UNIT-5

File Management

1) File Overview:

A file is a named collection of related information namely records on a secondry storage device such as disk or tape. Commonly file represents programs and data. Data files may be numeric, alphanumeric or binary. Files are logical units of information created by process.

File naming: Files are logical units of information created by process. When a process creates a file, it gives the file a name. When the process terminates, the file continues to exist and can be accessed by other processes using its name. The exact rules for file naming vary somewhat from system to system, but all current operating systems allows strings of one to eight letters as legal file names. Frequently digits and special characters are also permitted and are often valid as well. Many file systems support as long as 255 characters.

The newer operating systems have a much more advanced native file system (NTFS) that has different properties. There is second file system for Windows 8 known as ReFS (Resilient File System), but it is targated at the server version of Windows 8.

File Structure:- Files can be structed in several ways. The most common structures are:-

1) Unstructured → It consists of unstructured sequence of bytes or words. Os does not care what is in the file. Any meaning must be imposed by user level programs. It provides maximum facility user can put anything they want and name them in their own convinient way. Both UNIX and WINDOWS use this approach.

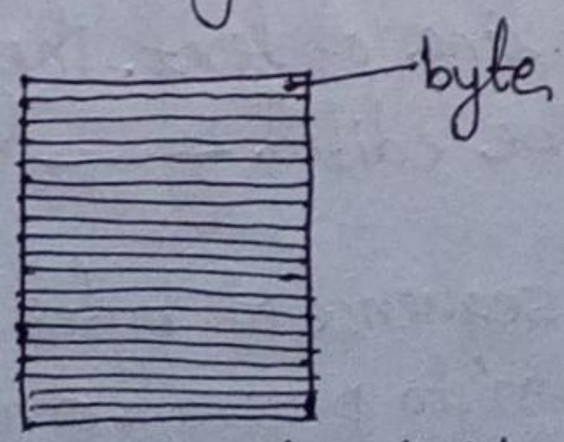


fig. unstructured file

ii) Record structured: It is a sequence of fixed length records each with some internal structure. Each read operation returns one record & write operation overwrites or append one record. Many old mainframe system use this structure.

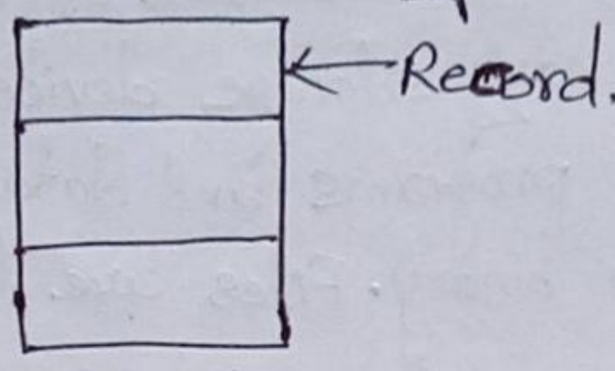


fig. Record structured

iii) Tree structured: It consists of tree of records, not necessarily all the same length. Each record containing a key field in a fixed position in the record and sorted on the key to allow the rapid searching. The operation is to get the record with the specific key. It is used in large mainframe systems for commercial data processing.

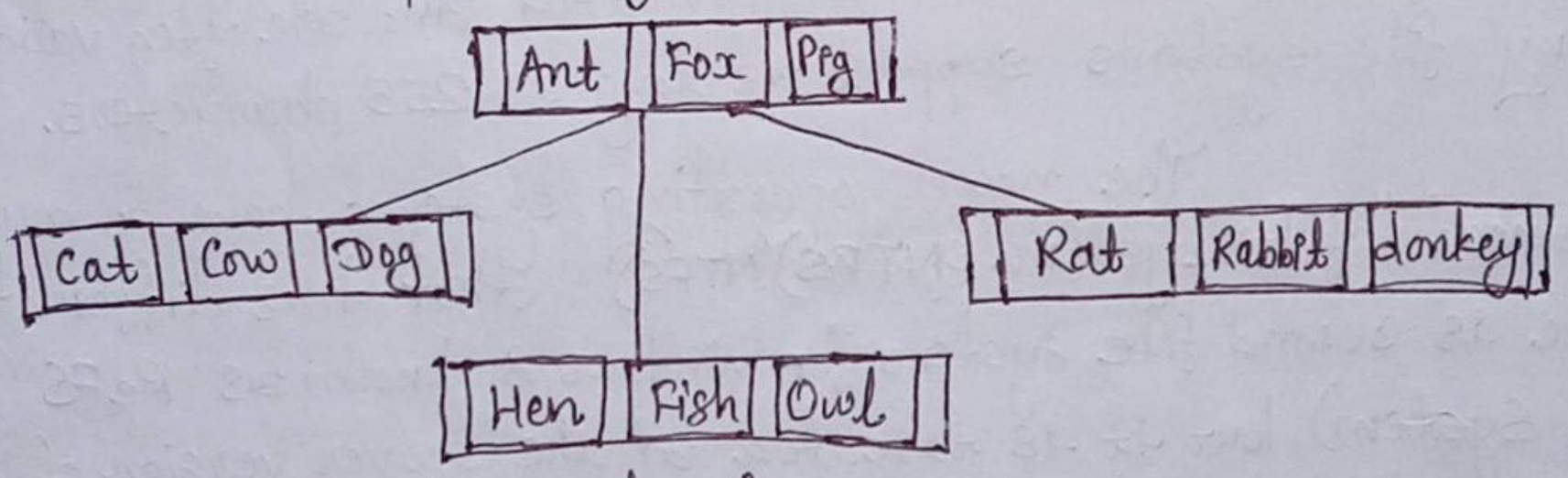


fig. tree structure

File Types:

i) Regular files → Contain user information, are generally ASCII or binary.

ii) Directories → System files for maintaining the structure of file system.

iii) Character special file → Related to I/O of computer to model serial I/O device such as terminals.

iv) ASCII files → Consists of line of text. They can be displayed and printed as it can be edited ~~by or~~ with ordinary text editor.

v) Binary files → Consists of sequence of bytes only. They have some internal structure known to program that use them. Executable files are its examples.

File Access:

i) Sequential file access → A sequential file is an ordered file. It is set of continiously stored records on a physical storage devices such as magnetic disk or CD ROM. To locate a particular record a program scan the file from the beginning until the desired record is located.

ii) Direct file access → File whose bytes or records can be read in any order is known as direct file access. It is based on the disk model of file, since disk allows random access to any block. Direct access is obtained by hashing method.

File Attributes: File attributes are settings associated with computer files that grant or deny certain rights to how a user or operating system can access that file. Some of the attributes of file are:-

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 ~~file systems~~ 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.

Archeive → Tells Windows backup to backup the file.

File Operations:

i) Create → The file is created with no data. The purpose of the call is to announce that the file is comming and to set some of the attributes.

ii) Delete → When the file is no longer needed, it has to be deleted to free up disk space.

iii) Open → Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.

v) Close → When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space.

vi) Read → Data are read from file.

vii) Write → Data are written to the file.

viii) Append → It can add data only to the end of the file.

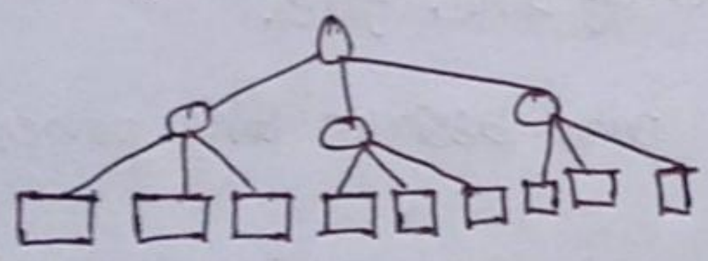
ix) Rename → It is used when the user needs to change the name of an existing file.

Directories:-

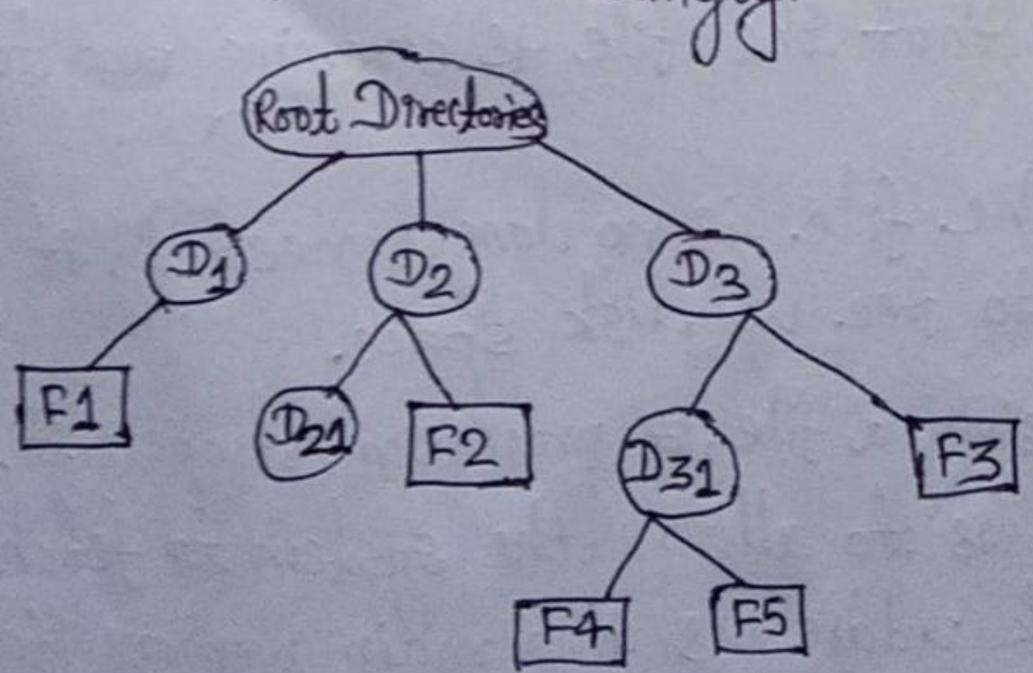
To keep track of files, file systems normally have directories or folders which are themselves files.

i) Single-level directory systems → The simplest form of directory system is having one directory containing all the files. Sometimes it is called the root directory, but since it is the only one, the name does not matter much.

ii) Two-level directory systems → It contains separate directory for each user. It is used on multi-user computer. It has problem when user want to co-operate on some task & to access one another files.



iii) Hierarchical directory systems → It is the generalization of two level structure to a tree of arbitary height. This allows the user to create their own sub-directories and to organize their file accordingly.



File-System Layout:

A file system is a set of files, directories and other structures. File systems maintain information and identify where a file or directory's data is located on the disk. In addition to files and directories, file systems contain a boot block, a super block, bitmaps and one or more allocation groups. An allocation group contains disk i-nodes and fragments. Each file system occupies one logical volume.

The boot block occupies the first 4096 bytes of the file system starting at byte offset 0 on the disk. The boot block is available to start the operating system. The superblock is 4096 bytes in size and starts at byte offset 4096 on the disk. The super-block maintains information about the entire file system.

2) Implementing Files:

Implementing file storage is keeping track of which disk blocks go with which file. Following are some of the methods.

a) Continious Allocation:

The simplest allocation scheme is to store each file as a contiguous run of disk blocks. Thus on a disk with 1-KB blocks, a 50-KB file would be allocated 50 consecutive blocks. With 2-KB blocks, it would be allocated 25 consecutive blocks. Each file begins at the start of a new block, so if some space is wasted then the wasted space is at the end of the last block.

Contiguous disk-space allocation has two significant advantages. First it is simple to implement & Second, the read performance is excellent. The disadvantages of contigious allocation are: for new files it is very difficult to find spaces, we can't extend the file and difficulty about fragmentation.

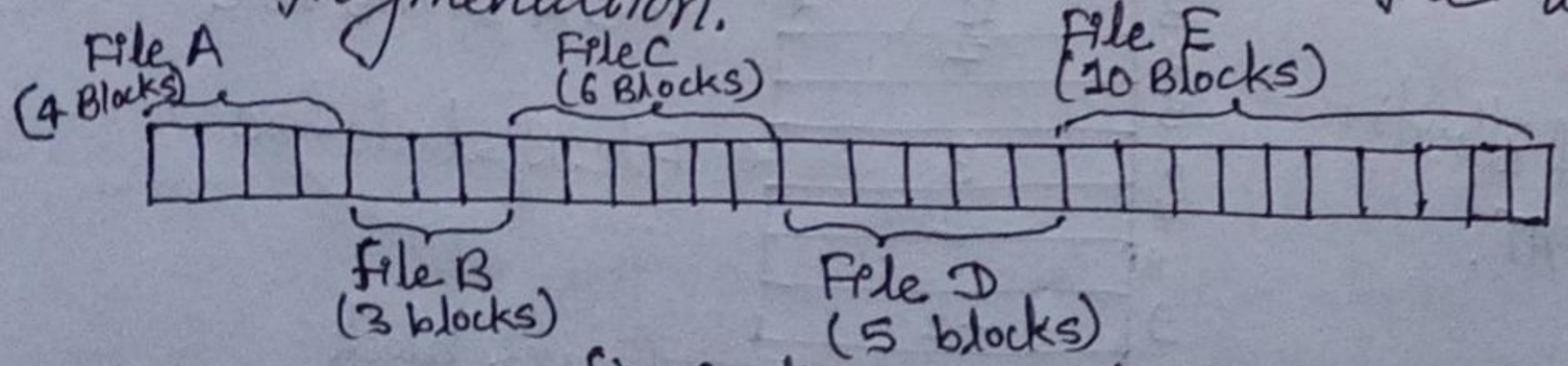


Fig: Continious allocation.

b) Linked-List Allocation:

This method for storing files is to keep each one file as a linked list of blocks. The first word of each block is used as a pointer to the next one. The rest of the block is for data.

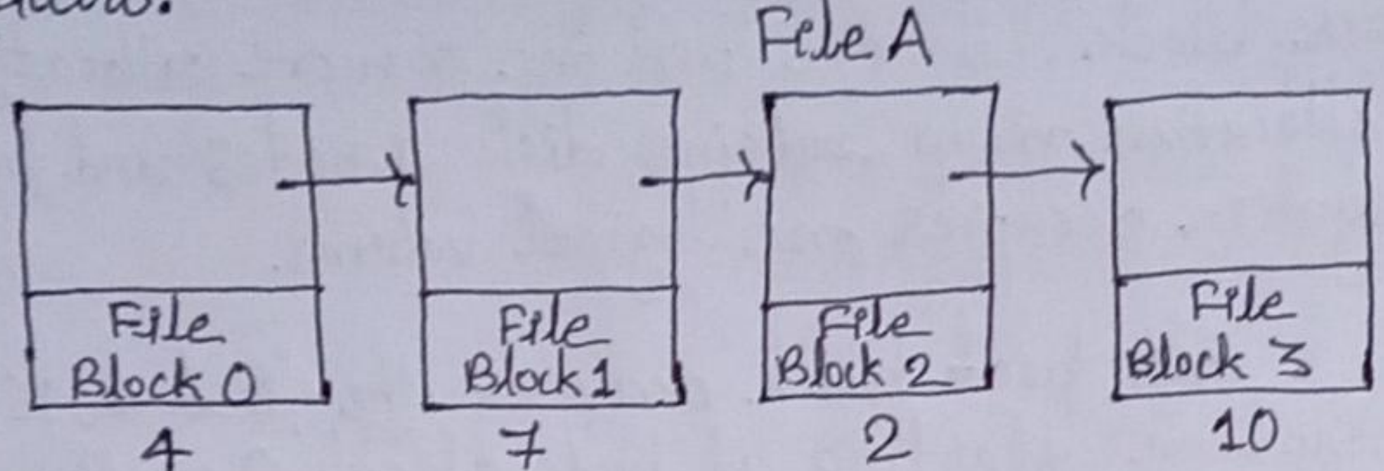


Fig: Storing file as a linked list of disk blocks.

Unlike contigious allocation, every disk block can be used in this method. Except for internal fragmentation in the last block no space is lost to disk fragmentation. To get to block n, the operating system has to start at the beginning and read the n-1 blocks prior to it, one at a time.

Problems: It solves all the problems of contigious allocation but it can be used only for sequential file access, random access is excessively slow. It also requires space for pointer. Each block access requires disk seek.

c) Linked-List Allocation using FAT (File Allocation Table / Table in Memory):

Using this organization, the entire block is available for data. The FAT is used as a linked list. The directory entry contains the block number of the first block of the file. The FAT is looked to find the next block until a special end of file value is reached.

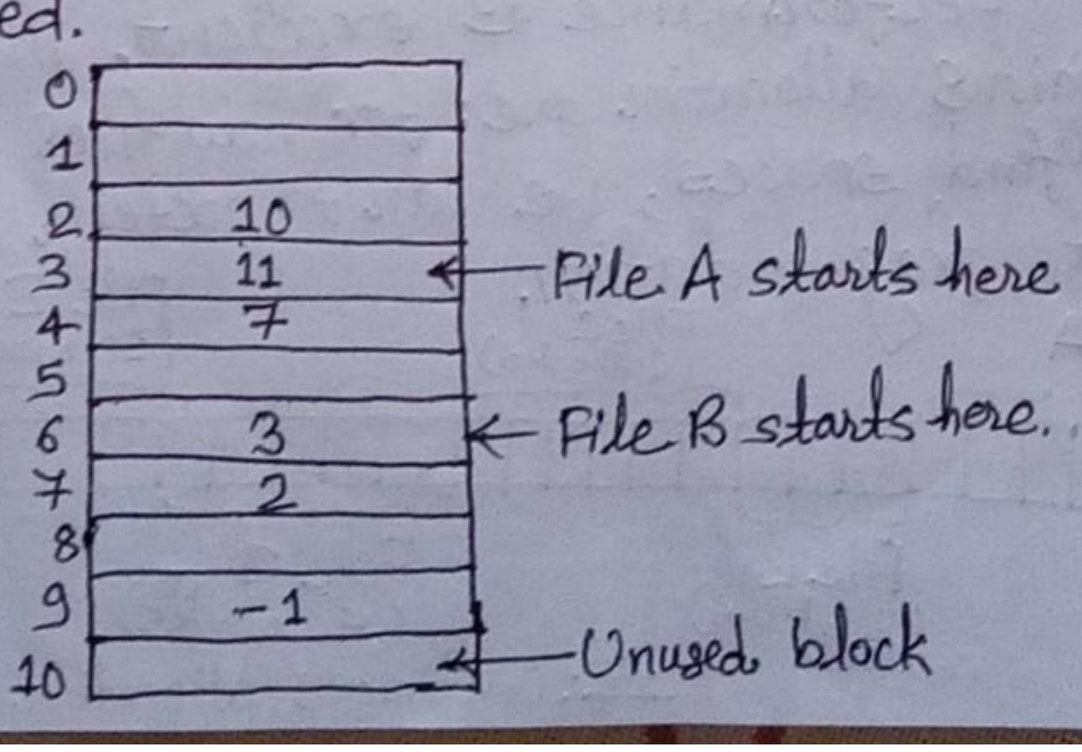


fig. Linked-list allocation using FAT.

Advantages:
→ Entire block is available for data.
→ Result the significant number of disk seek.
→ Random access time is included.

Problems: The entire table must be in memory all the time to make it work.
With a 1-TB disk and a 1-KB block size, the table needs 1 billion entries, one for each of the 1 billion disk blocks. Each entry has to be a minimum of 3 bytes. For speed in lookup, they should be 4 bytes. Thus the table will take up 3 GiB or 2.4 GiB of main memory all the time. Clearly the FAT idea does not scale well to large disks.

d) I-nodes: The method for keeping track of which blocks belong to which file is to associate with each file a data structure called an i-node (index-node), which lists the attributes and disk addresses of the file's blocks.

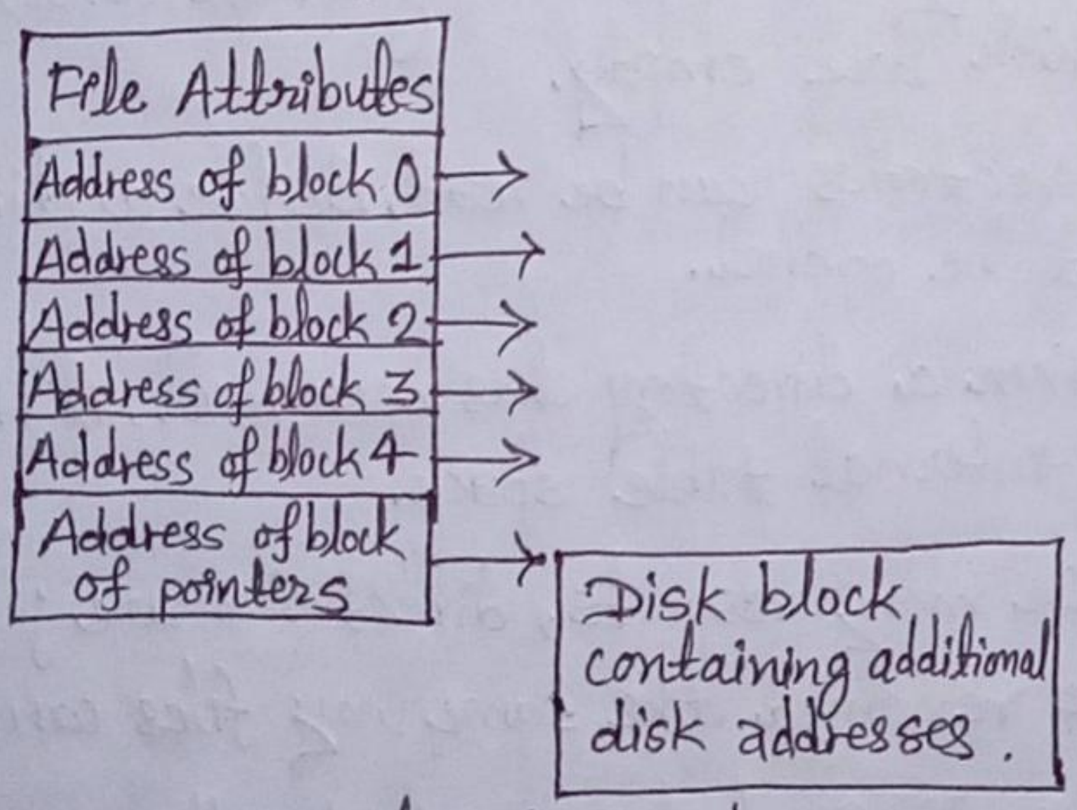


Fig: An example of i-node.

This array is usually far smaller than the space occupied by the file table. The i-node scheme requires an array in memory ~~size to~~ whose size is proportional to the maximum number of files that may be open at once. It does not matter if the disk is 100 GB, 1000 GB, or 10,000 GB.

Advantages:
→ This supports direct access to the blocks occupied by file so it provides fast access to file blocks.
→ It overcomes the problem of external fragmentation.

Disadvantages:

→ The pointer overhead for indexd allocation is greater than linked-list allocation.

→ For very small files, the indexd allocation would keep one entire block for the pointers which is inefficient in terms of memory utilization.

3) Directory:

Directory is a place/location where a set of file(s) will be stored. It is a folder which contains details about files, file size and time when they are created and last modified. To keep track of files, file systems normally have directories or folders which are themselves files.

Directory Operations:-

i) Create → A directory is created. It is empty except for dot and dotdot, which are put there automatically by the system.

ii) Delete → A directory is deleted. Only those directory can be deleted which are empty.

iii) Opendir → Directories can be read. Before a directory can be read, it must be opened.

iv) Closedir → When a directory has been read, it should be closed to free up internal table space.

v) Rename → In many respects, directories are just like files and can be renamed the same way files can be.

vi) Link → Linking is a technique that allows a file to appear in more than one directory.

vii) Unlink → A directory entry is removed.

viii) Readdir → This call returns the next entry in an open directory.

Path names: Each file and directory can be reached by a unique path, known as path name. The path name specifies the location of a file or directory within the file system. Path names cannot exceed 1023 characters in length. Following two different methods are commonly used for path names:

1) Absolute path name → In this method, each file is given an absolute path name consisting of the path from root directory to the file. Absolute path names always start at the root directory and are unique. Some programs need to access a specific file without regard to what the working directory is. In this case, we should always use absolute path names.

In UNIX the components of the path are separated by '/'. In Windows, the seperator is '\'.

Windows \usr\ast\mailbox.

UNIX /usr/ast/mailbox.

2) Relative path name → This is used in conjuction with the concept of working directory. A user can designate one directory as the current working directory, in which all path names not belonging at the root directory are taken relative to the working directory.

Directory Implementation:

There are number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system. The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are widely used.

1) Linear List → In this algorithm, all the files in the directory are maintained as singly linked list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics:

→ When a new file is created, then the entire list is checked whether the new file name is matching to a existing 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.

→ The list needs to be traversed in case of every operation (creation, deletion, updating etc.) on the files therefore the systems become inefficient.

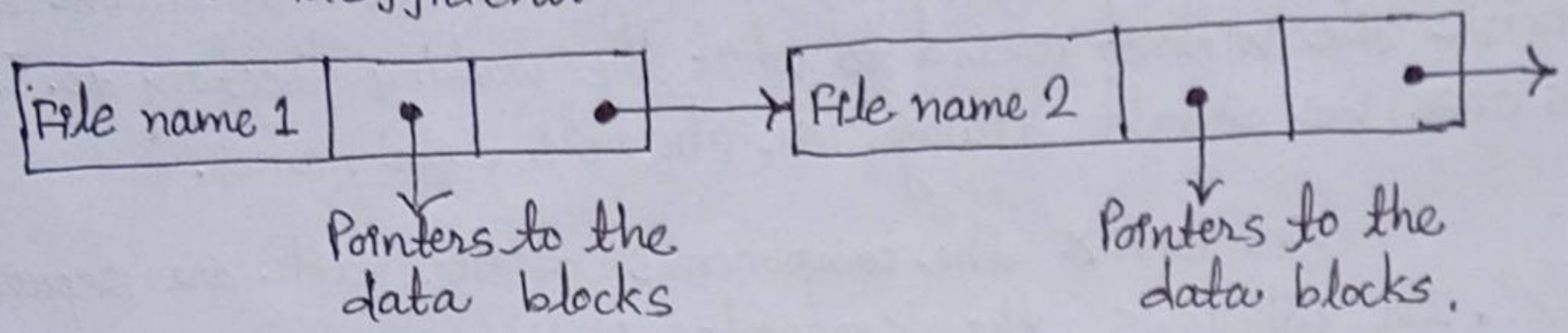
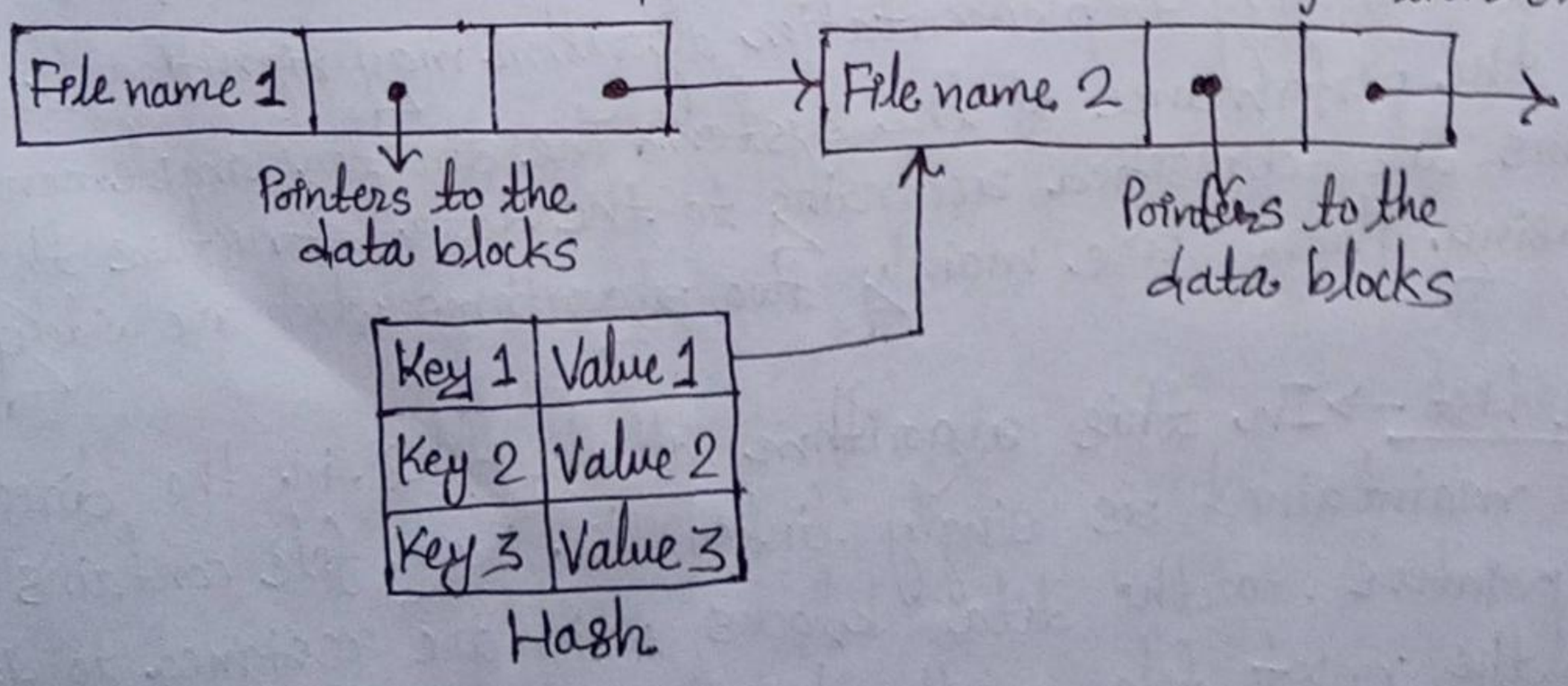


Fig: linear list.

⇒ 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 it greatly decrease the file search time which is its advantage. The problem with this method is that it has fixed size and dependence of the hash function on that size.



Shared Files:

When several users are working together on a project, they often need to share files. It is often convenient for a shared file to appear simultaneously in different directories belonging to different users. Thus it is better to represent file system by using directed acyclic graph (DAG) rather than tree structure as shown figure below;

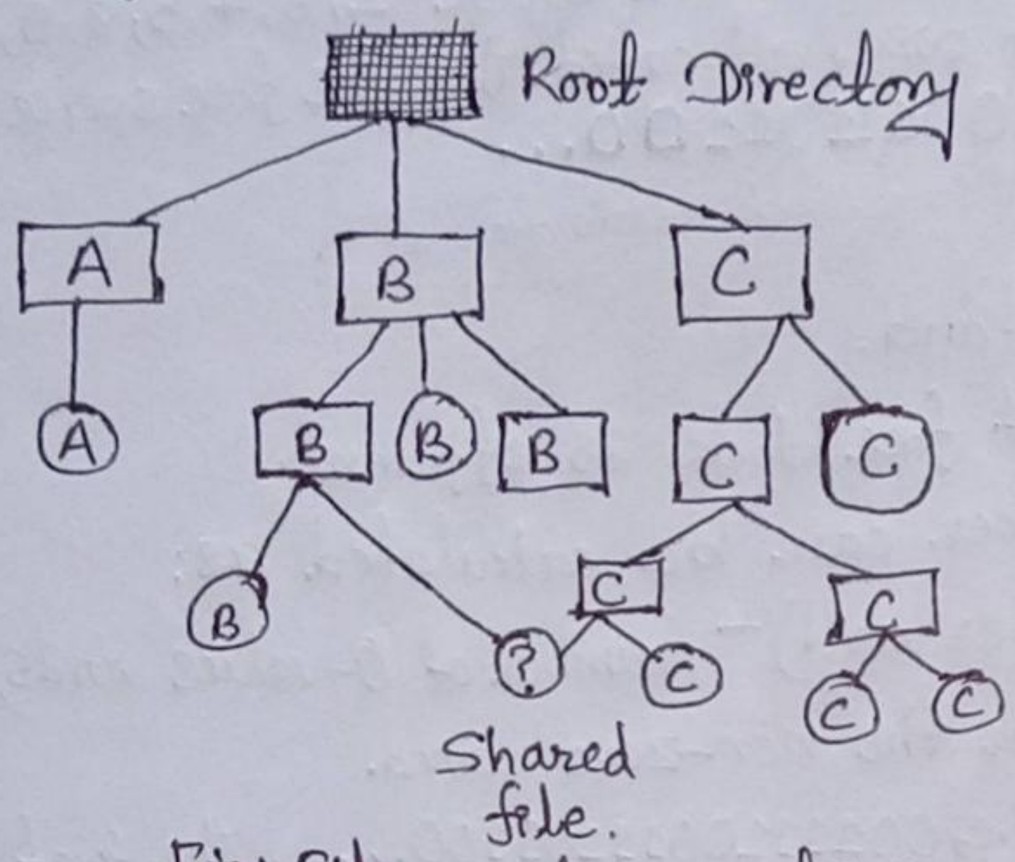


Fig: File system containing a shared file.

It is not good idea to make a copy of a file is being shared. If directories contain disk addresses problem may occur in synchronizing the change made to the file by multiple users. If one user appends new block in the file new block will be listed only in the directory of the user doing the append. Following two approaches can be used to solve this problem.

i) Directory entry that only points to i-nodes.
ii) Directory entry that points to link file.

4) Free Space Management:

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes very important. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly by Bitmaps and linked lists.

i) **Bitmap 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, where 0 indicates that the block is allocated and 1 indicates a free block. Initially all the blocks are empty therefore each bit in the bit map vector contains 1. A disk with n blocks requires a bitmap with n bits.

Example: Consider a disk where block 2,3,4,5,8,9,10,12,13... are free, and rest are allocated. Then the free space bit map would be 0011110011101100...

(Since block starts from 0)

Advantages:

→ Simple to understand.

→ Finding the first free block is efficient.

Note: The block number can be calculated as:

(number of bits per word) * (number of 0-values words) + offset of bit first 1 in the non-zero word.

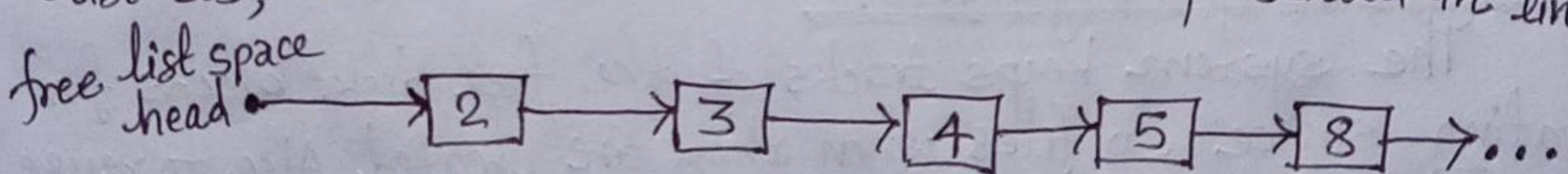
(A 0-valued word has all bits 0)

⇒ From above bitmap i.e, 0011110011101100..., The first group of 8 bits (00111100) constitute a non-zero word since all bits are not 0. Scanning from first 3rd bit is non-zero so, offset = 3.

Therefore, the first free block number = 8*0 + 3
= 3.

ii) **Linked List** → 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.

Example: Consider a disk where block 2,3,4,5,8 are free and rest are allocated then it can be represented in linked list as;



Advantage: Only one block is kept in the memory

Problem: Not efficient to traverse list, it must reach each block.