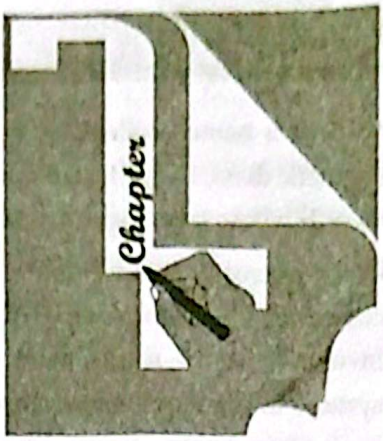


FILE SYSTEM INTERFACE MANAGEMENT

CHAPTER OUTLINE

After comprehensive study of this chapter, you will be able to:

- File Concept: File naming, File Structure, File Type, File Access, File Attributes, File Operation and File Descriptors;
- Directories: Single-Level Directory system, Hierarchical Directory Systems, Path names, Directory Operations;
- Access Methods: Sequential, Direct; Protection: Types of Access, Access Control List, Access Control Matrix.

FILE OVERVIEW

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

In the Microsoft Windows family of operating systems, users are presented with several different choices of file systems when formatting such media. These choices depend on the type of media involved and the situations in which the media is being formatted. The two most common file systems in Windows are as follows:

- NTFS
- FAT
- exFAT
- HFS Plus
- EXT

File systems can differ between operating systems (OS), such as Microsoft Windows, MacOS and Linux-based systems. Some file systems are designed for specific applications. Major types of file systems include distributed file systems, disk-based file systems and special purpose file systems.

A file system stores and organizes data and can be thought of as a type of index for all the data contained in a storage device. These devices can include hard drives, optical drives and flash drives. File systems specify conventions for naming files, including the maximum number of characters in a name, which characters can be used and, in some systems, how long the file name suffix can be. In many file systems, file names are not case sensitive.

FILE NAMING

When a process creates a file, it gives the file name; while process terminates, the file continue to exist and can be accessed by other processes. A file is named, for the convenience of its human users, and it is referred to by its name. A name is string of characters. The string may be of digits or special characters (e.g. 2,!,% etc.). Some system differentiates between the upper and lower case character, whereas other system considers the equivalent (like UNIX and MS-DOS). Normally the string of max 8 characters are legal file name (e.g., in DOS), but many recent system support as long as 255 characters (e.g. Windows 2000).

Many OS support two-part file names; separated by period; the part following the period is called the file extension and usually indicates something about the file (e.g., file.c C programming source file). But in some system it may have two or more extension such as in UNIXproc.c.Z (C Programming source file compressed using Ziv-Lampel algorithm. In some system e.g. UNIX, file extension are just conventions; in other system it requires (e.g., C compiler must requires .c source file).

The name of each file must be unique within the directory where it is stored. This ensures that the file also has a unique path name in the file system. File naming guidelines are:

- A file name can be up to 255 characters long and can contain letters, numbers, and underscores.
- The operating system is case-sensitive, which means it distinguishes between uppercase and lowercase letters in file names. Therefore, FILEA, FiLea, and file a are three distinct file names, even if they reside in the same directory.
- File names should be as descriptive and meaningful as possible.
- Directories follow the same naming conventions as files.
- Certain characters have special meaning to the operating system. Avoid using these characters when you are naming files. These characters include the following:

  / \ " ' * ; - ? [ ] ( ) ~ ! $ { } <> # @ & | space tab newline
- A file name is hidden from a normal directory listing if it begins with a dot (.). When the ls command is entered with the -a flag, the hidden files are listed along with regular files and directories.

Here, the table given below lists the most common file extensions with their meaning:

File Extension	File Meaning
myfile.bak	This indicates backup file
myfile.c	This indicates C programming language source file
myfile.gif	This indicates gif format image file
myfile.hlp	This indicates help file
myfile.html	This indicates Hyper Text Markup Language (HTML) file
myfile.jpg	This indicates jpg format image file
myfile.mp3	This indicates mp3 music or audio file in which music encoded in MPEG layer 3 audio format
myfile.mpg	This indicates mpg video file in which movie encoded with the MPEG standard
myfile.o	This indicates an object file
myfile.pdf	This indicates Portable Document Format (PDF) file
myfile.ps	This indicates PostScript file
myfile.tex	This indicates input for the TEX formatting program
myfile.txt	This indicates normal textual file
myfile.zip	This indicates compressed archive

BASIC FILE OPERATIONS

There are many file operations that can be perform by the computer system some of them are listed below:

- **Create:** find space for the file and make an entry in the directory.
- **Open:** find file and determine if it has already been opened. If not open search directory, cache information, add entry in per-process open-file table. If open check lock and cache information if lock can be acquired. Increment the open count.
- **Close:** decrement the open count and remove the file's entry from the open-file table if count reaches zero.
- **Read:** read data from the file.
- **Write:** write data to the file.
- **Delete:** search directory, release file space and erase directory entry.
- **Reposition:** reposition the file position pointer. This is more commonly known as seek.
- **Truncate:** delete content of a file, but keep file properties.
- **Lock:** file locks provide concurrency control. A shared lock allows multiple readers to acquire a lock concurrently, while exclusive lock ensures only one writer can modify a file. With mandatory locking the operating system ensures locking integrity, while with advisory locking the application process ensures that the correct locking strategy is followed. The Windows operating system uses the mandatory locking strategy.

FILE STRUCTURE

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. UNIX, MS-DOS support minimum number of file structure.

Files can be structured in several ways in which three common structures are given below:

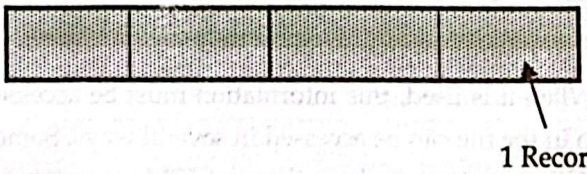
- Unstructured sequence of bytes
- Sequence of fixed-length records
- Tree of records

Byte Sequence

The file in Fig below is just an unstructured sequence of bytes. In effect, the operating system does not know or care what is in the file. All it sees are bytes. Any meaning must be imposed by user-level programs. Both UNIX and Windows 98 use this approach.

Record Sequence

In this model, a file is a sequence of fixed-length records, each with some internal structure. Central to the idea of a file being a sequence of records is the idea that the read operation returns one record and the write operation overwrites or appends one record. As a historical note, when the 80-column punched card was king many (mainframe) operating systems based their file systems on files consisting of 80-character records, in effect, card images.



Tree

In this organization, a file consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record. The tree is sorted on the key field, to allow rapid searching for a particular key.

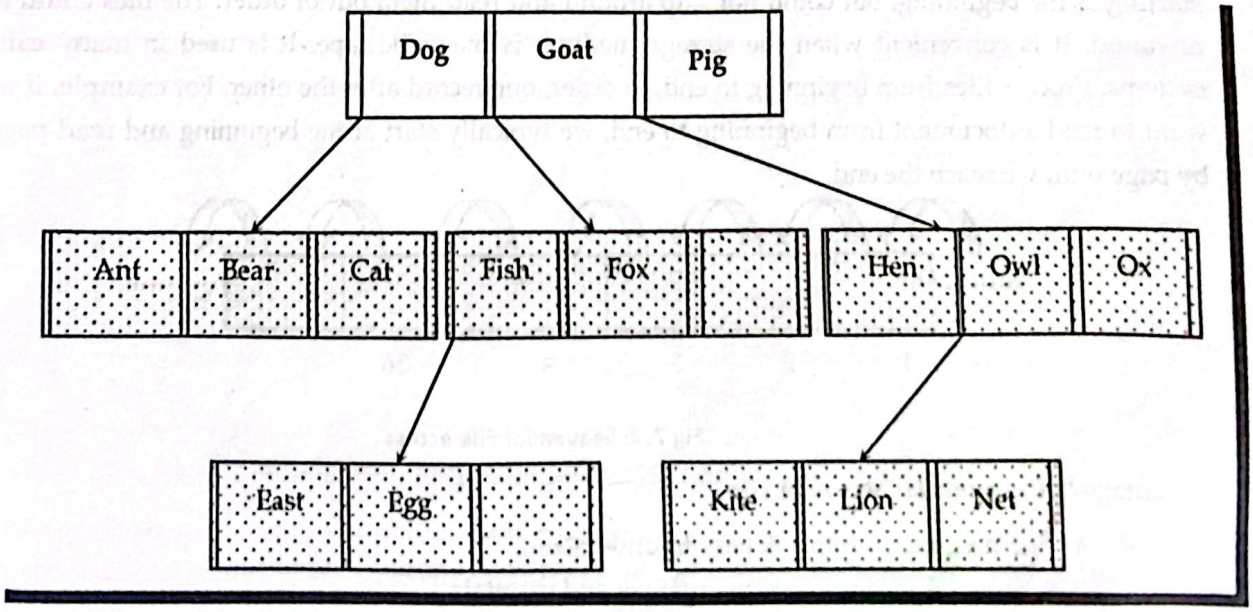


Fig 7.1: Tree structure of file

FILE TYPES

Many OS supports several types of files

- **Regular files:** contains user information, are generally ASCII or binary.
- **Directories:** system files for maintaining the structure of file system.
- **Character Special files:** related to I/O and used to model serial I/O devices such as terminals, printers, and networks.
- **Block special files:** used to model disks
- **ASCII files:** Consists of line of text. Each line is terminated either by carriage return character or by line feed character or both. They can be displayed and printed as is and can be edited with ordinary text editor.
- **Binary files:** Consists of sequence of byte only. They have some internal structure known to programs that use them (e.g., executable or archive files).Many OS use extension to identify the file types; but UNIX like OS use a magic number to identify file types.

FILE ACCESS

Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways. Some systems provide only one access method for files. Other system, such as those of IBM, supports many access methods, and choosing the right one for a particular application is a major design problem. The access methods are Sequential and Direct access

Sequential Access

The simplest access method; Information in the file is processed in order, one record after the other starting at the beginning, but could not skip around and read them out of order. The files could be rewound. It is convenient when the storage medium is magnetic tape. It is used in many early systems. Process files from beginning to end, in order, one record after the other. For example, if we want to read a document from beginning to end, we typically start at the beginning and read page, by page until we reach the end.

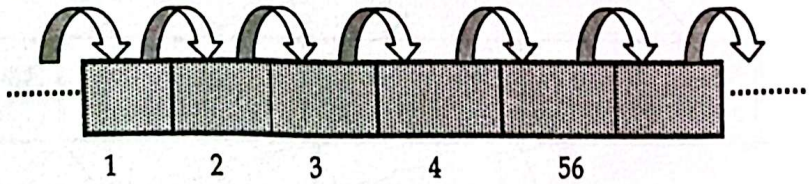


Fig 7.2: Sequential File access

Advantages of sequential access of file

- The method is simple & easy to understand.
- Sequential files are easy to organize and maintain.
- Loading or reading a record requires only the Record Key.
- It is efficient & economical if the number of file records to be processed is high.

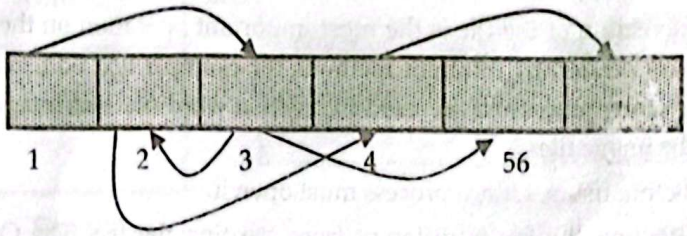
- Relatively inexpensive Input/output media and devices may be used.
- Errors in the files remain localized.

Disadvantages of sequential file organization

- The sorting does not remove the need to access other records as the search looks for particular records.
- Sequential records cannot support modern technologies that require fast access to stored records.
- The requirement that all records be of the same size is sometimes difficult to enforce.

Direct Access

Files whose bytes or records can be read in any order are called direct access. It is based on disk model of file, since disks allow random access to any block. It is used for immediate access to large amounts of information. When a query concerning a particular subject arrives, we compute which block contain the answer, and then read that block directly to provide desired information. Here process file by accessing the content in no specific order. For example, if we only want to read page 1013 it makes sense to reposition (seek) to page 1013 and read the page.



Advantages of random file access

- Quick retrieval of records.
- The records can be of different sizes.

Disadvantages of Random file organization

- Data may be accidentally erased or overwritten unless special precautions are taken.
- Random files are less efficient in the use of storage space compared to sequentially organized files.
- Expensive hardware and software resources are required.
- Relatively complex when programming
- System design based on random file organization is complex and costly

FILE ATTRIBUTES

File attributes are settings associated with computer files that grant or deny certain rights to how a user or the operating system can access that file. For example, IBM compatible computers running MS-DOS or Microsoft Windows have capabilities of having read, archive, system, and hidden attributes. Following are some of the attributes of a file:

- **Name:** It is the only information which is in human-readable form.
- **Identifier.** The file is identified by a unique tag (number) within file system.
- **Type.** It is needed for systems that support different types of files.
- **Location.** Pointer to file location on device.
- **Size.** The current size of the file.
- **Protection.** This controls and assigns the power of reading, writing, executing.
- **Time, date, and user identification.** This is the data for protection, security, and usage monitoring.
- **Read-only:** Allows a file to be read, but nothing can be written to the file or changed.
- **Archive:** Tells Windows Backup to backup the file.
- **System:** System file.
- **Hidden:** File will not be shown when doing a regular dir from DOS.

FILE OPERATIONS

OS provides system calls to perform operations on files. Some common calls are:

- **Create:** Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.
- **Open:** Before using a file, a process must open it.
- **Write:** Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which, the data needs to be written.
- **Read:** Every file is opened in three different modes: Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.
- **Re-position:** Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as seeking.
- **Delete:** Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.
- **Truncate:** Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.
- **Close:** When all access is finished, the file should be closed to free up the internal table space.
- **Append:** Add data at the end of the file.
- **Seek:** Repositions the file pointer to a specific place in the file.
- **Get attributes:** Returns file attributes for processing.
- **Set attributes:** To set the user settable attributes when file changed.
- **Rename:** Rename file.

DIRECTORY STRUCTURE

A directory is a node containing information about files. It is also called folder. Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.

To get the benefit of different file systems on the different operating systems, a hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks. Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.

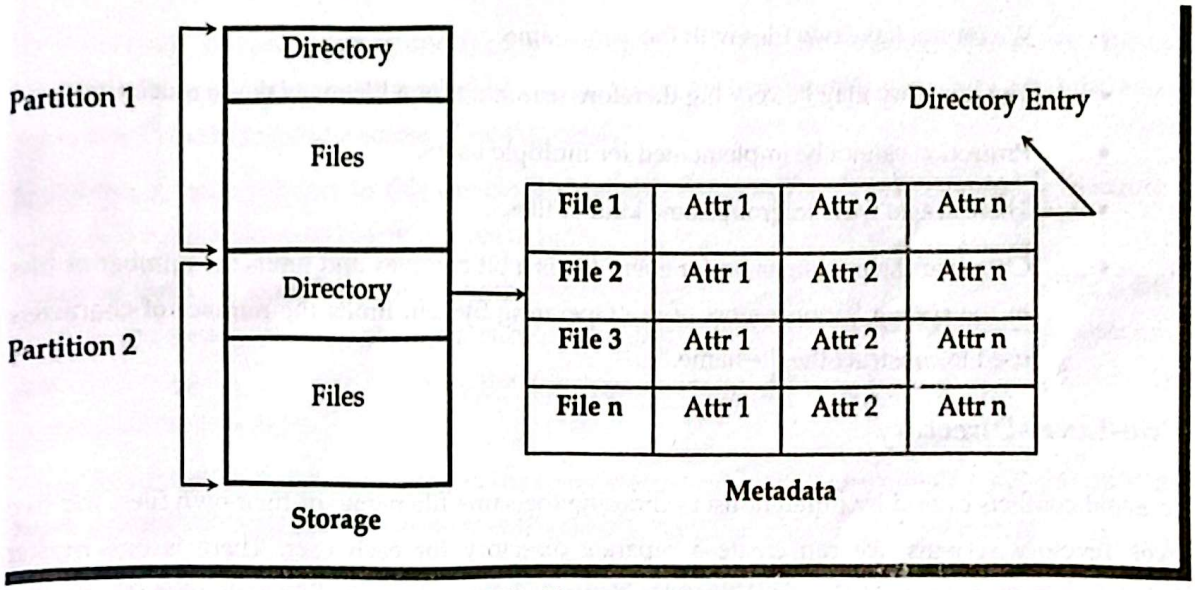


Fig 7.3: Directory structure

Directories can have different structures.

Single-Level-Directory

The simplest form of directory system is having one directory containing all the files. It is also called root directory. The entire system will contain only one directory which is supposed to mention all the files present in the file system. The directory contains one entry per each file present on the file system. It is easy to support and understand; but difficult to manage large amount of files and to manage different users. An example of a system with one directory is given below.

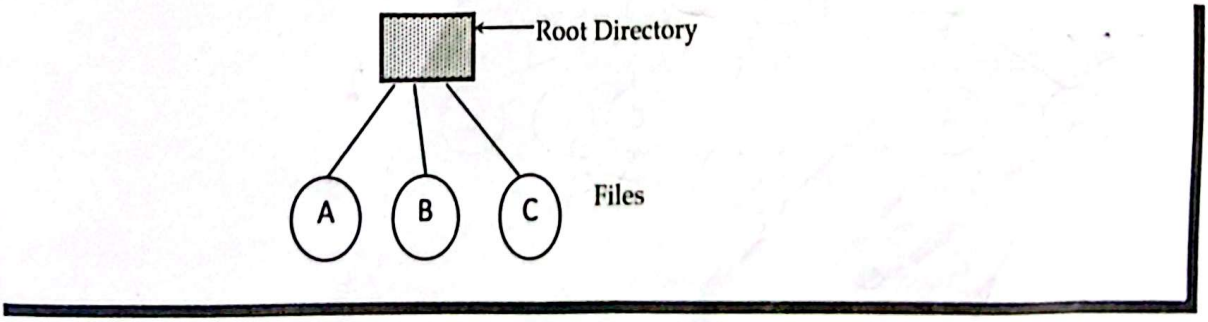


Fig 7.4: A single level directory containing different files

The problem with having only one directory in a system with multiple users is that different users may accidentally use the same names for their files.For examples, if A creates a file name called Ram, and then later user B also creates a file name called Ram, B's file will overwrite A's file. Consequently, this scheme is not used on multiuser system.

Advantages

- Implementation is very simple.
- If the sizes of the files are very small then the searching becomes faster.
- File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

- We cannot have two files with the same name.
- The directory may be very big therefore searching for a file may take so much time.
- Protection cannot be implemented for multiple users.
- There are no ways to group same kind of files.
- Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

Two-Level-Directory

To avoid conflicts caused by different users choosing the same file name for their own files, the two level directory systems, we can create a separate directory for each user. There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file. The system doesn't let a user to enter in the other user's directory without permission.

The example of this system is shown below.

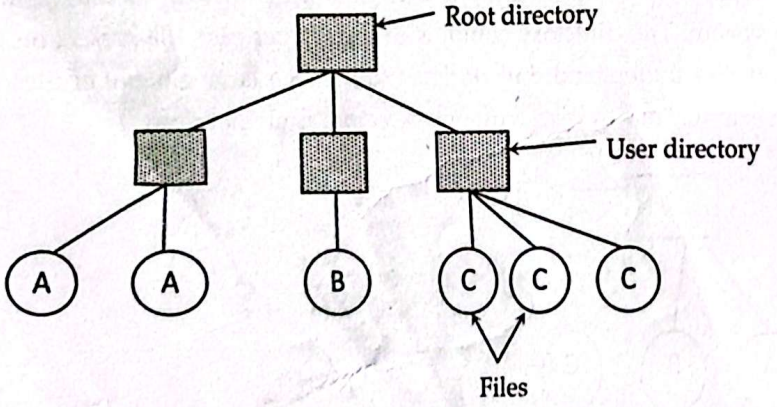


Fig 7.5: A two-level directory system

Characteristics of two level directory systems

- Each files has a path name as /User-name/directory-name/
- Different users can have the same file name.
- Searching becomes more efficient as only one user's list needs to be traversed.
- The same kind of files cannot be grouped into a single directory for a particular user.

Hierarchical-Directory

In Tree structured directory system, any directory entry can either be a file or sub directory. Tree structured directory system overcomes the drawbacks of two level directory system. The similar kind of files can now be grouped in one directory.

Each user has its own directory and it cannot enter in the other user's directory. However, the user has the permission to read the root's data but he cannot write or modify this. Only administrator of the system has the complete access of root directory.

Searching is more efficient in this directory structure. The concept of current working directory is used. A file can be accessed by two types of path, either relative or absolute.

Absolute path is the path of the file with respect to the root directory of the system while relative path is the path with respect to the current working directory of the system. In tree structured directory systems, the user is given the privilege to create the files as well as directories. This approach is shown below.

Here, the directory A, B, C contained in the root directory each belong to different user, two of whom have created subdirectories for projects they are working on.

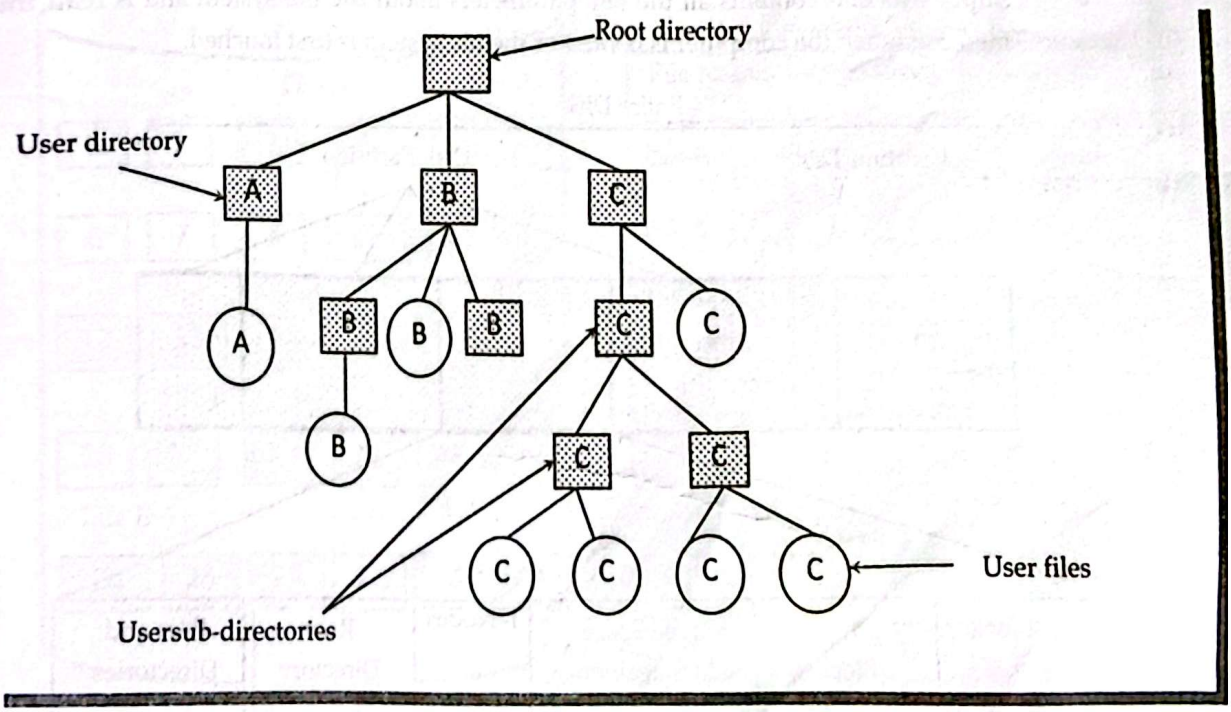


Fig 7.6: A hierarchical directory system.

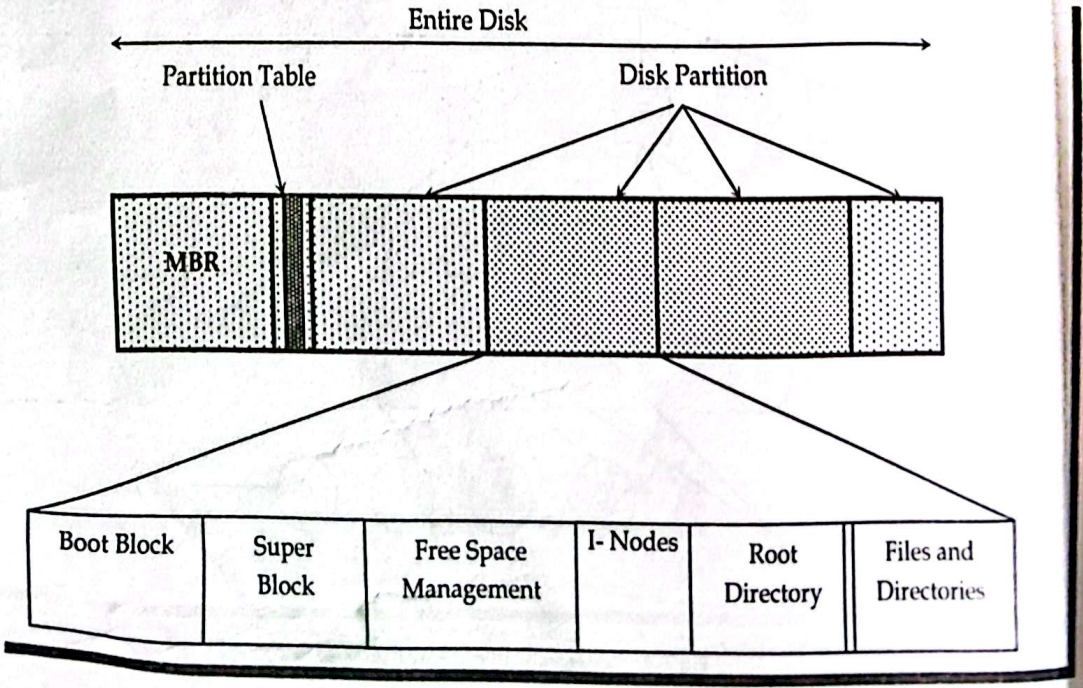
File System Layout

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 superblock, bitmaps, and one or more allocation groups. An allocation group contains disk i-nodes and fragments. Each file system occupies one logical volume.

Basically, file systems are stored on the disks. Almost all disks can be divided up into multiple partitions with independent file systems on each partition. Here, in the partition of the disk, Sector 0 is called as Master Boot Record (MBR), is used to boot the computer system. The Master Boot Record's end contains the partition table. That partition table gives the starting and ending addresses of each partition of the disk. From those partitions in the table, one is marked as active. So that, whenever the computer system is booted up, the BIOS read in and execute the Master Boot Record.

The very first thing that the master boot record program does is, locate the active partition, read in its first block that is called as the boot block and execute it. Now the program present inside the boot block loads the OS that contained in that partition. For the purpose of uniformity, each and every partition starts with a boot block, even if it doesn't contain a bootable OS. File Systems are stored on disks. The figure below depicts a possible File-System Layout.

- **MBR:** Master Boot Record is used to boot the computer
- **Partition Table:** Partition table is present at the end of MBR. This table gives the starting and ending addresses of each partition.
- **Boot Block:** When the computer is booted, the BIOS read in and execute the MBR. The first thing the MBR program does is locate the active partition, read in its first block, which is called the boot block, and execute it. The program in the boot block loads the operating system contained in that partition. Every partition contains a boot block at the beginning though it does not contain a bootable operating system.
- **Super Block:** It contains all the key parameters about the file system and is read into memory when the computer is booted or the file system is first touched.



IMPLEMENTING FILES

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

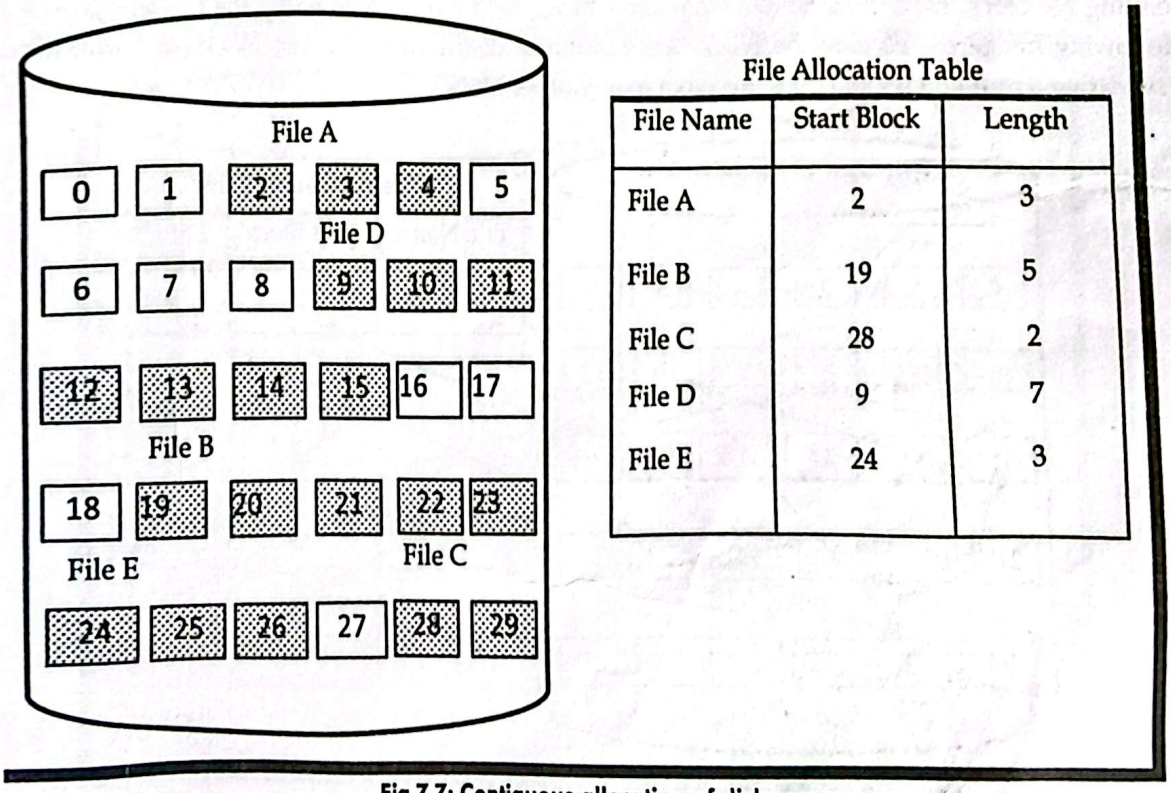
Implementing Files by Contiguous Allocation

A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing.

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,.....b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The file B in the following figure starts from the block 19 with length = 5 blocks. Therefore, it occupies 19, 20, 21, 22, 23 blocks.



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	19	5
File C	28	2
File D	9	7
File E	24	3

Fig 7.7: Contiguous allocation of disk space

Advantages

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

Disadvantages

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Compaction algorithm will be necessary to free up additional space on disk.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.
- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

Implementing Files by Linked List Allocation

Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. Increase in file size is always possible if free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of file.

In this scheme, each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file. The file-A in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.

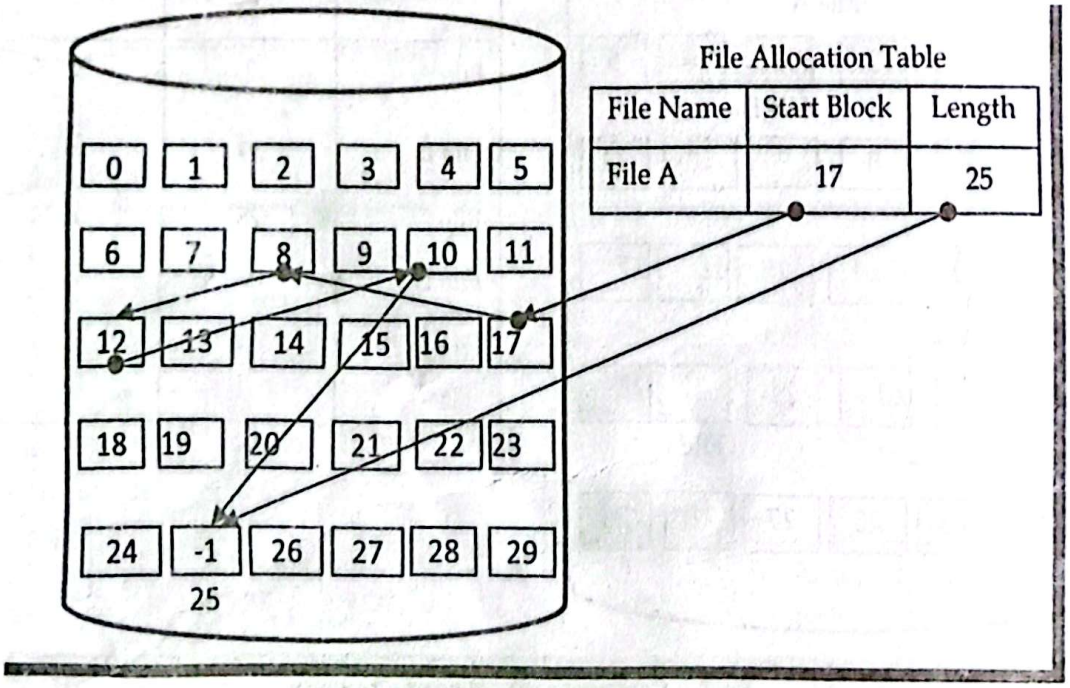


Fig 7.8: Linked list allocation of disk space

Advantages

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

Disadvantages

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

Index Allocation

It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. Allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block. The i^{th} entry in the index block contains the disk address of the i^{th} file block. The directory entry contains the address of the index block as shown in the image:

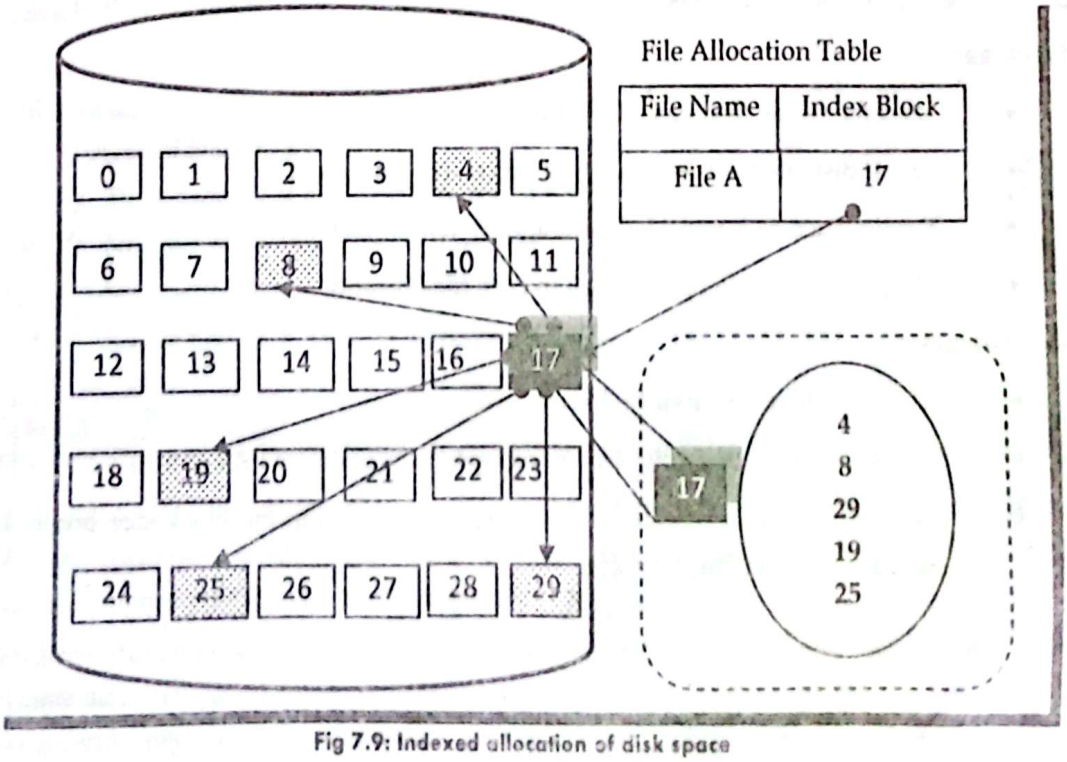


Fig 7.9: Indexed allocation of disk space

Advantages

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

Disadvantages

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

LINKED LIST ALLOCATION USING TABLE IN MEMORY

The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks. File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number. File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block.

It simply accesses the file allocation table, read the desired block entry from there and access that block. This is the way by which the random access is accomplished by using FAT. It is used by MS-DOS and pre-NT Windows versions.

Advantages

- Uses the whole disk block for data.
- A bad disk block doesn't cause all successive blocks lost.
- Random access is provided although it's not too fast.
- Only FAT needs to be traversed in each file operation.

Disadvantages

- Each Disk block needs a FAT entry.
- FAT size may be very big depending upon the number of FAT entries.
- Number of FAT entries can be reduced by increasing the block size but it will also increase Internal Fragmentation.

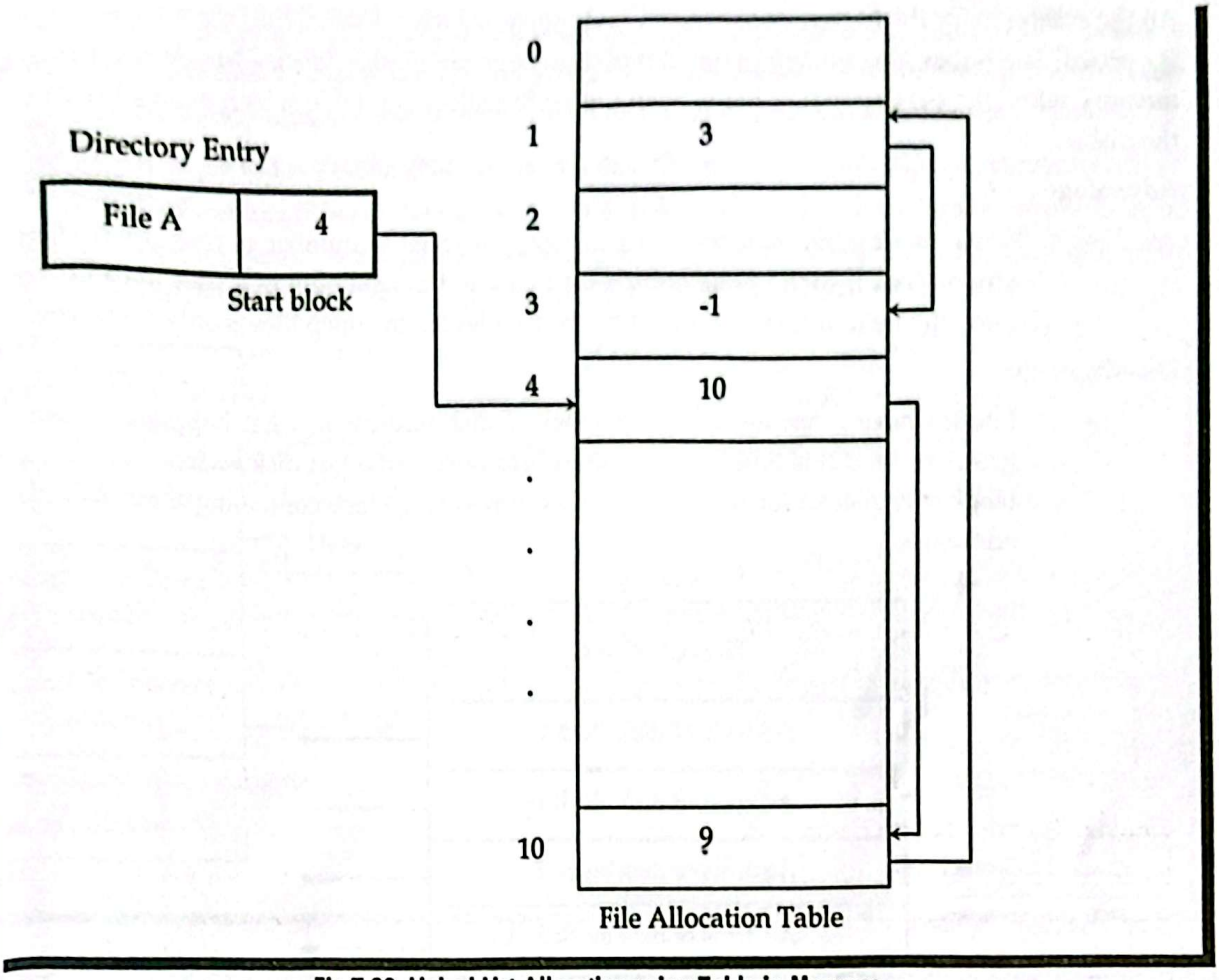


Fig 7.20: Linked List Allocation using Table in Memory

Example: Consider a disk of 100 KB. If block size is 2 KB, calculate the size of FAT assuming that each entry in FAT takes 4 bytes.

Solution:

Size of disk=100 GB =100 x 2^{20} KB = 104857600 KB

Size of block=2 KB

Thus, number of blocks=104857600/2 = 52428800

⇨ Number of Entries in FAT = 52428800

Since size of an entry= 4 byte

⇨ Size of FAT (File Allocation Table)= 52428800 x 4 byte = 200 MB

I-NODES

I-node (Index node) is a data structure which is used to identify which block belongs to which file. It contains the attributes and disk addresses of the file's blocks. Unlike the in-memory table the i-node need to be in memory only when the corresponding file is open.

List the attributes and disk address of the block. Which block belongs to which file, associate each file with a data structure called i-node. It is possible with i-node to find all the blocks of the file. The big advantage with this schema is that only i-node is in memory when its corresponding file is open.

All the attributes for the file are stored in an I-node entry, which is loaded into memory when the file is opened. The I-node also contains a number of direct pointers to disc blocks. I-node need only be in memory when the corresponding file is open unlike file allocation table which grows linearly with the disk.

Advantage

- Space needed bin memory is directly proportional to number of files opened not the size of disk. If each i-node occupies n bytes and a maximum of k files may be open at once, the total memory occupied by the i-nodes for the open files is only kn bytes.

Disadvantage

- I-nodes have room for a fixed number of disk addresses, what happens when a file grows beyond this limit? One solution is to reserve the last disk address not for a data block, but instead for the address of a address of a block containing more disks block addresses.

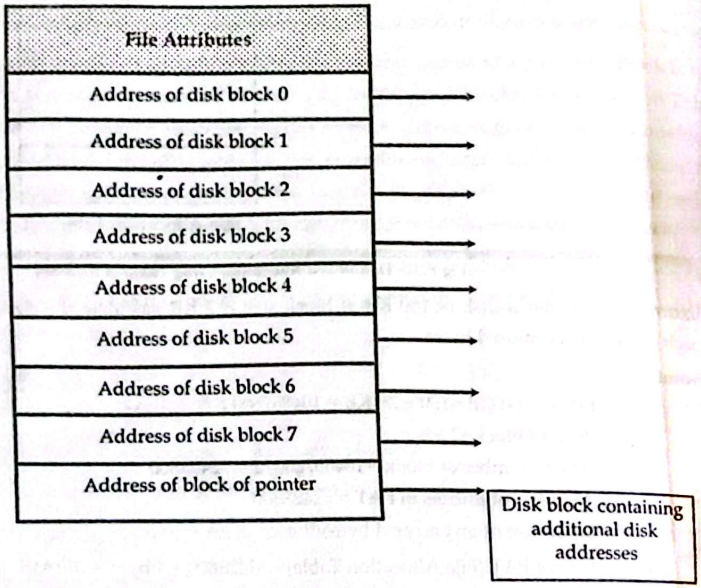
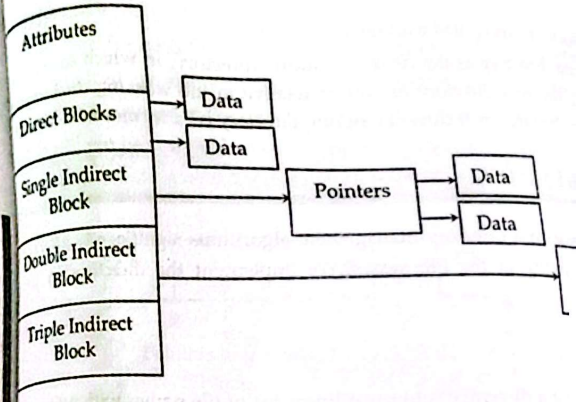


Fig 7.21: I-Nodes

In UNIX based operating systems, each file is indexed by an I-node. I-node is the special disk block which is created with the creation of the file system. The number of files or directories in a file system depends on the number of I-nodes in the file system. An I-node includes the following information

- Attributes (permissions, time stamp, ownership details, etc.) of the file
- A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.

- A double indirect pointer which points to a disk block that is
to the disk blocks which are index blocks. Double index poir
big to be indexed entirely by the direct blocks as well as the
- A triple index pointer that points to a disk block that is a co
the pointers is separately pointing to a disk block which
pointers which are separately pointing to an index block
the file blocks.



IRECTORY OPERATIONS

e allowed system calls for managing directories exhibit mor
stem calls for files. The following are the some of the key di

Create: A directory is created. It is empty except fo
automatically by the system

Delete: A directory is deleted. Only an empty directc

Opendir: Directory can be read. For example, to lis
opens the directory to read out the names of all the

Closedir: When a directory has been read, it shoul

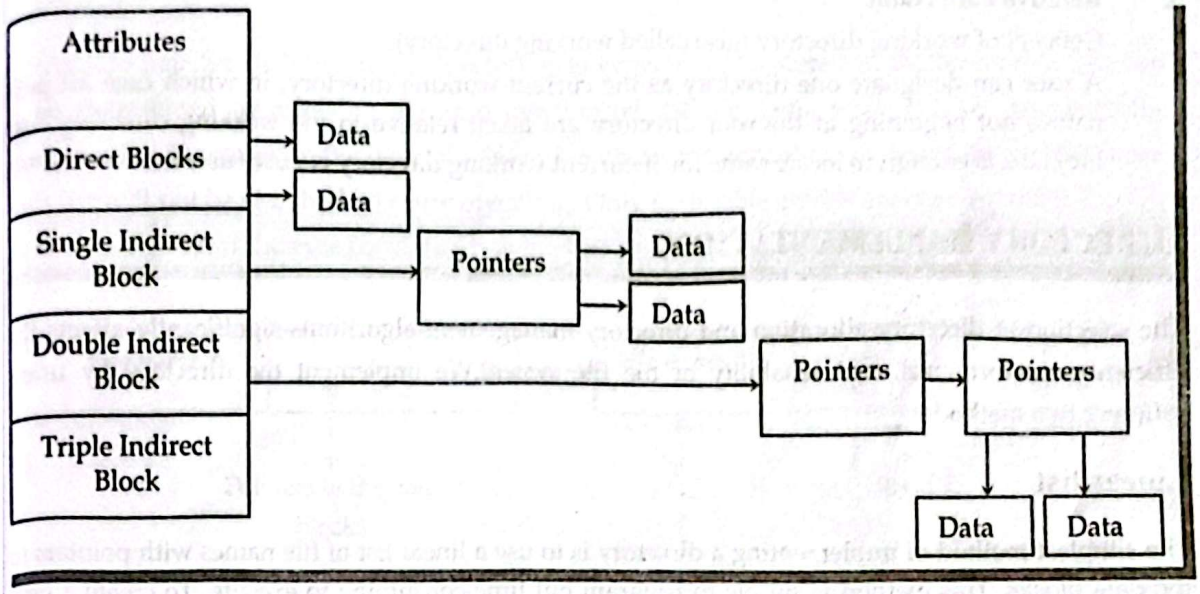
Readdir: This call returns the next entry in an ope
directories using the usable read system call, b
forcing the programmer to know and deal with t

Rename: In many respects, directories are just li
can be.

Link: Linking is the technique that allows a file

Unlink: A directory entry is removed. If t
directory, it is removed from the file system.

- A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
- A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file blocks.



DIRECTORY OPERATIONS

The allowed system calls for managing directories exhibit more variation from system to system then system calls for files. The following are the some of the key directory operations:

1. **Create:** A directory is created. It is empty except for dot and dot dot, which are put there automatically by the system
2. **Delete:** A directory is deleted. Only an empty directory can be deleted.
3. **Opendir:** Directory can be read. For example, to list all files in a directory, a listing program opens the directory to read out the names of all the files it contains.
4. **Closedir:** When a directory has been read, it should be closed to free up internal table space.
5. **Readdir:** This call returns the next entry in an open directory. Formally, it was possible to read directories using the usable read system call, but that approach has the disadvantage for forcing the programmer to know and deal with the internal structure of the directories.
6. **Rename:** In many respects, directories are just like files and can be renamed the same way file can be.
7. **Link:** Linking is the technique that allows a file to appear in more than one directory.
8. **Unlink:** A directory entry is removed. If the file being unlinked is only present in one directory, it is removed from the file system.

PATH NAMES

When the file system is organized as a directory tree, some way is needed for specifying file names. Two different methods are commonly used. These are **absolute** and **relative** path names.

1. **Absolute Path Name**
   The path name starting from root directory to the file. E.g. In UNIX: /usr/user1/bin/lab2.Path separated by / in UNIX and \ in windows.
2. **Relative Path Name**
   Concept of working directory (also called working directory).
   A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory.E.g., bin/lab2 is enough to locate same file if current working directory is /usr/user1.

DIRECTORY IMPLEMENTATION

The selection of directory-allocation and directory-management algorithms significantly affects the efficiency, performance, and reliability of the file system.We implement the directory by using different two methods.

Linear list

The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute. To create a new file, we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory. To delete a file, we search the directory for the named file, then release the space allocated to it.The real disadvantage of a linear list of directory entries is that finding a file requires a linear search. Directory information is used frequently, and users will notice if access to it is slow. In fact, many operating systems implement a software cache to store the most recently used directory information. A cache hit avoids the need to constantly reread the information from disk. A sorted list allows a binary search and decreases the average search time.

Characteristics

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

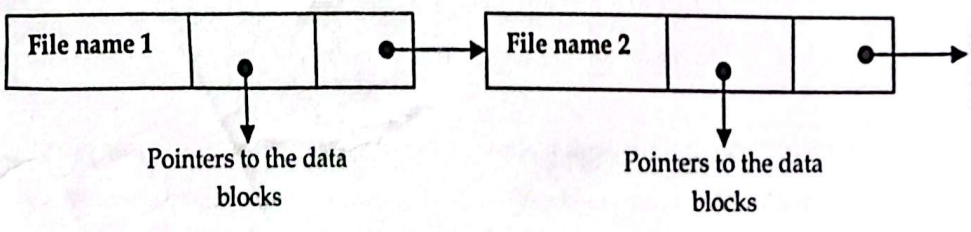


Fig 7.23: Linear List

Hash Table

Another data structure that has been used for a file directory is a hash table. It consist linear list with hash table. The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time. Insertion and deletion are also fairly straightforward, although some provision must be made for collisions-situations where two file names hash to the same location. If that key is already in use, a linked list is constructed.

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.

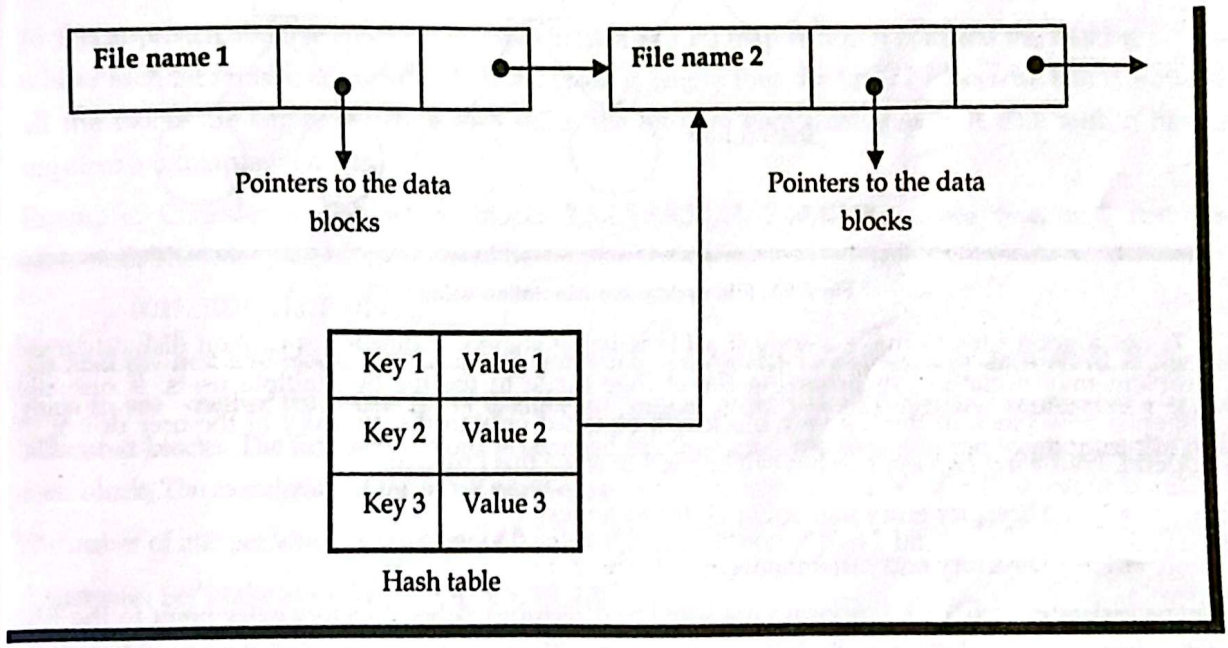


Fig 7.24: Hash table

Advantages: greatly decrease the file search time.

Problem: It greatly fixed size and dependence of the hash function on that size.

SHARED FILES

In many situations we need to share files between users. For example, 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 in figure below;

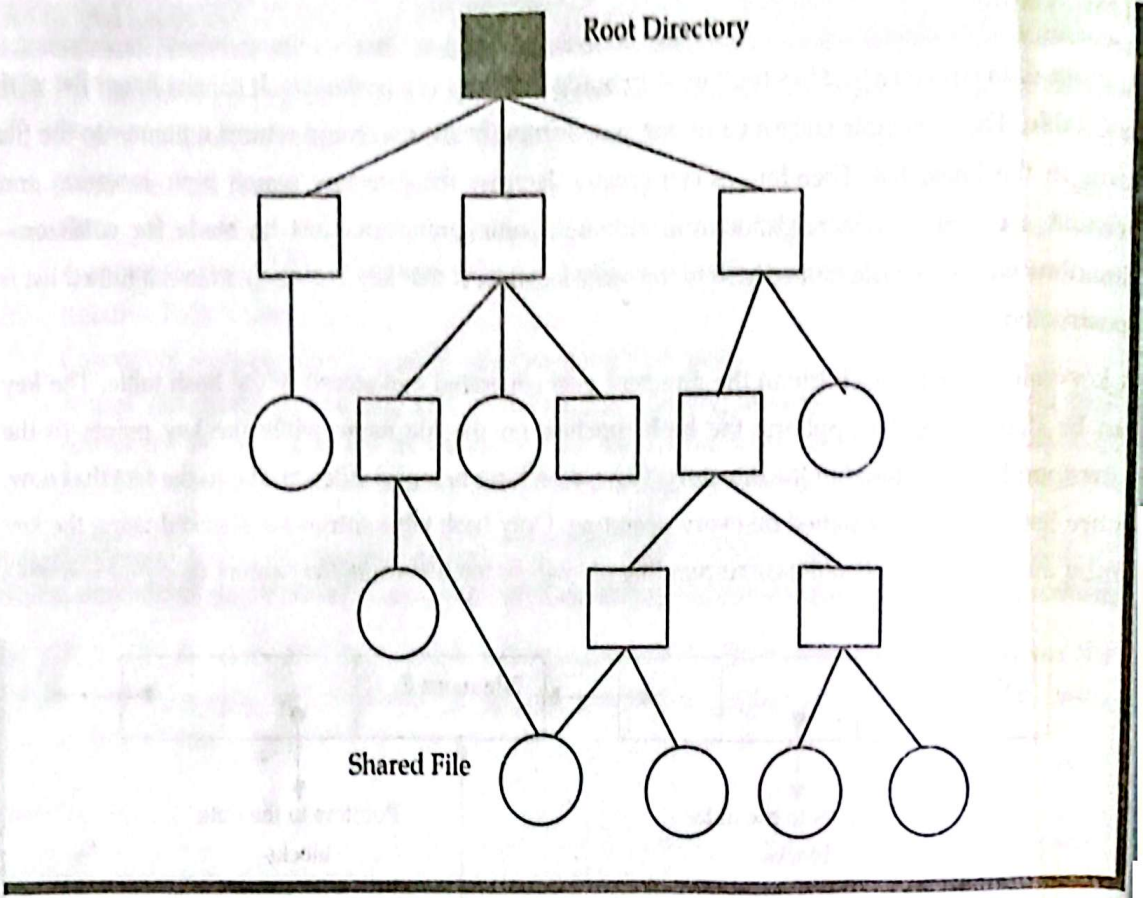


Fig 7.25: File system representation using DAG

It is not a good idea to make a copy if 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.

- Directory entry that only points to i-nodes
- Directory entry that points to link file

In the first approach, disk blocks are not listed in directories rather directory entry point to the little data structure (i-node in UNIX) associated with the file itself. Creating a link does not change the ownership, but it does increase the link count in the i-node. Main problem with this approach is that if owner fo the shared file tries to remove the file, the system is faced with a problem. If it removes the file and clears the i-node, another user may have a directory entry pointing to an invalid i-node. One solution to this is to only remove owner's directory entry. But again if the system has quotas, owner will continue to be billed for the file until another user decides a remove it.

In second approach, when another user B want to share the file owned by used C then system creates a link file and then maker entry of the link file in B's directory. Link file contains just the path name of the file to which it is linked. When B reads from the linked file, the operating system sees that the file being read from is of type link, looks up the name of the file, and reads that file. This approach is called symbolic linking. Extra overhead is the main problem associated with this approach. The file containing the path must be read, and then the path must be parsed and followed, component by component, until the I-node is reached. All of this activity may require a considerable number of extra disk accesses.

FREE SPACE MANAGEMENT

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free space list. The free space list records all free disk blocks those not allocated to some file or directory. A file system is responsible to allocate the free blocks to the file therefore it has to keep track of all the free blocks present in the disk. Just as the space that is allocated to files must be managed, so the space that is not currently allocated to any file must be managed. To perform any of the file allocation techniques, it is necessary to know what blocks on the disk are available. Thus we need a disk allocation table in addition to a file allocation table. There are mainly two approaches by using which, the free blocks in the disk are managed.

- Bit Vector and
- Linked List

Bitmaps Free Space Management

In this approach, the free space list is implemented as a bit map vector. It contains the number of bits where each bit represents each block. If the block is empty then the bit is 1 otherwise it is 0. 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 blocks 2,3,4,5,8,9,10,11,12,13,17,18,..... are free, and rest are allocated. The free space bit map would be

0011110011111100010.....

To find the first free block, the Macintosh operating system checks sequentially each word in the bit map to see whether that value is not 0, since a 0 valued word has all 0 bits and represents a set of allocated blocks. The first non 0 word is scanned for the first 1 bit, which is the location of the first free block. The calculation of the block number is

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

Example: Let's take block list {2, 3, 4, 5, 9, 10, 13}

Solution:

Blocks →	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Bits →	0	0	1	1	1	1	0	0	0	1	1	0	0	1

Advantages

Simple and efficient in finding first free block, or n consecutive free blocks.

Problems

Inefficient unless the entire bitmap is kept in main memory. Keeping in main memory is possible only for small disk, when disk is large the bitmap would be large. Many system use bitmap method.

Linked List Free Space Management

It is another approach for free space management. This approach suggests linking together all the free blocks and keeping a pointer in the cache which points to the first free block. Therefore, all the free blocks on the disks will be linked together with a pointer. Whenever a block gets allocated, its previous free block will be linked to its next free block. We would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block. 3, which would point to block 4, which would point to block 5, which would point to block 8, However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial 1/0 time. Fortunately, traversing the free list is not a frequent action. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used. The FAT method incorporates free-block accounting into the allocation data structure. No separate method is needed.

Advantages

Only one block is kept in memory.

Problems

Not efficient; to traverse list, it must read each block. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used. The FAT method incorporates free block accounting into the allocation data structure. No separate method is needed.

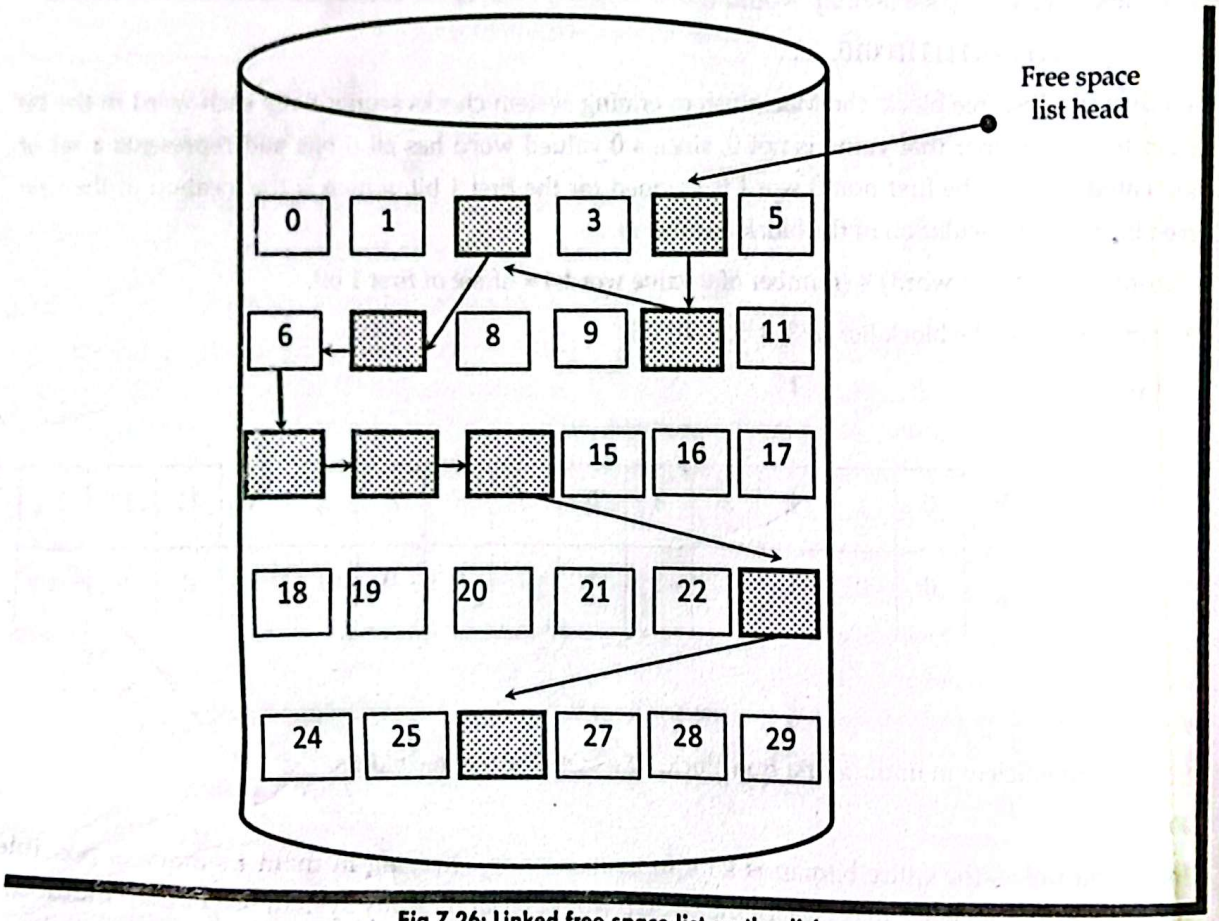


Fig 7.26: Linked free space list on the disk

Problem 1: Consider a 40 MB disc with 2 K blocks. Calculate the number of blocks needed to hold the disc bitmap. If we require 16-bit to hold a disc block number (i.e. disk block number range from 0-65535). What will be number of blocks needed by linked list of free blocks?

Answer: Case for Bitmap

Disk size = 40 MB = 40 x 1024 KB

⇨ Number of blocks= (40 x 1024)/2= 20 x 1024

Since every block needs 1- bit in bitmap

⇨ Size of Bitmap=20 x 1024 bit = 2560 byte = 2.5 KB

Since block size is 2 KB

⇨ 2 blocks are used to hold the bitmap

For Case of Free list

Block size = 2K= 2 x 1024 byte

Number of bits needed to store block number = 16 bit = 2 byte

Thus, number of blocks that can be stored in a block = (2 x 1024)/2= 1024

Since one of the addresses is used to store address of the next block in the free list

⇨ Number of blocks that can be stored in a block = 1024-1= 1023

We know that,

Size of disk=40 MB = 40 x 1024 KB

Since, size of block= 2 K

⇨ Number of blocks in disc= (40 x 1024)/2 = 20 x 1024

Thus, number of blocks needed to store free list= (20 x 1024)/1023 = 20.019 = 20

Problem 2: Assume you have an i-node-based file system. The file system has 512 byte blocks. Each i-node has 10 direct, 1 single indirect, 1 double indirect and 1 triple indirect block pointers. Block pointers are 4 bytes each. Assume the i-node and any block free list is always in memory. Blocks are not cached.

i) What is the maximum file size that can be stored before?
   a) The single indirect pointer is needed?
   b) The double indirect pointer is needed?
   c) The triple indirect pointer is needed?

ii) What is the maximum file size supported?

iii) What is the number of disk block reads required to read 1 byte from a file
   a) In the best case?
   b) In the worst case?

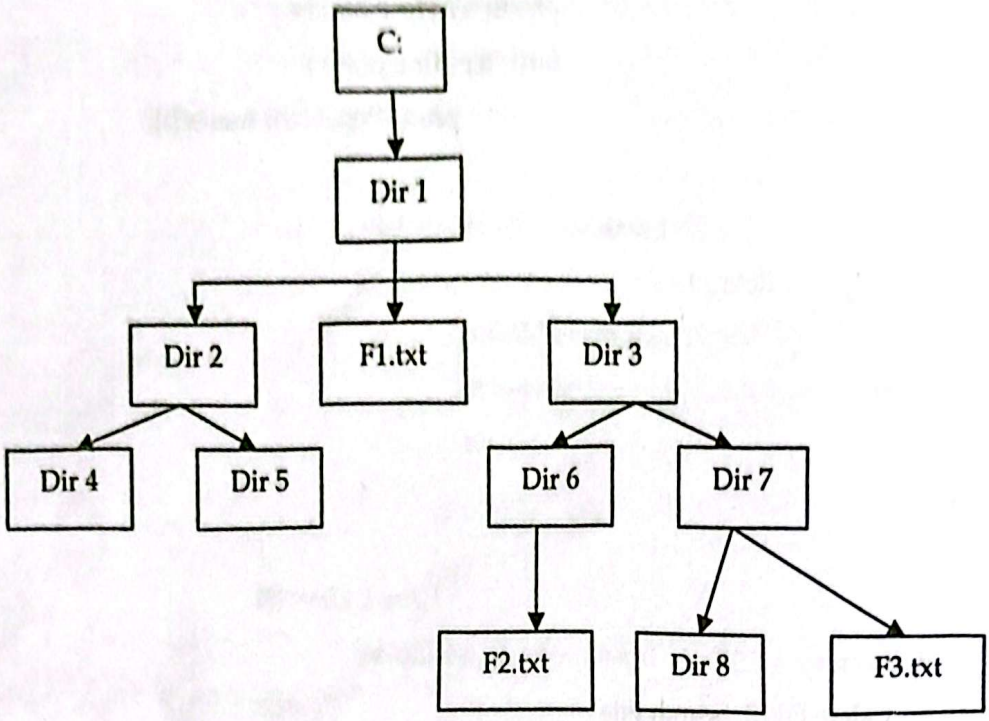
Answer:

i) a) 10 × 512 bytes = 5 KB
   b) 10 × 512 + (512 / 4) × 512 bytes = 69 KB
   c) 10 × 512 + (512 / 4) × 512 + (512 / 4)^2 × 512 bytes ≈ 8 MB

ii) 10 × 512 + (512 / 4) × 512 + (512 / 4) ^{2} × 512 + (512 / 4) ^{3} × 512 bytes ≈ 1 GB

iii) a) 1
   b) 4

Program 4: Create the files and directories as following structure:



1. Copy the F3.txt to dir 9 and F1.txt to dir 8
2. Rename the file c:\dir1\dir3\dir6\F2.txt to F.txt and c:\dir1\dir3\dir7\F3.txt to F3.txt
3. Delete the file F1.txt
4. Delete the directory dir 9

EXERCISE

Multiple Choice Questions

1. _____ is a unique tag, usually a number identifies the file within the file system.
   a) File identifier
   b) File name
   c) File type
   d) None of the mentioned
2. To create a file _____
   a) Allocate the space in file system
   b) Make an entry for new file in directory
   c) Allocate the space in file system & make an entry for new file in directory
   d) None of the mentioned

4. File type can be represented by _____

a) File name
b) file extension
c) File identifier
d) none of the mentioned

What is the mounting of file system?

a) Crating of a file system
b) Deleting a file system
c) Attaching portion of the file system into a directory structure
d) Removing the portion of the file system into a directory structure

Once the changes are written to the log, they are considered to be _____

a) Committed
b) aborted
c) Completed
d) none of the mentioned

When an entire committed transaction is completed, _____

a) It is stored in the memory
b) it is removed from the log file
c) It is redone
d) none of the mentioned

The larger the block size, the _____ the internal fragmentation.

a) Greater
b) lesser
c) Same
d) none of the mentioned

For a direct access file _____

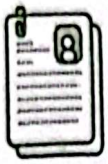
a) There are restrictions on the order of reading and writing
b) There are no restrictions on the order of reading and writing
c) Access is restricted permission wise
d) Access is not restricted permission wise

What will happen in the single level directory?

a) All files are contained in different directories all at the same level
b) All files are contained in the same directory
c) Depends on the operating system
d) None of the mentioned

0. What will happen in the two level directory structure?

a) Each user has his/her own user file directory
b) The system doesn't its own master file directory
c) All of the mentioned
d) None of the mentioned



Subjective Questions

1. What is a file? Write down their importance in OS.
2. What are the typical operations performed on files?
3. What are File Control Blocks? Explain their structure with suitable example.
4. What are file types? Explain types of file used in OS.
5. How is free space managed? Explain free space management techniques in detail.
6. Consider a file system where a file can be deleted and its disk space Reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?
7. What are the advantages and disadvantages of a system providing mandatory locks instead of providing advisory locks whose usage is left to the users' discretion?
8. What are the advantages and disadvantages of recording the name of the creating program with the file's attributes (as is done in the Macintosh Operating System)?
9. If the operating system were to know that a certain application is going to access the file data in a sequential manner, how could it exploit this information to improve performance?
10. Give an example of an application that could benefit from operating system support for random access to indexed files.
11. Discuss the merits and demerits of supporting links to files that cross mount points (that is, the file link refers to a file that is stored in a different volume).
12. Discuss the advantages and disadvantages of associating with remote file systems (stored on file servers) a different set of failure semantics from that associated with local file systems.
13. Give an example of a Directory implementation that could benefit from operating system.
14. What is Hierarchical Directory Systems? Explain with suitable example.
15. What does Contiguous Allocation requires? Write down their advantages and disadvantages.
16. Suppose tree-directory structure and acyclic-graph directory structure is implemented in two different file systems to organize files and directories. Which one would be better and why?
17. What is virtual file system (VFS)? How multiple file systems are handled by virtual file system?
18. What is indexed-allocation method? Is multilevel indexed allocation is a better solution for applications that need files with very large size? Explain your answer.
19. Why access protection is necessary in file systems? How protection can be implemented by the identity of the users?
20. Compare linked file allocation method with indexed allocation method.

ANSWERS KEY

1. (a)	2. (c)	3. (b)	4. (c)	5. (a)	6. (b)	7. (a)	8. (b)	9. (b)	10. (a)

❑❑❑