cylinder satisfying the waiting requests in order of their locations. When it reaches the outermost cylinder it sweeps back to the innermost cylinder without satisfying any requests and then starts again.

6.8.3 LOOK

Similarly to SCAN, the drive sweeps across the surface of the disk, satisfying requests, in alternating directions. However the drive now makes use of the information it has about the locations requested by the waiting requests. For example, a sweep out towards the outer edge of the disk will be reversed when there are no waiting requests for locations beyond the current cylinder.

6.8.4 Circular LOOK (C-LOOK)

Based on C-SCAN, C-LOOK involves the drive head sweeping across the disk satisfying requests in one direction only. As in LOOK the drive makes use of the location of waiting requests in order to determine how far to continue a sweep, and where to commence the next sweep. Thus it may curtail a sweep towards the outer edge when there are locations requested in cylinders beyond the current position, and commence its next sweep at a cylinder which is not the innermost one, if that is the most central one for which a sector is currently requested.

6.9 DISK MANAGEMENT

The hard disk is the secondary storage device that is used in the computer system. Usually the primary memory is used for the booting up of the computer. But a hard disk drive is necessary in the computer system since it needs to store the operating system that is used to store the information of the devices and the management of the user data.

The management of the IO devices that is the Input Output devices, like the printer and the other peripherals like the keyboard and the etc; all require the usage of the operating system. Hence the information of the all such devices and the management of the system are done by the operating system. The operating system works as an interpreter between the machine and the user.

The operating system is a must for the proper functioning of the computer. The computer is a device that needs to be fed with the instructions that are to be carried out and executed. Hence there needs to be an interpreter who is going to carry out the conversions from the high level language of the user to the low level language of the computer machine.

The hard disk drive as secondary memory is therefore needed for the purpose of installing the operating system. If there is no operating system then the question arises where to install the operating system. The operating system obviously cannot be installed in the primary memory however large that may be. The primary memory is also a volatile memory that cannot be used for the permanent storage of the system files of the operating system. The operating system requires the permanent file storage media like the hard disk. More over the hard disk management is an important part of maintaining the computer, since it requires an efficient management of the data or the user information. The information regarding the Master Boot Record is stored in the hard disk drive. This is the information that is required during the start up of the computer. The computer system needs this information for loading the operating system.

The file management and the resources management is also a part of the hard disk management. The hard disk management requires an efficient knowledge of the operating system and its resources and the methods of how these resources can be employed in order to achieve maximum benefit. The operating system contains the resources and the tools that are used to manage the files in the operating system. The partitioning and the installation of the operating system itself may be considered as the hard disk management.

The hard disk management also involves the formatting of the hard disk drive and to check the integrity of the file system. The data redundancy check can also be carried out for the consistency of the hard disk drive. The hard disk drive management is also important in the case of the network where there are many hard disk drives to be managed.

Managing a single hard disk in a single user operating system is quite easy in comparison with the management of the hard disk drives in a multi user operating system where there is more than one user. It is not that much easy since the users are also required to be managed.

Check Your Progress

Fill in the blanks:

1. In computer architecture, the combination of the .... and .... is considered the heart of a computer.
2. I/O devices can be installed into .... of the computer.
3. A hardware device that sends information into the CPU is known as .... device.
4. A .... is the smallest physical storage unit on the disk.
5. The size of a sector is ....

6.10 LET US SUM UP

Hard disk is the secondary storage device that is used in the computer system. It is physically composed of a series of flat, magnetically coated platters stacked on a spindle. The spindle turns while the heads move between the platters, in tandem, radically reading/writing data onto the platters. On a hard disk, data is stored in thin, concentric bands. A drive head, while in one position can read or write a circular ring, or band called a track. Most disks used in personal computers today rotate at a constant angular velocity. The system disk controller reads this data to place the drive heads in the correct sector position. A sector, being the smallest physical storage unit on the disk, is almost always 512 bytes in size because 512 is a power of 2 (2 to the power of 9). There is a trade-off between throughput (the average number of requests satisfied in unit time) and response time (the average time between a request arriving and it being satisfied). Various different disk scheduling policies are there. Various different disk scheduling policies are there. The operating system works as an interpreter between the machine and the user. It keeps the information about all I/O devices and manages them. The file management and the resources management is also a part of the hard disk management.

6.11 KEYWORDS

Input: It is the signal or data received by the system.

Output: It is the signal or data sent from the system.

I/O devices: Hardware, which are used by a person (or other system) to communicate with a computer.

Keyboard: One of the main input devices used on a computer, a PC's keyboard looks very similar to the keyboards of electric typewriters, with some additional keys.

Mouse: An input device that allows an individual to control a mouse pointer in a graphical user interface (GUI).

Scanner: Hardware input device that allows a user to take an image and/or text and convert it into a digital file, allowing the computer to read and/or display the scanned object.

Microphone: A microphone is a hardware peripheral that allows computer users to input audio into their computers.

Web Cam: A camera connected to a computer or server that allows anyone connected to the internet to view still pictures or motion video of a user.

Monitor: Also called a video display terminal (VDT) a monitor is a video display screen and the hard shell that holds it.

Printer: An external hardware device responsible for taking computer data and generating a hard copy of that data.

Sound card: Also known as a sound board or an audio card, a sound card is an expansion card or integrated circuit that provides a computer with the ability to produce sound that can be heard by the user.

Video card: Also known as a graphics card, video card, video board, or a video controller, a video adapter is an internal circuit board that allows a display device, such as a monitor, to display images from the computer.

Hard disk: It is the secondary storage device that is used in the computer system.

Track: Data is stored in thin, concentric bands on the hard disk and drive head while in one position can read or write the band; that band is called a track.

Sector: It is the smallest physical storage unit on the disk and almost always 512 bytes in size.

Cluster: It is a group of one or more consecutive sectors.

Disk management: It is a process by which operating system manages the secondary storage area to maximize the throughput and minimize the response time.

6.12 QUESTIONS FOR DISCUSSIONS

1. What is device driver? How it communicates with the devices?
2. What is device controller? How it differs from device driver?
3. What is Memory-mapped I/O? Describe its role in the I/O system.
4. How a hard disk is physically composed? Describe it with suitable diagram.
5. What is the function of a system disk controller?

Check Your Progress: Model Answers

1. CPU, Main memory
2. Physical slots
3. Input
4. Sector
5. 512 bytes

6.13 SUGGESTED READINGS

Andrew S. Tanenbaum, *Modern Operating System,* Published By Prentice Hall

Silberschatz Galvin, *Operating System Concepts,* Published By Addison Wesley

Andrew M. Lister, *Fundamentals of Operating Systems,* Published By Wiley

Colin Ritchie, *Operating Systems,* Published By BPB Publications

LESSON

7

FILE MANAGEMENT

CONTENTS

7.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- An overview file management systems
- Brief idea about file system architecture and its functions
- How to access a file
- Concept of file sharing, file directory structure and file allocation
- Understand security polices and mechanisms
- Explain protection and access control

7.1 INTRODUCTION

Another part of the operating system is the file manager. While the memory manager is responsible for the maintenance of primary memory, the file manager is responsible for the maintenance of secondary storage (e.g., hard disks). Each file is a named collection of data stored in a device. The file manager implements this abstraction and provides directories for organizing files. It also provides a spectrum of commands to read and write the contents of a file, to set the file read/write position, to set and use the protection mechanism, to change the ownership, to list files in a directory, and to remove a file. The file manager provides a protection mechanism to allow machine users to administer how processes executing on behalf of different users can access the information in files. File protection is a fundamental property of files because it allows different people to store their information on a shared computer, with the confidence that the information can be kept confidential.

In addition to these functions, the file manager also provides a logical way for users to organize files in secondary storage.

7.2 FILES

A file is a collection of letters, numbers and special characters: it may be a program, a database, a dissertation, a reading list, a simple letter etc. Sometimes you may import a file from elsewhere, for example from another computer. If you want to enter your own text or data, you will start by creating a file. Whether you copied a file from elsewhere or created your own, you will need to return to it later in order to edit its contents.

The most familiar file systems make use of an underlying data storage device that offers access to an array of fixed-size blocks, sometimes called sector, generally 512 bytes each. The file system software is responsible for organizing these sectors into files and directories, and keeping track of which sectors belong to which file and which are not being used. Most file systems address data in fixed-sized units called "clusters" or "blocks" which contain a certain number of disk sectors (usually 1-64). This is the smallest logical amount of disk space that can be allocated to hold a file.

However, file systems need not make use of a storage device at all. A file system can be used to organize and represent access to any data, whether it be stored or dynamically generated (e.g, from a network connection).

Whether the file system has an underlying storage device or not, file systems typically have directories which associate file names with files, usually by connecting the file name to an index into a file allocation table of some sort, such as the FAT in an MS-DOS file system, or an inode in a Unix-like file system. Directory structures may be flat, or allow hierarchies where directories may contain subdirectories. In some file systems, file names are structured, with special syntax for filename extensions and version numbers. In others, file names are simple strings, and per-file metadata is stored elsewhere.

Other bookkeeping information is typically associated with each file within a file system. The length of the data contained in a file may be stored as the number of blocks allocated for the file or as an exact byte count. The time that the file was last modified may be stored as the file's timestamp. Some file systems also store the file creation time, the time it was last accessed, and the time that the file's meta-data was changed. (Note that many early PC operating systems did not keep track of file times.) Other information can include the file's device type (e.g., block, character, socket, subdirectory, etc.), its owner user-ID and group-ID, and its access permission settings (e.g., whether the file is read-only, executable, etc.).

The hierarchical file system was an early research interest of Dennis Ritchie of Unix fame; previous implementations were restricted to only a few levels, notably the IBM implementations, even of their early databases like IMS. After the success of Unix, Ritchie extended the file system concept to every object in his later operating system developments, such as Plan 9 and Inferno.

Traditional file systems offer facilities to create, move and delete both files and directories. They lack facilities to create additional links to a directory (hard links in Unix), rename parent links (".." in Unix-like OS), and create bidirectional links to files.

Traditional file systems also offer facilities to truncate, append to, create, move, delete and in-place modify files. They do not offer facilities to prepend to or truncate from the beginning of a file, let alone arbitrary insertion into or deletion from a file. The operations provided are highly asymmetric and lack the generality to be useful in unexpected contexts. For example, interprocess pipes in Unix have to be implemented outside of the file system because the pipes concept does not offer truncation from the beginning of files.

Secure access to basic file system operations can be based on a scheme of access control lists or capabilities. Research has shown access control lists to be difficult to secure properly, which is why research operating systems tend to use capabilities. Commercial file systems still use access control lists.

7.3 FILE MANAGEMENT SYSTEMS

A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File systems may use a data storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files, they might provide access to data on a file server by acting as clients for a network protocol (e.g., NFS, SMB, or 9P clients), or they may be virtual and exist only as an access method for virtual data.

More formally, a file system is a set of abstract data types that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data. File systems share much in common with database technology, but it is debatable whether a file system can be classified as a special-purpose database (DBMS).

7.3.1 Types of File Systems

File system types can be classified into disk file systems, network file systems and special purpose file systems.

- *Disk file systems:* A disk file system is a file system designed for the storage of files on a data storage device, most commonly a disk drive, which might be directly or indirectly connected to the computer. Examples of disk file systems include FAT, FAT32, NTFS, HFS and HFS+, ext2, ext3, ISO 9660, ODS-5, and UDF. Some disk file systems are journaling file systems or versioning file systems.

- *Flash file systems:* A flash file system is a file system designed for storing files on flash memory devices. These are becoming more prevalent as the number of mobile devices is increasing, and the capacity of flash memories catches up with hard drives.

  While a block device layer can emulate a disk drive so that a disk file system can be used on a flash device, this is suboptimal for several reasons:

  - *Erasing blocks:* Flash memory blocks have to be explicitly erased before they can be written to. The time taken to erase blocks can be significant, thus it is beneficial to erase unused blocks while the device is idle.
  - *Random access:* Disk file systems are optimized to avoid disk seeks whenever possible, due to the high cost of seeking. Flash memory devices impose no seek latency.
  - *Wear leveling:* Flash memory devices tend to wear out when a single block is repeatedly overwritten; flash file systems are designed to spread out writes evenly.

Log-structured file systems have all the desirable properties for a flash file system. Such file systems include JFFS2 and YAFFS.

- *Database file systems:* A new concept for file management is the concept of a database-based file system. Instead of, or in addition to, hierarchical structured management, files are identified by their characteristics, like type of file, topic, author, or similar metadata. Example: dbfs.

- *Transactional file systems:* Each disk operation may involve changes to a number of different files and disk structures. In many cases, these changes are related, meaning that it is important that they all be executed at the same time. Take for example a bank sending another bank some money electronically. The bank's computer will "send" the transfer instruction to the other bank and also update its own records to indicate the transfer has occurred. If for some reason the computer crashes before it has had a chance to update its own records, then on reset, there will be no record of the transfer but the bank will be missing some money.

  Transaction processing introduces the guarantee that at any point while it is running, a transaction can either be finished completely or reverted completely (though not necessarily both at any given point). This means that if there is a crash or power failure, after recovery, the stored state will be consistent. (Either the money will be transferred or it will not be transferred, but it won't ever go missing "in transit".)

  This type of file system is designed to be fault tolerant, but may incur additional overhead to do so.

Journaling file systems are one technique used to introduce transaction-level consistency to file system structures.

- *Network file systems:* A network file system is a file system that acts as a client for a remote file access protocol, providing access to files on a server. Examples of network file systems include clients for the NFS, SMB protocols, and file-system-like clients for FTP and WebDAV.
- *Special purpose file systems:* A special purpose file system is basically any file system that is not a disk file system or network file system. This includes systems where the files are arranged dynamically by software, intended for such purposes as communication between computer processes or temporary file space.

  Special purpose file systems are most commonly used by file-centric operating systems such as Unix. Examples include the procfs (/proc) file system used by some Unix variants, which grants access to information about processes and other operating system features.

  Deep space science exploration craft, like Voyager I & II used digital tape based special file systems. Most modern space exploration craft like Cassini-Huygens used Real-time operating system file systems or RTOS influenced file systems. The Mars Rovers are one such example of an RTOS file system, important in this case because they are implemented in flash memory.
- *Flat file systems:* In a flat file system, there are no subdirectories-everything is stored at the same (root) level on the media, be it a hard disk, floppy disk, etc. While simple, this system rapidly becomes inefficient as the number of files grows, and makes it difficult for users to organise data into related groups. Like many small systems before it, the original Apple Macintosh featured a flat file system, called Macintosh File System. Its version of Mac OS was unusual in that the file management software (Macintosh Finder) created the illusion of a partially hierarchical filing system on top of MFS. This structure meant that every file on a disk had to have a unique name, even if it appeared to be in a separate folder. MFS was quickly replaced with Hierarchical File System, which supported real directories.

7.3.2 File systems and Operating Systems

Most operating systems provide a file system, as a file system is an integral part of any modern operating system. Early microcomputer operating systems' only real task was file management - a fact reflected in their names. Some early operating systems had a separate component for handling file systems which was called a disk operating system. On some microcomputers, the disk operating system was loaded separately from the rest of the operating system. On early operating systems, there was usually support for only one, native, unnamed file system; for example, CP/M supports only its own file system, which might be called "CP/M file system" if needed, but which didn't bear any official name at all.

Because of this, there needs to be an interface provided by the operating system software between the user and the file system. This interface can be textual (such as provided by a command line interface, such as the Unix shell, or OpenVMS DCL) or graphical (such as provided by a graphical user interface, such as file browsers). If graphical, the metaphor of the folder, containing documents, other files, and nested folders is often used.

7.4 FILE SYSTEM ARCHITECTURE

Disks provide the bulk of secondary storage on which a file system is maintained. To improve I/O efficiency, I/O transfer between memory and disk are performed in units of blocks. Each block is one or more sectors. Depending on the disk drive, sectors vary from 32 bytes to 4096 bytes; usually, they are 512 bytes. Disks have two important characteristics that make them a convenient medium for storing multiple files:

- They can be rewritten in place; it is possible to read a block from the disk, to modify the block, and to write it back into the same place.
- One can access directly any given block of information on the disk. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires only moving the read-write heads and waiting for the disk to rotate.

To provide an efficient and convenient access to the disk, the operating system imposes a file system to allow the data to be stored, located, and retrieved easily. A file system poses two quite different design problems. The first problem is defining how the file system should look to the user. This task involves the definition of a file and its attributes, operations allowed on a file, and the directory structure for organizing the files. Next, algorithms and data structure must be created to map the logical file system onto the physical secondary storage devices.

The file system itself is generally composed of many different levels. The structure shown in the figure is an example of a layered design. Each level in the design uses the features of lower levels to create new features for use by higher levels.

The lowest level, the I/O control, consists of device drivers and interrupts handlers to transfer information between the memory and the disk system. A device driver can be thought of as a translator. Its input consists of high level commands such as "retrieve block 123". Its output consists of low-level hardware-specific instructions that are used by the hardware controller which interfaces the I/O device to the rest of the system. The device driver usually writes specific bit patterns to special locations in the I/O controller's memory to tell the controller on which device location to act and what actions to take.

The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (for example, drive 1, cylinder 3, track 2, sector 10).

The file-organization module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer. Each file's logical blocks are numbered from 0 (or 1) through N, whereas the physical blocks containing the data usually do not match the logical numbers, so a translation is needed to locate each block. The file-organization module also includes the free-space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.

Finally, the logical file system uses the directory structure to provide the file organization module with the information the latter needs, given a symbolic file name.

To create a new file, an application program calls the logical file system. The logical file system knows the format of the directory structures. To create a new file, it reads the appropriate directory into memory, updates it with the new entry, and writes it back to the disk. Some operating systems, including Unix, treat a directory exactly as a file - one with a type field indicating that it is a directory. Other operating systems, including Windows/NT, implement separate system calls for files and directories and treat directories as entities separate from files. When a directory is treated as a special file, the logical file system can call the file-organization module to map the directory I/O into disk block numbers, which are passed on to the basic file system and I/O control system.

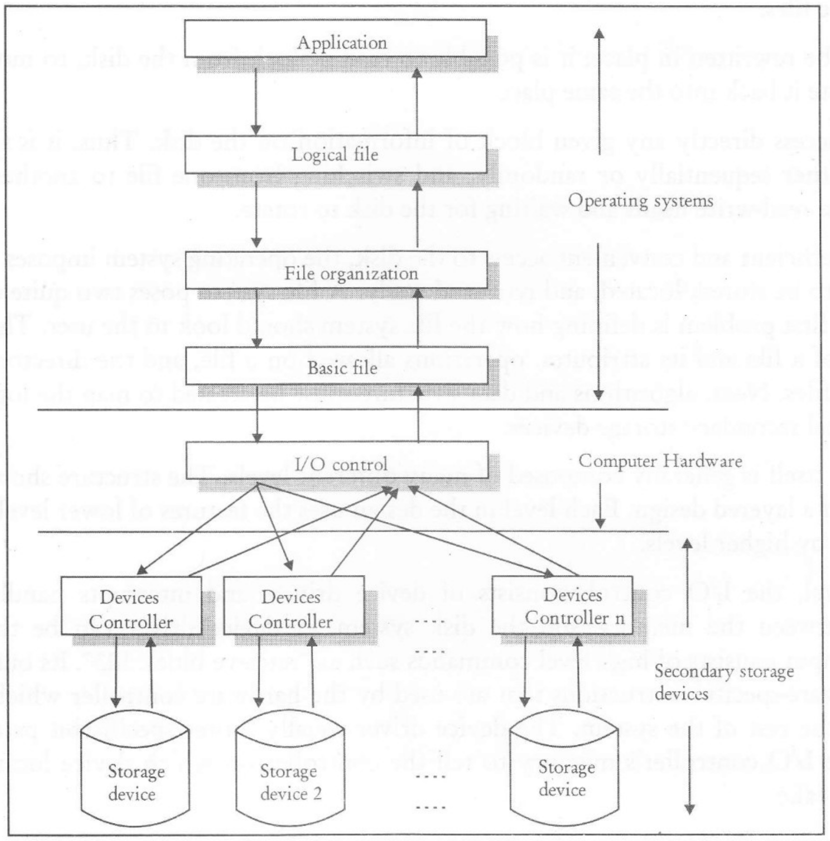

Figure 7.1: File system Architecture

Now that a file has been created, it can be used for I/O. For each I/O operation, the directory structure could be searched to find the file, its parameters checked, its data blocks located, and finally the operation on those data blocks performed. Each operation would entail high over-heads. Rather, before the file can be used for I/O procedures, it must be opened. When a file is opened, the directory structure is searched for the desired file entry. Parts of the directory structure are usually cached in memory to speed directory operations. Once the file is found, the associated information such as size, owner, access permissions and data block locations are generally copied into a table in memory, referred to as the open-file table, consisting of information about all the currently opened files.

The first reference to a file (normally an open) causes the directory structure to be searched and the directory entry for this file to be copied into the table as opened files. The index into this table is returned to the user program, and all further references are made through the index rather than with the symbolic name. The name given to the index varies. Unix systems refer to it as a file descriptor, Windows/NT as a file handle, and other systems as a file control block. Consequently, as long as the file is not closed, all file operations are done on the open-file table. When the file is closed by all users who have opened it, the updated file information is copied back to the disk-based directory structure.

Some systems complicate this scheme even further by using multiple levels of in-memory tables. For example, in the BSD UNIX file system, each process has an open-file table that holds a list of pointers, indexed by descriptor. The pointers lead to a system-wide open-file table. This table contains information about the underlying entity that is open. For files, it points to a table of active inodes. For other entities, such as network connections and devices, it points to similar access information. The active-inodes table is an in-memory cache of inodes currently in use, and includes the inode index fields that point to the on-disk data blocks. In this way, once a file is opened, all but the actual data blocks are in memory for rapid access by any process accessing the file. In reality, the open first searches the open-file table entry is created pointed to the system-wide open-file table. If not, the inode is copied into the active-inodes table, a new system-wide entry is created and a new per-process entry is created.

7.5 FILE SYSTEM FUNCTIONS

The file system provides the mechanism for online storage and access to both data and programs. The file system resides permanently on secondary storage, which has the main requirement that it must be able to hold a large amount of data, permanently. This chapter is primarily concerned with issues concerning file storage and access on the most common secondary-storage medium, the disk. We explore ways to allocate disk space, to recover freed space, to track the locations of data, and to interface other parts of the operating system to secondary storage. Performance issues are considered throughout the chapter.

- *File Naming:* Each file is a distinct entity and therefore a naming convention is required to distinguish one from another. The operating systems, generally employ a naming system for this purpose. In fact, there is a naming convention to identify each resource in the computer system not files alone.
- *File Types:* The files under UNIX can be categorized as follows:
  - *Ordinary files:* Ordinary files are the one, with which we all are familiar. They may contain executable programs, text or databases. You can add, modify or delete them or remove the file entirely.
  - *Directory files:* Directory files, as discussed earlier also, represent a group of files. They contain a list of file names and other information related to these files. Some of the commands that manipulate these directory files differ from those for ordinary files.
  - *Special files:* Special files are also referred to as device files. These files represent physical devices such as terminals, disks, printers and tape-drives etc. These files are read from or written into just like ordinary files, except that operation on these files activates some physical devices. These files can be of two types: Character device files and block device files. In character device files, data is handled character by character, as in case of terminals and printers. In block device files, data is handled in large chunks of blocks, as in the case of disks and tapes.
  - *FIFO files:* FIFO (first-in-first-out) are files that allow unrelated processes to communicate with each other. They are generally used in applications where the communication path is in only one direction, and several processes need to communicate with a single process. For an

example of FIFO file, take the pipe in UNIX. This allows transfer of data between processes in a first-in-first-out manner. A pipe takes the output of the first process as the input to the next process, and so on.

- *File Operations:* Major file operations are as follows:
  - ❖ Read operation
  - ❖ Write operation
  - ❖ Execute
  - ❖ Copying file
  - ❖ Renaming file
  - ❖ Moving file
  - ❖ Deleting file
  - ❖ Creating file
  - ❖ Merging files
  - ❖ Sorting file
  - ❖ Appending file
  - ❖ Comparing file
- *Symbolic Link:* A link is effectively a pointer or an alias to another file or sub-directory. For example, a link may be implemented as an absolute or relative path name (a symbolic link). When a reference to a file is made, we search the directory. The directory entry is marked as a link and the name of the real file (or directory) is given. We resolve the link by using the path name to locate the real file. Links are easily identified by their format in the directory entry (or by their having a special type on systems that support types), and are effectively named indirect pointers.

  A symbolic link can be deleted without deleting the actual file it links. There can be any number of symbolic links attached to a single file.

  Symbolic links are helpful in sharing a single file called by different names. Each time a link is created, the reference count in its inode is incremented by one. Whereas, deletion of link decreases the count by one. The operating system denies deletion of such files whose reference count is not 0, meaning that the file is in use.

  In a system where sharing is implemented by symbolic links, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed. If the file entry itself is deleted, the space for the file is deallocated, leaving the links dangling. We can search for these links and remove them also, but unless a list of the associated link is kept with each file, this search can be expensive. Alternatively, we can leave the links until an attempt is made to use them. At that time, we can determine that the file of the name given by the link does not exist, and can fail to resolve the link name; the access is treated just like any other illegal file name. (In this case, the system designer should consider carefully what to do when a file is deleted and another file of the same name is created, before a symbolic link to the original file is used.) In the case of UNIX, symbolic links are left when a file is deleted, and it is up to the user to realize that the original file is gone or has been replaced.

Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining that the last reference to the file has been deleted. We could keep a list of all references to a file (directory entries or symbolic links). When a link or a copy of the directory entry is established, a new entry is added to the file-reference list. When a link or directory entry is deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty.

The trouble with this approach is the variable and potentially large size of the file-reference list. However, we really do not need to keep the entire list — we need to keep only a count of the number of references. A new link or directory entry increments the reference counts; deleting a link or entry decrements the count. When the count is 0, the file can be deleted; there are no remaining references to it. The UNIX operating system uses this approach for non-symbolic links, or hard links, keeping a reference count in the file information block or inode). By effectively prohibiting multiple references to directories, we maintain an acyclic-graph structure.

To avoid these problems, some systems do not allow shared directories link. For example, in MS-DOS, the directory structure is a tree structure.

7.6 FILE SHARING

In a multi-user environment, a file is required to be shared among more than one users. There are several techniques and approaches to affect this operation. A simple approach is to copy the file at the users' local hard disk. This approach essentially creates different files, it therefore cannot be treated as file sharing.

A file can be shared in three different modes:

1. *Read only:* In this mode, the user can only read or copy the file.
2. *Linked shared:* In this mode, all the users sharing the file can make changes in this file but the changes are reflected in the order determined by the operating systems.
3. *Exclusive mode:* In this mode, the file is acquired by one single user who can make the changes while others can only read or copy it.

Another approach is to share a file through symbolic links. This approach poses a couple of problems - concurrent updation problem, deletion problem. If two users try to update the same file, the updation of one of them will be reflected at a time. Besides, a file must not be deleted while it is in use by another user.

Locking is a mechanism through which operating systems insures that the user making changes in the file is the one who has the lock on the file. As long as the lock remains with this user, no other user can alter the file.

7.7 FILE DIRECTORIES

The file systems of computers can be extensive. Some systems store thousands of files on hundreds of gigabytes of disk. To manage all these data, we need to organize them. This organization is usually done in two parts. First, the file system is broken into in the IBM world or volumes in the PC and Macintosh arenas. Typically, each disk on a system contains at least_< gngJ2flrtitioyi), which ISJI low-level structure in which files and directories inside. Sometimes, partitions are used to provide several separate areas within one disk, each treated as a separate storage device, whereas other systems allow partitions to be larger than a disk to group disks into one logical structure. In this way, the user needs to be concerned with only the logical directory and file structure, and can ignore completely the problems of physically allocating space for files. For this reason, partitions can be thought of as virtual disks.

Secondly, each partition contains information about files within it. This information is kept in a device directory or volume table of contents.

The device directory (more commonly known simply as a "directory") records information - such as name, location, size, and type - for all files on that partition.

The Logical Structure of a Directory

7.7.1 Single-Level Directory

The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand. A single-level directory has significant limitations however, when the number of files increases or when there is more than one user. Since all files are in the same directory, they must have unique names. If we have two users who call their data file test, then the unique-name rule is violated. For example, in one programming class, 23 students called the program for their second assignment prog 2, another 11 called it assigns. Although file names are generally selected to reflect the content of the file, they are often limited in length. The MS-DOS operating system allows only 11-character file names; UNIX allows 255 characters.

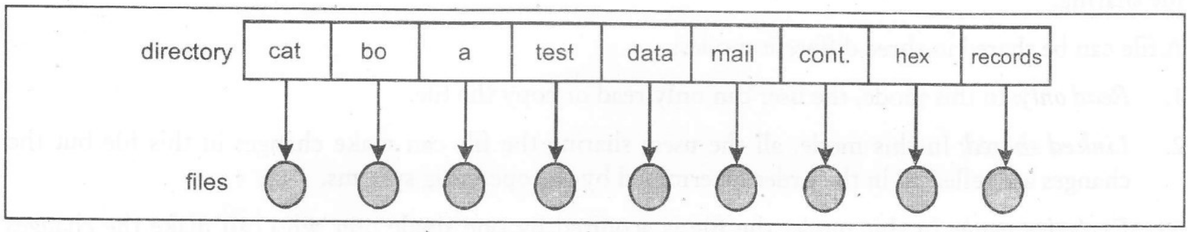

Figure 7.2: Single-Level Directory

Even with a single user, as the number of files increases, it becomes difficult to remember the names of all the files, so as to create only files with unique names. It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system. In such an environment, keeping track of so many files is a daunting task.

7.7.2 Two-Level Directory

The disadvantage of a single-level directory is confusion of file names. The standard solution is to create a separate directory for each user. In the two-level directory structure, each user has her own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. When a user starts or a user logs in, the system's master file directory is searched. The master file directory is indexed by user name or account.

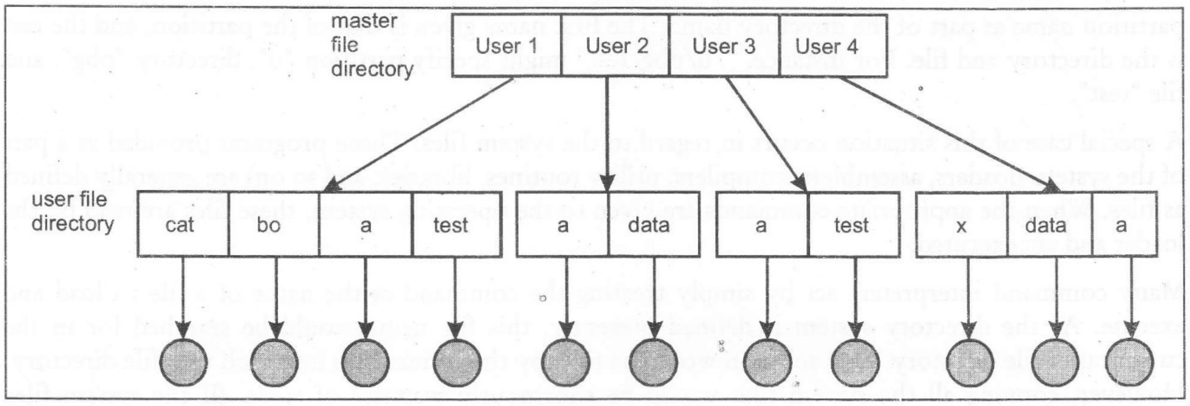

Figure 7.3: Two-Level Directory

When in a UDP a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the filenames within each UFD are unique.

To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.

The user directories themselves must be created and deleted as necessary. A special system program is run with the appropriate user name and account information. The program creates a new user file directory and adds an entry for it to the master file directory. The execution of this program might be restricted to system administrators. The allocation of disk space for user directories can be handled with the techniques discussed earlier for files themselves.

The two-level directory structure solves the name-collision problem, but it still has problems. This structure effectively isolates one user from another. This isolation is an advantage when the users are completely independent, but is a disadvantage when the users co-operate on some task and to access one user's account by other users is not allowed.

If access is to be permitted, one user must have the ability to name a file in another user's directory.

A two-level directory can be thought of as a tree, or at least an inverted tree. The root of the tree is the master file directory. Its direct descendants are the UFDs. The descendants of the user file directories are the files themselves.

Thus, a user name and a file name define a path name. Every file in the system has a path name. To name a file uniquely, user must know the path name of the file desired.

For example, if user A wishes to access her own test file named test, she can simply refer to test. To access the test file of user B (with directory-entry name user b), however, she might have to. refer to userb7/test. Every system has its own syntax for naming files in directories other than the user's own.

There is additional syntax to specify the partition of a file. For instance, in MS-DOS a partition is specified by a letter followed by a colon. Thus, file specification might be "C:/userb/bs.test". Some systems go even further and separate the partition, directory name, and file name parts of the specification. For instance, in VMS, the file "login.com" might be specified as:

"u:[sstdeck1]login.com;" where "u" is the name of the partition, "sst" is the name of the directory, "deck" is the name of subdirectory, and "1", is the version number. Other systems simply treat the partition name as part of the directory name. The first name given is that of the partition, and the rest is the directory and file. For instance, "/u/pbg/test" might specify partition "u", directory "pbg", and file "test".

A special case of this situation occurs in regard to the system files. Those programs provided as a part of the system (loaders, assemblers, compilers, utility routines, libraries, and so on) are generally defined as files. When the appropriate commands are given to the operating system, these files are read by the loader and are executed.

Many command interpreters act by simply treating the command as the name of a file to load and execute. As the directory system is defined presently, this file name would be searched for in the current user file directory. One solution would be to copy the system files into each user file directory. However, copying all the system files would be enormously wasteful of space. (If the system files require 5 megabytes, then supporting 12 users would require $5 \times 12 = 60$ megabytes just for copies of the system files.)

The standard solution is to complicate the search procedure slightly. A special user directory is defined to contain the system files (for example, user 0).

Whenever a file name is given to be loaded, the operating system first searches the local user file directory. If the file is found, it is used. If it is not found, the system automatically searches the special user directory that contains the system files. The sequence of directories searches the special user directory that contains the system files. The sequence of directories searched when a file is named, is called the search path. This idea can be extended, such that the search path contains an unlimited list of directories to search when a command name is given. This method is the one mostly used in UNIX and MS-DOS.

7.7.3 Tree-Structured Directories

Once we have seen how to view a two-level directory as a two-level tree, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own sub-directories and to organize their files accordingly. The MS-DOS system, (for instance) is structured as a tree. In fact, a tree is the most common directory structure. The tree has a root directory. Every file in the system has a unique path name. A path name is the path from the root, through all the sub-directories, to a specified file.

Directory (or sub-directory) contains a set of files or sub-directories. A directory is simply another file but it is treated in a special way. All directories have the same internal format, one bit in each directory entry defines the entry as a file (0) or as a subdirectory (1) Special system calls are used to create and delete directories. In normal use, each user has a current directory. The current directory should contain most of the files that are of current interest to the user. When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user must either specify a path name or change the current directory to be the directory holding that file. To change the current directory to a different directory, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory.

Thus, the user can change his current directory whenever he desires. From one change directory system call to the next, all open system calls search the current directory for the specified file.

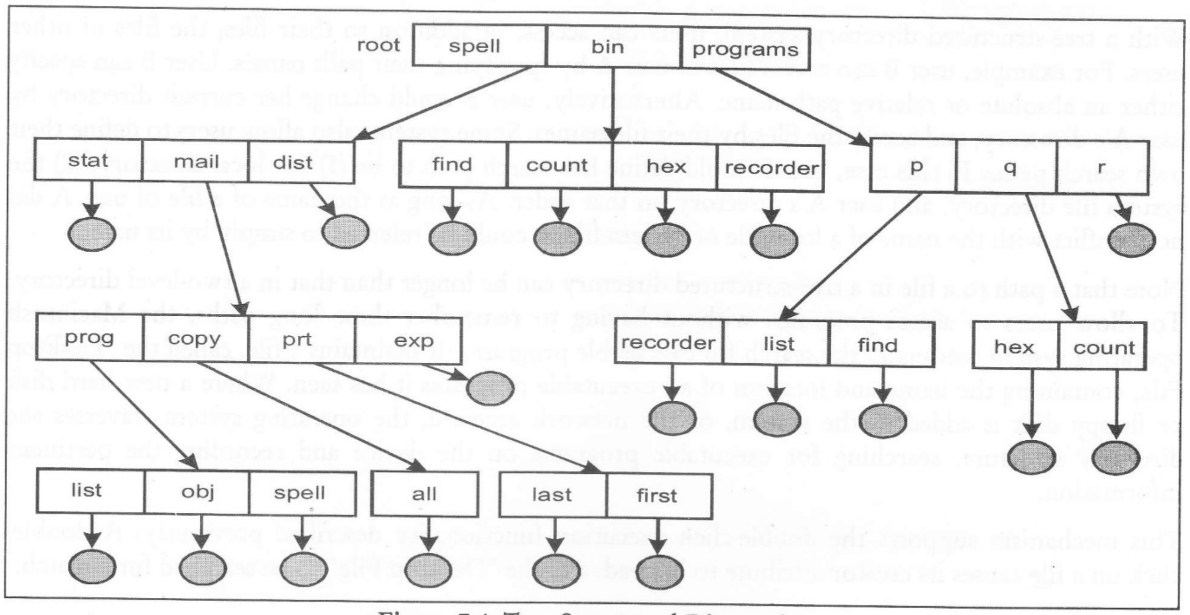

Figure 7.4: Tree-Structured Directories

The initial current directory of a user is designated when the user job starts or the user logs in. The operating system searches the accounting file (or ask) other predefined location to find an entry for this user (for accounting). In the accounting file is a pointer to (or the name of) the user's initial directory. This pointer is copied to a local variable for this user, which specifies the user's initial current directory.

Path names can be of two types: absolute path names or relative path names. An absolute path name begins at the root and follows a path down to the desired file, giving the directory names on the path. A relative path name defines a path from the current directory.

Allowing the user to define his own sub-directories permits him to impose a structure on his files. This structure might result in separate directories for files associated with different topics (for example, a subdirectory was created to hold the text of this book or different forms of information for example, the directory programs may contain source programs; the directory bin may store all the binaries. An interesting policy decision in a tree-structured directory structure is how to handle the deletion of a directory. If a directory is empty, its entry in its containing directory can simply be deleted. However, suppose the directory to be deleted is not empty, but contains several files, or possibly sub-directories. One of two approaches can be taken. Some systems, such as MS-DOS, will not delete a directory unless it is empty. Thus, to delete a directory, the user must first delete all the files in that directory. If there are any subdirectories, the procedure must be applied recursively to them, so that they can be deleted also.

This approach may result in a substantial amount of work. An alternative approach, such as that taken by the UNIX rm command, to provide the option that, when a request is made to delete a directory, and that directory's files and subdirectories are also to be deleted. Note that either approach is fairly easy to implement; the choice is one of policy. The latter policy is more convenient, but more dangerous, because an entire directory structure may be removed with one command. If that command was issued in error, a large number of files and directories would need to be restored from backup tapes.

With a tree-structured directory system, users can access, in addition to their files, the files of other users. For example, user B can access files of user A by specifying their path names. User B can specify either an absolute or relative path name. Alternatively, user B could change her current directory by user A's directory, and access the files by their file names. Some systems also allow users to define their own search paths. In this case, user B could define her search path to be (1) her local directory, (2) the system file directory, and user A's directory, in that order. As long as the name of a file of user A did not conflict with the name of a local file or system file, it could be referred to simply by its name.

Note that a path to a file in a tree-structured directory can be longer than that in a two-level directory. To allow users to access programs without having to remember these long paths, the Macintosh operating system automates the search for executable programs. It maintains a file, called the "Desktop File, containing the name and location of all executable programs it has seen. Where a new hard disk or floppy disk is added to the system, or the network accessed, the operating system traverses the directory structure, searching for executable programs on the device and recording the pertinent information.

This mechanism supports the double-click execution functionality described previously. A double-click on a file causes its creator attribute to be read, and the "Desktop File" to be searched for a match.

7.7.4 Acyclic-Graph Directories

Consider two programmers who are working on a joint project. The files associated with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories. The common subdirectory should be shared. A shared directory or file will exist in the file system in two (or more) places at once. Notice that a shared file (or directory) is not the same as two copies of the file. With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy. With a shared file, there is only one actual file, so any changes made by the person would be immediately visible to the other.

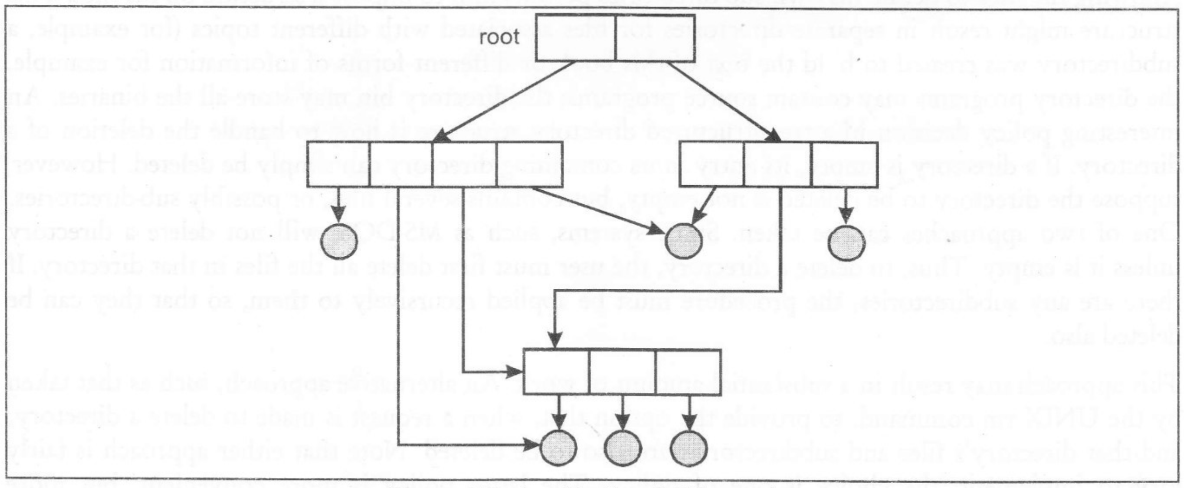

Figure 7.5: Acyclic-Graph Directories

This form of sharing is particularly important for shared subdirectories; a new file created by one person will automatically appear in all the shared subdirectories. A tree structure prohibits the sharing of files or directories. An acyclic graph allows directories to have shared subdirectories and files. The same file or subdirectory may be in two different directories. An acyclic graph (that is, a graph with no cycles) is a natural generalization of the tree-structured directory scheme.

In a situation where several people are working as a team, all the files to be shared may be put together into one directory. The user file directories of all the team members would each contain this directory of shared files as a subdirectory. Even when there is a single user, his file organization may require that some files be put into several different subdirectories. For example, a program written for a particular project should be both in the directory of all programs and in the directory for that project.

Shared files and subdirectories can be implemented in several ways. A common way, exemplified by many of the UNIX systems, is to create a new directory entry called a link. A link is effectively a pointer to another file or subdirectory. For example, a link may be implemented as an absolute or relative path name (a symbolic link). When a reference to a file is made, we search the directory. The directory entry is marked as a link and the name of the real file (or directory) is given. We resolve the link by using the path name to locate the real file. Links are easily identified by their format in the directory entry (or by their having a special type on systems that support types), and are effectively named indirect pointers. The operating system ignores these links while traversing directory trees to preserve the acyclic structure of the system.

The other approach to implementing shared files is simply to duplicate all information about them in both sharing directories. Thus, both entries are identical and equal. A link is clearly different from the original directory entry; thus, the two are not equal. Duplicate directory entries, however, make the original and the copy indistinguishable. A major problem with duplicate directory entries is maintaining consistency if the file is modified. An acyclic-graph directory structure is more flexible than is a simple tree structure, but is also more complex. Several problems must be considered carefully. Notice that a file may now have multiple absolute path names. Consequently, distinct file names may refer to the same file. This situation is similar to the aliasing problem for programming languages. If we are trying to traverse the entire file system (to find a file, to accumulate statistics on all files, or to copy all files to backup storage), this problem becomes significant, since we do not want to traverse shared structures more than once.

Another problem involves deletion. When can the space allocated to a shared file be deallocated and reused? One possibility is to remove the file whenever anyone deletes it, but this action may leave dangling pointers to the non-existent file. Worse, if the remaining file pointers contain actual disk addresses, and the space is subsequently reused for other files, these dangling pointers may point into the middle of other files.

In a system where sharing is implemented by symbolic links, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed. If the file entry itself is deleted, the space for the file is deallocated, leaving the links dangling. We can search for these links and remove them to, but unless a list of the associated link is kept with each file, this search can be expensive. Alternatively, we can leave the links until an attempt is made to use them. At that time, we can determine that the file of the name given by the link does not exist, and can fail to resolve the link name; the access is treated just like any other illegal file name. (In this case, the system designer should consider carefully what to do when a file is deleted and another file of the same name is created, before a symbolic link to the original file is used.) In the case of UNIX, symbolic links are left when a file is deleted, and it is up to the user to realize that the original file is gone or has been replaced.