4 files (A,B.C,D).the advantage of the schemeis its simplicity and the ability to locate files quickly.The problem is different users may accidentally use the same names for their files.

E.g :- If user 1 creates a files caled sample and then later user 2 to creates a file called sample,then user2's file will overwrite user 1 file.Thats why it is not usedin the multi user system.

Advantages:

- Simplicity: Single-level directories are straightforward to implement and understand.
- Easy file access: With a flat structure, file access is direct, as there are no nested directories to navigate.

Disadvantages:

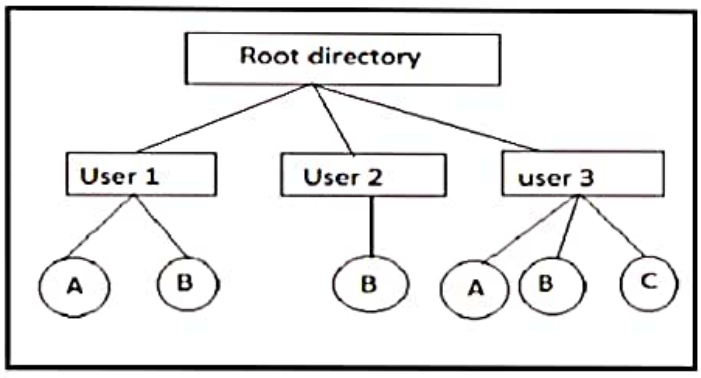
- Limited scalability: As the number of files increases, it becomes difficult to manage and locate specific files.
- Lack of organization: Without subdirectories, it's challenging to organize files based on categories or topics.
- Naming conflicts: Since file names must be unique within the directory, naming conflicts can arise if two files have the same name.

2. Two level directory:

The problem in single level directory is different user may be accidentally use

the same name for their files. To avoid this problem each user need a privatedirectory,

Names chosen by one user don't interfere with names chosen by a differentuser.



Root directory is the first level directory.user 1,user2,user3 are user level ofdirectory A,B,C are files.

Advantages:

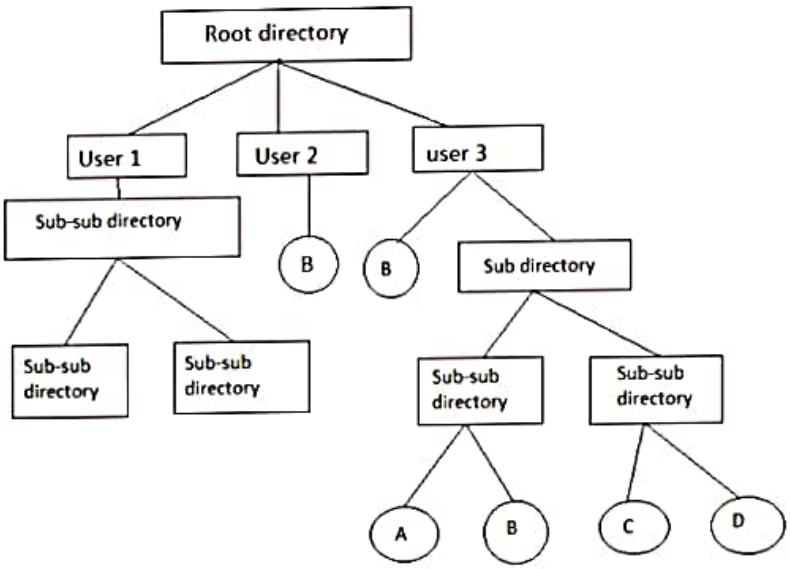
- Improved organization: Each user has their own directory, making it easier to organize and manage files.
- User isolation: Users can have files with the same name, as long as they are in different directories.
- Simplicity: The two-level structure is relatively simple to implement compared to more complex structures.

Disadvantages:

- Limited collaboration: It can be challenging for users to share files with others since they are stored in separate directories.
- Duplication of user names: If two users have the same username, conflicts may occur when creating their directories.
- Increased complexity with more users: Managing a large number of users can become difficult due to the manual mapping of names to directories.

3. Tree structured directory:

Two level directory eliminates name conflicts among users but it is not satisfactory for users with a large number of files.To avoid this create the sub- directory and load the same type of files into the sub-directory.so, here each can have as many directories are needed.



There are 2 types of path

1. Absoulte path
2. Relative path

Absoulte path : Begging with root and follows a path down to specifiedfiles giving directory, directory name on the path.

Relative path : A path from current directory.

Advantages:

- Scalability and organization: The hierarchical structure allows for an unlimited number of directories and subdirectories, providing scalability and improved organization.
- Easy file management: Files can be logically grouped into directories based on categories, projects, or any other desired criteria, making file management more intuitive.
- Shared resources: Subdirectories can be shared with specific users or user groups, enabling collaboration and easy access control.

Disadvantages:

- Complexity: Tree-structured directories can become complex as the hierarchy deepens and the number of files and directories increases.
- Path length limitation: Some operating systems impose limits on the length of file paths, which can restrict the depth of the directory structure.
- Performance impact: Searching for a file in a deep directory hierarchy may require traversing multiple levels, potentially impacting performance.

There are 2 types of path

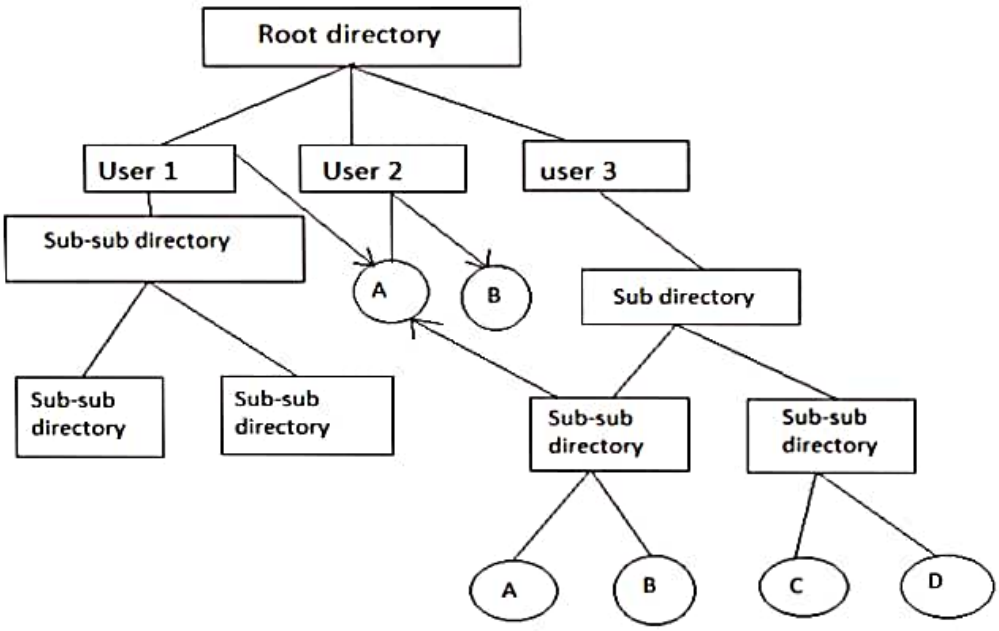
1. Absoulte path
2. Relative path

Absoulte path : Begging with root and follows a path down to specifiedfiles giving directory, directory name on the path.

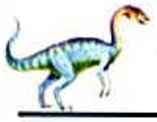
Relative path : A path from current directory.

4. Acyclic graphdirectory

Multiple users are working on a project, the project files can be stored in a comman sub-directory of the multiple users. This type of directory is called acyclic graph directory .The common directory will be declared a shared directory. The graph contain no cycles with shared files, changes made by one user are made visible to other users.A file may now have multiple absolute paths. when shared directory/file is deleted, all pointers to the directory/ files also to be removed.

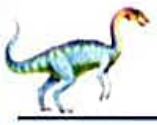


Advantages :- Traversing is easy. Easy sharing is possible.



10 C-SCAN-Circular SCAN

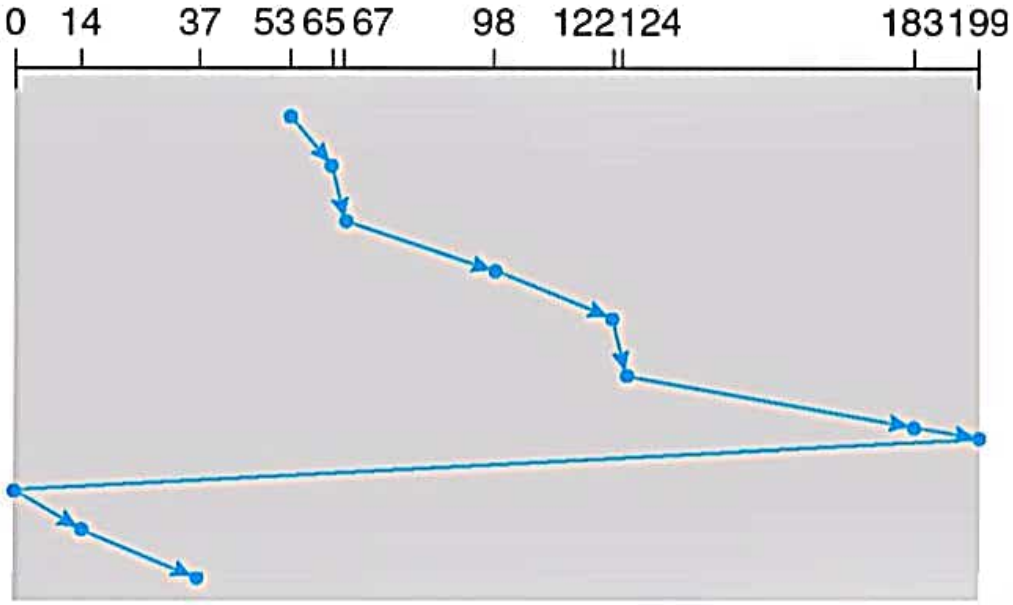
- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

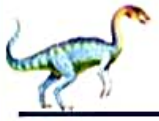


C-SCAN (Cont.)

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



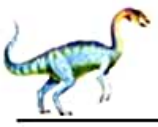


LOOK Scheduling

- They look for a request before move to other direction.
- It goes to the final request not to end of the disk.
- Arm only goes as far as the last request in each direction.

C-LOOK Scheduling

- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



C-LOOK (Cont.)

queue 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53