Unit II- File System

1. <u>What is mean by file? Explain file structure types.</u>

- All computer applications need to store and get back information. While a process is running, it can store a limited amount of information within its own address space. On the other hand, the storage capacity is restricted to the size of the virtual address space. For some applications this size is adequate, but for others, such as airline reservations, banking, or corporate record keeping, it is far too small.
- A second problem with keeping information within a process address space is that when the process terminates, the information is lost. For many applications, (e.g., for databases), the information must be retained for weeks, months, or even forever. Having it vanish when the process using it terminates is unacceptable. Moreover, it must not go away when a computer crash kills the process.
- A third problem is that it is often necessary for multiple processes to access (parts of) the information at the same time. If we have an online telephone directory stored inside the address space of a single process, only that process can access it. The way to solve this problem is to make the information itself independent of any one process.

  Therefore we have three essential requirements for long-term information storage:

  1 It must be possible to store a very large amount of information.

  2. The information must survive the termination of the process using it.

  3. Multiple processes must be able to access the information concurrently.

- Files are logical units of information created by processes. A disk will generally contains thousands or even millions of them, each one independent of the others. In reality, if you think of each file as a kind of address space, you are not that far off, except that they are used to model the disk instead of modeling the RAM.

<u>File Naming</u>

- Files are an abstraction mechanism. They provide a way to store information on the disk and read it back later. This must be done in such a way as to shield the user from the details of how and where the information is stored, and how the disks really work.

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Figure 1. Some typical file extensions.

<u>File Structure</u>

- Files can be structured in any of various ways. Three common possibilities are s in Figure 2. The file in Figure 2(a) is 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 use this approach.

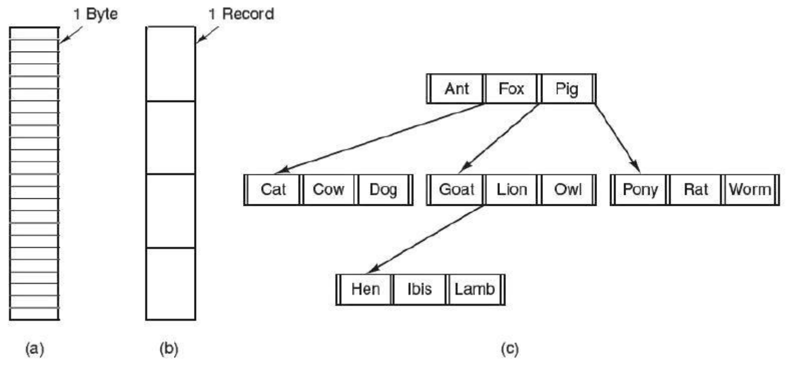

Figure 2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

- Having the operating system regard files as nothing more than byte sequences provides the maximum flexibility. User programs can put anything they want in their files and name them any way that is convenient. The operating system does not help, but it also does not get in the way. For users who want to do unusual things, the latter can be very important. All versions of UNIX, MS-DOS, and Windows use this file model.
- The first step up in structure is shown in Figure 2(b). 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, in decades gone by, 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. These systems also supported files of 132-character records, which were intended for the line printer (which in those days were big chain printers having 132

columns). Programs read input in units of 80 characters and wrote it in units of 132 characters, although the final 52 could be spaces, of course. No current general-purpose system use this model as its primary file system any more, but back in the days of 80-column punched cards and 132-character line printer paper this was a common model on mainframe computers.

- The third kind of file structure is shown in Figure 2(c). 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.

2. Give any 5 types of file attributes and file extensions.

- Every file has a name and its data. Moreover, all operating systems associate other information with each file, for instance, the date and time the file was last modified and the file's size. We will call these extra items the file's attributes. Some people call them metadata. The list of attributes varies significantly from system to system. The table of Figure 1 shows some of the possibilities, but other ones also exist. No existing system has all of these, but each one is present in some system.

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 1. Some possible file attributes

- The first four attributes relate to the file's protection and tell who may access it and who may not. All kinds of schemes are possible, some of which we will study later. In some systems the user must present a password to access a file, in which case the password must be one of the attributes.

- The flags are bits or short fields that control or enable some particular property. Hidden files, for instance, do not appear in listings of all the files. The archive flag is a bit that keeps track of whether the file has been backed up recently. The backup program clears it, and the operating system sets it whenever a file is changed. Thus, the backup program can tell which files need backing up. The temporary flag allows a file to be marked for automatic deletion when the process that created it terminates.
- The record length, key position, and key length fields are only present in files whose records can be looked up using a key. They provide the information required to find the keys.
- The various times keep track of when the file was created, most recently accessed, and most recently modified. These are useful for a variety of purposes. For instance, a source file that has been modified after the creation of the corresponding object file needs to be recompiled. These fields provide the necessary information.
- The current size tells how big the file is at present. Some old mainframe operating systems require the maximum size to be specified when the file is created, in order to let the operating system reserve the maximum amount of storage in advance. Workstation and personal computer operating systems are clever enough to do without this feature.

3. Explain any five file operations.

Files exist to store information and allow it to be retrieved later. Different systems provide different operations to allow storage and retrieval.

1. Create. The file is created with no data. The purpose of the call is to announce that the file is corning and to set some of the attributes.

2. Delete. When the file is no longer required, it has to be deleted to free up disk space. There is always a system call for this purpose.

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

4. Close. When all the accesses are finished, the attributes and disk addresses are no longer required, so the file should be closed to free up internal table space. Many systems encourage this by imposing a maximum number of open files on processes. A disk is written in blocks, and closing a file forces writing of the file's last block, even though that block may not be entirely full yet.

5. Read. Data are read from file. Generally, the bytes come from the current position. The caller must specify how many data are required and must also provide a buffer to put them in.

6. Write. Data are written to the file again, generally at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten and lost forever.

7. Append. This call is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide several ways of doing the same thing, and these systems sometimes have append.

8. Seek. For random access files, a method is required to specify from where to take the data. One common approach is a system call, seek, that repositions the file pointer to a particular place in the file. After this call has completed, data can be read from, or written to, that position.

9. Get attributes. Processes often need to read file attributes to do their work. For instance, the UNIX make program is commonly used to manage software development projects consisting of many source files. When make is called, it examines the modification times of all the source and object files and arranges for the minimum number of compilations required to bring everything up to date. To do its job, it must look at the attributes, namely, the modification times.

10. Set attributes. Some of the attributes are user settable and can be changed after the file has been created. This system call makes that possible. The protection mode information is an obvious example. Most of the flags also fall in this category.

11. Rename. It frequently happens that a user needs to change the name of an existing file. This system call makes that possible. It is not always strictly necessary, because the file can generally be copied to a new file with the new name, and the old file then deleted.

5. Explain file types.

File Type

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

Ordinary files

- These are the files that contain user information.

- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

- **Character special files** − data is handled character by character as in case of terminals or printers.
- **Block special files** − data is handled in blocks as in the case of disks and tapes.

6. Give file access methods.

File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

- Random access file organization provides, accessing the records directly.

- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contigous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

7. What is means by directory?

To keep track of files, file systems generally have directories or folders, which in many systems are themselves files. Now we will discuss directories, their organization, their properties, and the operations that can be performed on them.

Single-Level Directory Systems

The simplest form of directory system is having one directory containing all the files. Sometimes it is called the root directory, but since it is the only one, the name does not matter much.
An example of a system with one directory is given in Figure 1. Here the directory includes four files. The advantages of this scheme are its simplicity and the ability to locate files quickly - there is only one place to look, after all. It is normally used on simple embedded devices such as telephones, digital cameras, and some portable music players.

Hierarchical Directory Systems

The single-level is sufficient for simple dedicated applications (and was even used on the first personal computers), but for modern users with thousands of files, it would be impossible to find anything if all files were in a single directory.

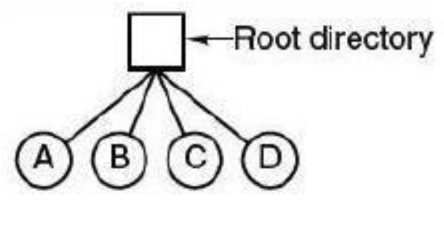

Figure 1. A single-level directory system containing four files

- Therefore, a way is required to group related files together. A professor, for instance, might have a collection of files that together form a book that he is writing for one course, a second collection of files containing student programs submitted for another course, a third group of files containing the code of an advanced compiler-writing system he is building, a fourth group of files containing grant proposals, as well as other files for electronic mail, minutes of meetings, papers he is writing, games, and so on.

- What is required is a hierarchy (i.e., a tree of directories). With this approach, there can be as many directories as are required to group the files in natural ways. In addition, if multiple users share a common file server, as is the case on many company networks, each user can have a private root directory for his or her own hierarchy. This approach is shown in Figure 2. Here, the directories A, B, and C contained in the root directory each belong to a different user, two of whom have created subdirectories for projects they are working on.

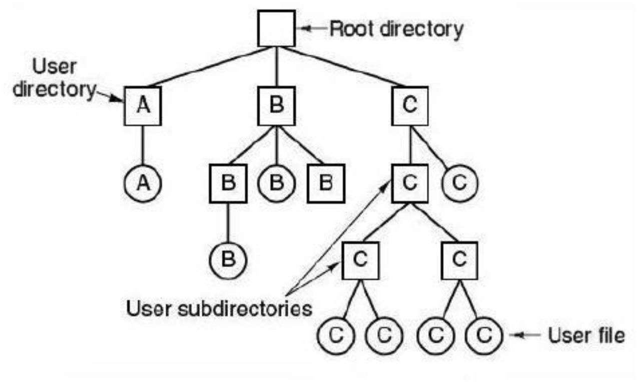

Figure 2. A hierarchical directory system.

- The ability for users to create an arbitrary number of subdirectories provides a powerful structuring tool for users to organize their work. Therefore, nearly all modern file systems are organized in this manner.

<u>Path Names</u>

- When the file system is organized as a directory tree, some way is required for specifying file names. Two different methods are frequently used. In the first method, each file is given an absolute path name consisting of the path from the root directory to the file.
- The first path instructs the system to go upward (to the usr directory), then to go down to the directory lib to find the file dictionary.

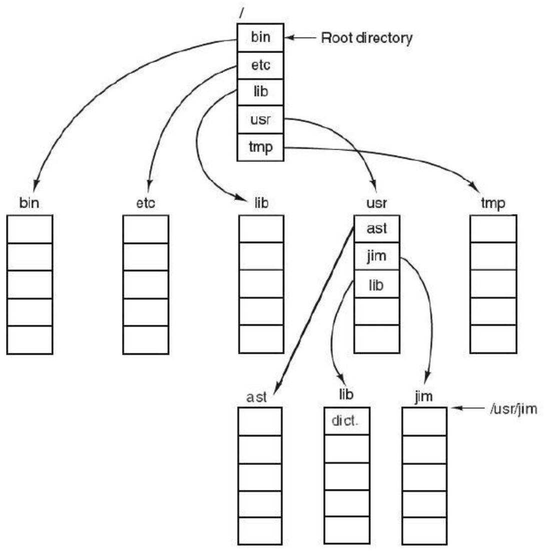

Figure 3. A UNIX directory tree.

- The second argument (dot) names the current directory. When the cp command gets a directory name (including dot) as its last argument, it copies all the files to that directory. Of course, a more normal way to do the copy would be to use the full absolute path name of the source file:

```
cp /usr/lib/dictionary .
```

Here the use of dot saves the user the trouble of typing dictionary a second time. However, typing

```
cp /usr/lib/dictionary dictionary
```

also works fine, as does

```
cp /usr/lib/dictionary /usr/ast/dictionary
```

8. What are the operations that can be performed on a directory?

The allowed system calls for managing directories exhibit more variation from system to system than system calls for files. To give an impression of what they are and how they work, we will give a sample (taken from UNIX).

1. Create. A directory is created. It is empty except for dot and dotdot, which are put there automatically by the system (or in a few cases, by the mkdir program).

2. Delete. A directory is deleted. Only an empty directory can be deleted. A directory containing only dot and dotdot is considered empty as these cannot usually be deleted.

3. Opendir. Directories can be read. For example, to list all the files in a directory, a listing program opens the directory to read out the names of all the files it includes. Before a directory can be read, it must be opened, similar to opening and reading a file.

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. Formerly, it was possible to read directories using the usual read system call, but that approach has the disadvantage of forcing the programmer to know and deal with the internal structure of directories. On the contrary, readdir always returns one entry in a standard format, no matter which of the possible directory structures are being used.

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

7. Link. Linking is a technique that allows a file to appear in more than one directory. This system call specifies an existing file and a path name, and creates a link from the existing file to the name specified by the path. In this way, the same file may appear in multiple directories. A link of this kind, which increments the counter in the file's i-node (to keep track of the number of directory entries containing the file), is sometimes called a hard link.

8. Unlink. A directory entry is removed. If the file being unlinked is only present in one directory (the normal case), it is removed from the file system. If it is present in multiple directories, only the path name specified is removed. The others remain. In UNIX, the system call for deleting files (discussed earlier) is, in fact, unlink.

9. Explain Absolute and relative pathnames.

An **absolute pathname** is a pathname that starts from the root path and specifies all directories between the root path and the specified directory or file.

For example, here are some example root paths:

In DOS/Windows, the root path is typically a device letter such as C:\.
In Unix-based file systems, the root path is usually just / or some device volume locations such as/volume/hdd1.
On the Mac, the root path is usually a device name like My hard drive:, but since OS X is Unix-based it can also have that form of root path (usually only when operating from the command-line or running Unix-like apps).

Here are some example absolute pathnames:

In DOS/Windows: C:\Documents and Settings\user1\Local Settings\Application Data\Adobe\Acrobat

Unix: /users/user1/Library/Quicktime/
Mac: HD:Users:user1:Library:Quicktime:

A **relative pathname** is a pathname that specifies some abbreviations for traversing up the directory structure and down to another directory or file in the same directory tree. In most file systems (except Mac), the .. (two periods) abbreviation indicates "go up one directory level".
Here are some example relative pathnames:

In DOS/Windows: ..\Application Data\Adobe\Acrobat
Unix: ~user1/Library/Quicktime/, ~../user1/Library/Quicktime/
Mac: ::user1:Library:Quicktime:

10. Explain any two file system implementation methods.

<u>File System Layout</u>

- File systems are stored on disks. Most disks can be divided up into one or more partitions, with independent file systems on each partition. Sector 0 of the disk is called the MBR (Master Boot Record) and is used to boot the computer. The end of the MBR contains the partition table. This table gives the starting and ending addresses of each partition. One of the partitions in the table is marked as active. When the computer is booted, the BIOS reads in and executes the MBR. The first thing the MBR program does is locate the active partition, read in its first block, called the boot block, and execute it. The program in the boot block loads the operating system contained in that partition. For uniformity, every partition starts with a boot block, even if it does not contain a boatable operating system. Besides, it might contain one in the future.

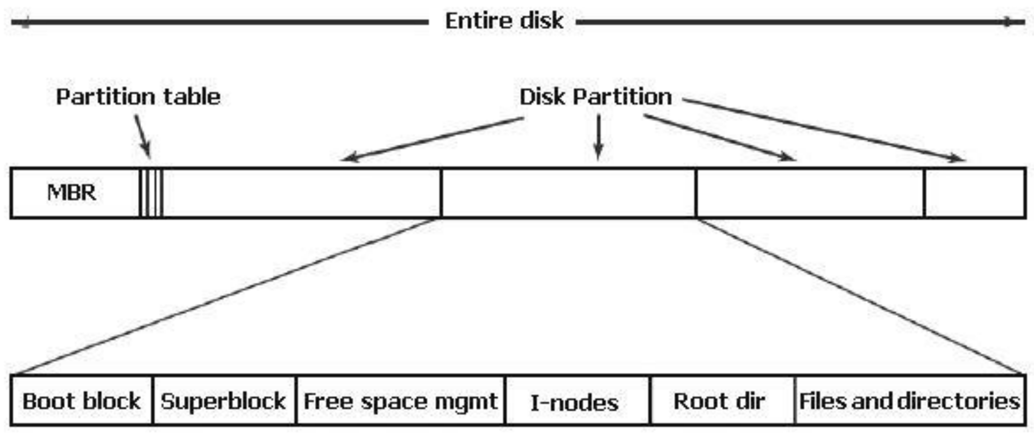

Figure 1. A possible file system layout

Implementing Files

1. Contiguous Allocation

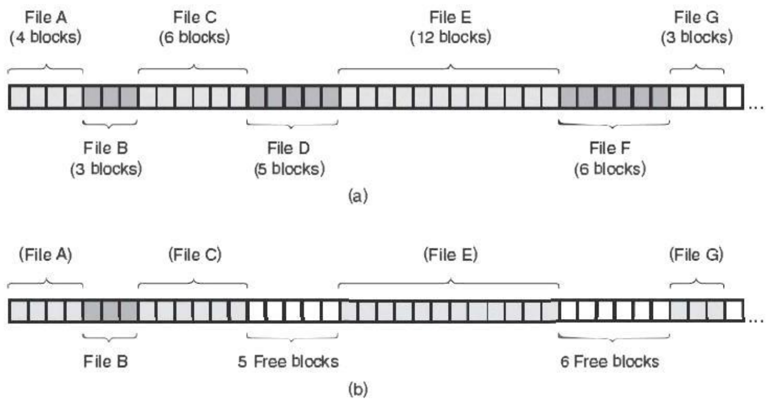

Figure 2.(a) Contiguous allocation of disk space for seven files.
(b) The state of the disk after files D and F have been removed

- Contiguous disk space allocation has two significant advantages. First, it is simple to implement because keeping track of where a file's blocks are is reduced to remembering two numbers: the disk address of the first block and the number of blocks in the file. Given the number of the first block, the number of any other block can be found by a simple addition.

Linked List Allocation

- The second method for storing files is to keep each one as a linked list of disk blocks, as shown in Figure 3. The first word of each block is used as a pointer to the next one. The rest of the block is for data.

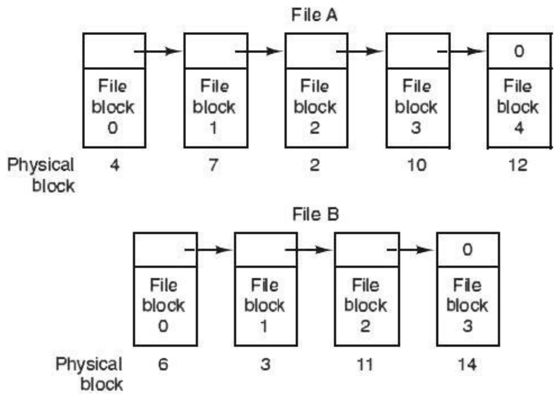

Figure 3. Storing a file as a linked list of disk blocks

- Unlike contiguous allocation, every disk block can be used in this method. No space is lost to disk fragmentation (except for internal fragmentation in the last block). Also, it is sufficient for the directory entry to merely store the disk address of the first block. The rest can be found starting there.

<u>Linked List Allocation Using a Table in Memory</u>

- Both disadvantages of the linked list allocation can be eliminated by taking the pointer word from each disk block and putting it in a table in memory.

- Using this organization, the entire block is available for data. Moreover, random access is much easier. Although the chain must still be followed to find a given offset within the file, the chain is entirely in memory, so it can be followed without making any disk references. Like the previous method, it is sufficient for the directory entry to keep a single integer (the starting block number) and still be able to locate all the blocks, no matter how large the file is.

- The primary disadvantage of this method is that the entire table must be in memory all the time to make it work.

11. Write a note on I-nodes and linked list allocation.

<u>Linked List Allocation</u>

- The linked list method for storing files is to keep each one as a linked list of disk blocks, as shown in Figure 3. The first word of each block is used as a pointer to the next one. The rest of the block is for data.

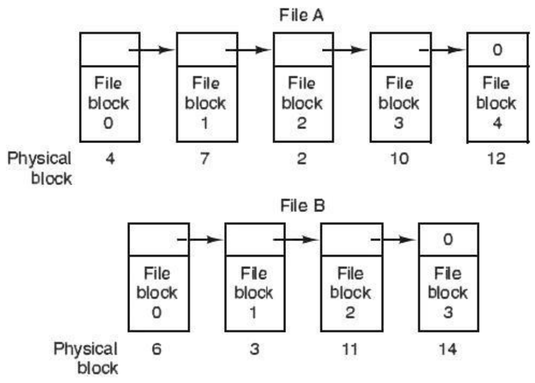

Figure 3. Storing a file as a linked list of disk blocks

- Unlike contiguous allocation, every disk block can be used in this method. No space is lost to disk fragmentation (except for internal fragmentation in the last block). Also, it is sufficient for the directory entry to merely store the disk address of the first block. The rest can be found starting there.

- On the other hand, although reading a file sequentially is straightforward, random access is extremely slow. To get to block n, the operating system has to start at the beginning and read the n - 1 blocks prior to it, one at a time. Clearly, doing so many reads will be painfully slow.

- Also, the amount of data storage in a block is no longer a power of two because the pointer takes up a few bytes. While not fatal, having a peculiar size is less efficient because many programs read and write in blocks whose size is a power of two. With the first few bytes of each block occupied to a pointer to the next block, reads of the full block size require acquiring and concatenating information from two disk blocks, which generates extra overhead due to the copying.

I-nodes

- Our last method for keeping track of which blocks belong to which file is to associate with each file a data structure called an i- node (index-node), which lists the attributes and disk addresses of the file's blocks. A simple example is depicted in Figure 5. Given the i-node, it is then possible to find all the blocks of the file. The big advantage of this scheme over linked files using an in-memory table is that the i-node need only be in memory when the corresponding file is open. If each i-node occupies n bytes and a maximum of k files may be open at once, the total memory occupied by the array holding the i-nodes for the open files is only kn bytes. Only this much space need be reserved in advance.

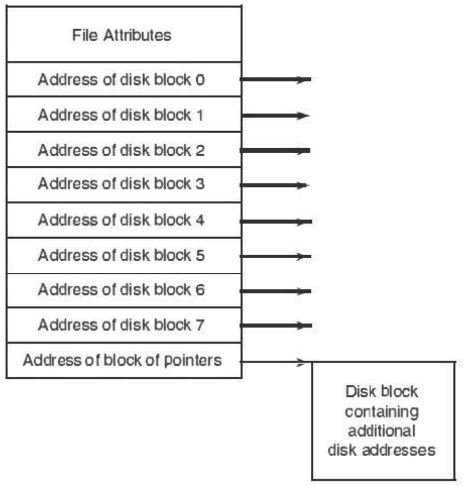

Figure 5. An example i-node

- This array is generally far smaller than the space occupied by the file table described in the previous section. The reason is simple. The table for holding the linked list of all disk blocks is proportional in size to the disk itself. If the disk has n blocks, the table needs n entries. As disks grow larger, this table grows linearly with them. In contrast, the i-node scheme requires an array in memory whose size is proportional to the maximum number of files that may be open at once. It does not matter if the disk is10 GB or 100 GB or 1000 GB.

- One problem with i-nodes is that if each one has 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 block containing more disk block addresses, as shown in Figure 5. Even more advanced would be two or more such blocks containing disk addresses or even disk blocks pointing to other disk blocks full of addresses.

12. What is mean by shared files and virtual file system?

- When various users are working together on a project, they often need to share files. As a result, it is often convenient for a shared file to appear simultaneously in different directories belonging to different users. Figure 1 shows the file system of "DIRECTORIES" Figure 2, again, only with one of C's files now present in one of B's directories as well. The connection between B's directory and the shared file is called a link. The file system itself is now a Directed Acyclic Graph, or DAG, rather than a tree.

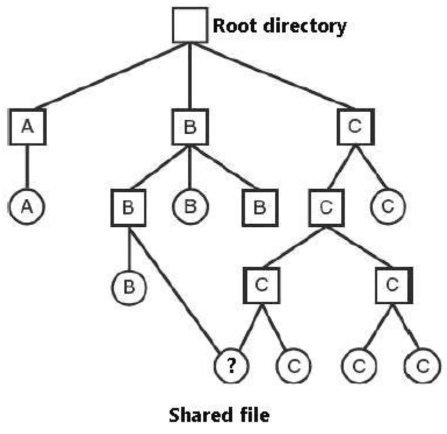

Figure 1. File system containing a shared file.

- Sharing files is convenient, but it also introduces some problems. To start with, if directories really do contain disk addresses, then a copy of the disk addresses will have to be made in B's directory when the file is linked. If either B or C subsequently appends to the file, the new blocks will be listed only in the directory of the user doing the append. The changes will not be visible to the other user, thus defeating the purpose of sharing.

- This problem can be solved in two ways. In the first solution, disk blocks are not listed in directories, but in a little data structure associated with the file itself. The directories would then point just to the little data structure. This is the approach used in UNIX (where the little data structure is the i-node).

- In the second solution, B links to one of C's files by having the system create a new file, of type LINK, and entering that file in B 's directory. The new 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, to contrast it with traditional (hard) linking.

Virtual File Systems

- A virtual file system (VFS) is programming that forms an interface between an operating system's kernel and a more concrete file system.

- The VFS serves as an abstraction layer that gives applications access to different types of file systems and local and network storage devices. For that reason, a VFS may also be known as a *virtual file system switch*. It also manages the data storage and retrieval between the operating system and the storage sub-system. The VFS maintains a cache of directory lookups to enable easy location of frequently accessed directories.

- Sun Microsystems introduced one of the first VFSes on Unix-like systems. The VMware Virtual Machine File System (VMFS), NTFS, Linux's Global File System (GFS) and the Oracle Clustered File System (OCFS) are all examples of virtual file systems.

- Several different file systems are in use - often on the same computer - even for the same operating system. A Windows system may have a main NTFS file system, but also a legacy FAT-32 or FAT-16 drive or partition that includes old, but still needed, data and occasionally a CD-ROM or DVD (each with its own unique file system) may be needed as well. Windows handles these disparate file systems by identifying each one with a different drive letter, as in C:, D:, etc. When a process opens a file, the drive letter is explicitly or implicitly present so Windows knows which file system to pass the request to. There is no attempt to integrate heterogeneous file systems into a unified whole.

- On the contrary, all modern UNIX systems make a very serious attempt to integrate multiple file systems into a single structure. A Linux system could have ext2 as the root file system, with an ext3 partition mounted on /usr and a second hard disk with a ReiserFS file system mounted on /home as well as an ISO 9660 CD-ROM temporarily mounted on /mnt. From the user's point of view, there is a single file system hierarchy.

That it happens to include multiple (incompatible) file systems is not visible to users or processes.

- On the other hand, the presence of multiple file systems is very definitely visible to the implementation, and since the pioneering work of Sun Microsystems (Kleiman, 1986), most UNIX systems have used the concept of a VFS (virtual file system) to try to integrate multiple file systems into an orderly structure. The key idea is to abstract out that part of the file system that is common to all file systems and put that code in a separate layer that calls the underlying concrete file systems to actual manage the data. The overall structure is shown in Figure 1. The discussion below is not specific to Linux or FreeBSD or any other version of UNIX, but gives the general flavor of how virtual file systems work in UNIX systems.

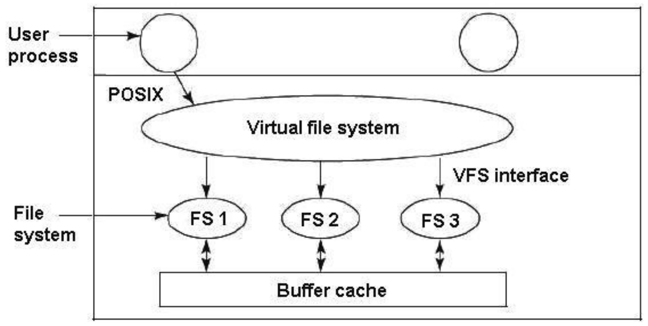

Figure 1. Position of the virtual file system.

13. Write a note on Disk Space Management.
14. How to keep track of free block?

- Making the file system work is one thing; making it work efficiently and robustly in real life is another thing. In the following sections we will examine some of the issues involved in managing disks.

Disk Space Management

- Files are usually stored on disk, so management of disk space is a main concern to file system designers. Two common strategies are possible for storing an n byte file: n consecutive bytes of disk space are allocated, or the file is split up into a number of (not necessarily) contiguous blocks. The same trade-off is present in memory management systems between pure segmentation and paging.

- As we have seen, storing a file as a contiguous sequence of bytes has the obvious problem that if a file grows, it will probably have to be moved on the disk. The same problem holds for segments in memory, except that moving a segment in memory is a relatively fast operation compared to moving a file from one disk position to another. Therefore, nearly all file systems chop files up into fixed-size blocks that need not be adjacent.

Block Size

- Once it has been decided to store files in fixed-size blocks, the question arises of how big the block should be. Given the way disks are organized, the sector, the track, and the cylinder are obvious candidates for the unit of allocation (although these are all device dependent, which is a minus). In a paging system, the page size is also a major contender.

- Having a large block size means that every file, even a 1-byte file, ties up an entire cylinder. It also means that small files waste a large amount of disk space. On the other hand, a small block size means that most files will span multiple blocks and thus need multiple seeks and rotational delays to read them, reducing performance. Thus if the allocation unit is too large, we waste space; if it is too small, we waste time.

Keeping Track of Free Blocks

- Once a block size has been chosen, the next issue is how to keep track of free blocks. Two methods are broadly used, The first one consists of using a linked list of disk blocks, with each block holding as many free disk block numbers as will fit. The other free space management technique is the bitmap. A disk with n blocks requires a bitmap with n bits. Free blocks are represented by 1s in the map, allocated blocks by 0s (or vice versa).

- An alternative approach that avoids most of this disk I/O is to split the full block of pointers. Thus instead of going from Figure 4(a) to Figure 4(b), we go from Figure 4(a) to Figure 4(c) when three blocks are freed. Now the system can handle a series of temporary files without doing any disk I/O. If the block in memory fills up, it is written to the disk, and the half-full block from the disk is read in.

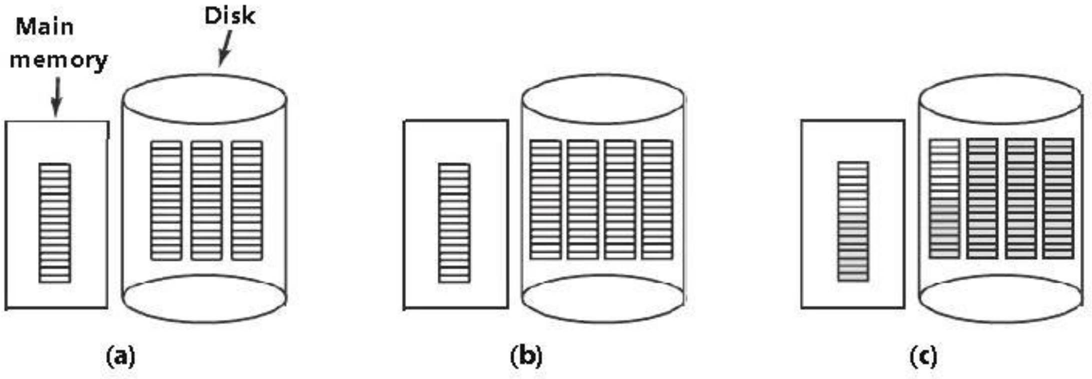

Figure 4. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

- The idea here is to keep most of the pointer blocks on disk full (to minimize disk usage), but keep the one in memory about half full, so it can handle both file creation and file removal without disk I/O on the free list.

Disk Quotas

- To prevent people from hogging too much disk space, multiuser operating systems often provide a mechanism for enforcing disk quotas. The idea is that the system administrator assigns each user a maximum allotment of files and blocks, and the operating system makes sure that the users do not exceed their quotas.

15. Explain File system backup.

- Destruction of a file system is sometimes a far greater tragedy than destruction of a computer. If a computer is destroyed by fire, lightning surges, or a cup of coffee poured onto the keyboard, it is annoying and will cost money, but usually a replacement can be purchased with a minimum of fuss. Inexpensive personal computers can even be replaced within an hour by just going to a computer store.
- If a computer's file system is permanently lost, whether due to hardware or software, restoring all the information will be difficult, time consuming, and in many cases, impossible. For the people whose programs, documents, tax records, customer files, databases, marketing plans, or other data are gone forever, the consequences can be extremely harmful. While the file system cannot offer any protection against physical destruction of the equipment and media, it can help protect the information. It is pretty straightforward: make backups. But that is not quite as simple as it sounds. Let us examine.
- Most people do not think making backups of their files is worth the time and effort - until one fine day their disk suddenly dies, at which time most of them undergo a deathbed conversion. Companies, on the other hand, (generally) well understand the value of their data and usually do a backup at least once a day, generally to tape. Modern tapes hold hundreds of gigabytes and cost pennies per gigabyte. However, making backups is not quite as trivial as it sounds, so we will look at some of the related issues below.

- Backups to tape are usually made to handle one of two potential problems:

  1. Recover from disaster.
  2. Recover from stupidity.

- The first one covers getting the computer running again after a disk crash, fire, flood, or other natural disaster. In practice, these things do not happen very often, which is why many people do not bother with backups. Making a backup takes a long time and occupies a large amount of space, so doing it efficiently and conveniently is important.

- These considerations raise the following issues. First, should the entire file system be backed up or only part of it? At many installations, the executable (binary) programs are kept in a limited part of the file system tree. It is not necessary to back up these files if they can all be reinstalled from the manufacturer's CD-ROMs. Also, most systems have a directory for temporary files. There is generally no reason to back it up either. In UNIX, all the special files (I/O devices) are kept in a directory /dev. Not only is backing up this directory not necessary, it is downright dangerous because the backup program would hang forever if it tried to read each of these to completion. In short, it is generally desirable to back up only particular directories and everything in them rather than the entire file system.
- Second, it is wasteful to back up files that have not changed since the previous backup, which leads to the idea of incremental dumps. The simplest form of incremental dumping is to make a complete dump (backup) from time to time, say weekly or monthly, and to make a daily dump of only those files that have been modified since the last full dump.
- Third, since huge amounts of data are normally dumped, it may be desirable to compress the data before writing them to tape.
- Fourth, it is difficult to perform a backup on an active file system. If files and directories are being added, deleted, and modified during the dumping process, the resulting dump may be inconsistent.
- Fifth and last, making backups introduces many nontechnical problems into an organization. The best online security system in the world may be useless if the system administrator keeps all the backup tapes in his office and leaves it open and unguarded whenever he walks down the hall to get output from the printer.

Two strategies can be used for dumping a disk to tape: a physical dump or a logical dump.

- A physical dump starts at block 0 of the disk, writes all the disk blocks onto the output tape in order, and stops when it has copied the last one. Such a program is so simple that it can probably be made 100% bug free, something that can probably not be said about any other useful program. However, it is worth making a number of comments about physical dumping. For one thing, there is no value in backing up unused disk blocks. If the dumping program can obtain access to the free block data structure, it can avoid dumping unused blocks. On the other hand, skipping unused blocks requires writing the number of each block in front of the block (or the equivalent), since it is no longer true that block k on the tape was block k on the disk.
- A logical dump starts at one or more specified directories and recursively dumps all files and directories found there that have changed since some given base date (e.g., the last backup for an incremental dump or system installation for a full dump). Thus in a logical dump, the dump tape gets a series of carefully identified directories and files, which makes it easy to restore a particular file or directory upon request.

16. Which mechanism is used to check block consistency?

- Another area where reliability is an issue is file system consistency. Many file systems read blocks, modify them, and write them out later. If the system crashes before all the modified blocks have been written out, the file system can be left in an inconsistent state. This problem is particularly critical if some of the blocks that have not been written out are i-node blocks, directory blocks, or blocks containing the free list.
- To deal with the problem of inconsistent file systems, most computers have a utility program that checks file system consistency. For instance, UNIX has fsck and Windows has scandisk. This utility can be run whenever the system is booted, particularly after a crash. The description below tells how fsck works. Scandisk is somewhat different because it works on a different file system, but the general principle of using the file system's inherent redundancy to repair it is still valid. All file system checkers verify each file system (disk partition) independently of the other ones.

Two types of consistency checks can be made: blocks and files.

- To check for block consistency, the program builds two tables, each one containing a counter for each block, initially set to 0. The counters in the first table keep track of how many times each block is present in a file; the counters in the second table record how often each block is present in the free list (or the bitmap of free blocks).
- The program then reads all the i-nodes using a raw device, which ignores the file structure and just returns all the disk blocks starting at 0. Starting from an i-node, it is possible to build a list of all the block numbers used in the corresponding file. As each block number is read, its counter in the first table is incremented. The program then examines the free list or bitmap to find all the blocks that are not in use. Each occurrence of a block in the free list results in its counter in the second table being incremented.
- If the file system is consistent, each block will have a 1 either in the first table or in the second table, as shown in Figure 1(a). On the other hand, as a result of a crash, the tables might look like Figure 1(b), in which block 2 does not occur in either table. It will be reported as being a missing block. While missing blocks do no real harm, they waste space and thus reduce the capacity of the disk. The solution to missing blocks is straightforward: the file system checker just adds them to the free list.

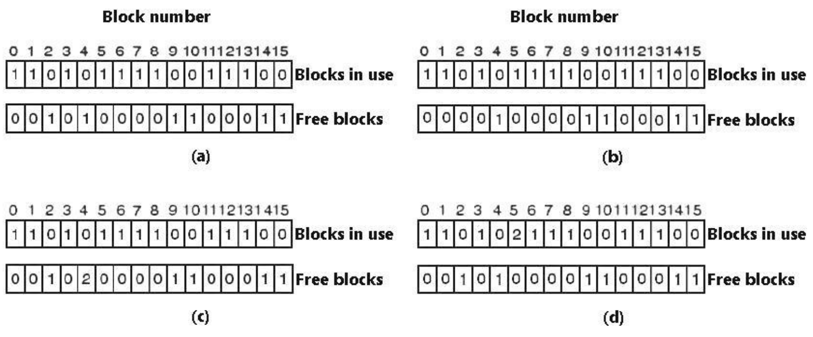
Block number
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(a)
Block number
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(b)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 1 0 2 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(c)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 2 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(d)

Figure 1. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

- Another situation that might occur is that of Figure 1(c). Here we see a block, number 4, that occurs twice in the free list. (Duplicates can occur only if the free list is really a list; with a bitmap it is impossible.) The solution here is also simple: rebuild the free list.
- The worst thing that can happen is that the same data block is present in two or more files, as shown in Figure 1(d) with block 5. If either of these files is removed, block 5 will be put on the free list, leading to a situation in which the same block is both in use and free at the same time. If both files are removed, the block will be put onto the free list twice.
- The appropriate action for the file system checker to take is to assign a free block, copy the contents of block 5 into it, and insert the copy into one of the files. Thus, the information content of the files is unchanged (although almost assuredly one is garbled), but the file system structure is at least made consistent. The error should be reported, to allow the user to inspect the damage.

17. Explain any one type of file system.
18. Explain UNIX V7 file system.
19. Write a note on CD ROM file system.
20. Write a note on MS DOS file system.

CD-ROM File Systems

- As our first instance of a file system, let us examine the file systems used on CD-ROMs. These systems are particularly simple because they were designed for write-once media. Among other things, for instance, they have no provision for keeping track of free blocks because on a CD-ROM files cannot be freed or added after the disk has been manufactured. Below we will take a look at the main CD-ROM file system type and two extensions to it. Some years after the CD-ROM made its debut, the CD-R (CD Recordable) was introduced. Unlike the CD-ROM, it is possible to add files after the initial burning, but these are simply appended to the end of the CD-R. Files are never removed (although the directory can be updated to hide existing files). As a consequence of this "append-only" file system, the essential properties are not changed. Particularly, all the free space is in one contiguous chunk at the end of the CD.

The ISO 9660 File System

- The most common standard for CD-ROM file systems was adopted as an International Standard in 1988 under the name ISO 9660. Virtually every CD-ROM currently on the market is compatible with this standard, sometimes with the extensions to be discussed below. One of the goals of this standard was to make every CD-ROM readable on every computer, independent of the byte ordering used and independent of the operating system used. As a result, some limitations were placed on the file system to make it possible for the weakest operating systems then in use (such as MS-DOS) to read it. CD-ROMs do not have concentric cylinders the way magnetic disks do. Instead there is a single continuous spiral containing the bits in a linear sequence (although seeks across the spiral are possible). The bits along the spiral are divided into logical blocks (also called logical sectors) of 2352 bytes. Some of these are for preambles, error correction, and other overhead. The payload portion of each logical block is 2048 bytes. When used for music, CDs have leadins, leadouts, and intertrack gaps, but these are not used for data CD-ROMs.

<u>Rock Ridge Extensions</u>

- As we have seen, ISO 9660 is highly restrictive in several ways. Shortly after it came out, people in the UNIX community began working on an extension to make it possible to represent UNIX file systems on a CD-ROM. These extensions were named Rock Ridge, after a town in the Gene Wilder movie Blazing Saddles, probably because one of the committee members liked the film. The extensions use the System use field in order to make Rock Ridge CD-ROMs readable on any computer. All the other fields retain their normal ISO 9660 meaning. Any system not aware of the Rock Ridge extensions just ignores them and sees a normal CD-ROM.

- The extensions are divided up into the following fields:

  1. PX - POSIX attributes.
  2. PN - Major and minor device numbers.
  3. SL - Symbolic link.
  4. NM - Alternative name.
  5. CL - Child location.
  6. PL - Parent location.
  7. RE - Relocation.
  8. TF - Time stamps.

<u>Joliet Extensions</u>

- The UNIX community was not the only group that wanted a way to extend ISO 9660. Microsoft also found it too restrictive (although it was Microsoft's own MS-DOS that caused most of the restrictions in the first place). Therefore Microsoft invented some extensions that were called Joliet. They were designed to allow Windows file systems to be copied to CD-ROM and then restored, in precisely the same way that Rock Ridge was designed for UNIX. Virtually all programs that run under Windows and use CD-ROMs support Joliet, including programs that burn CD-recordable. Generally, these programs offer a choice between the various ISO 9660 levels and Joliet.

The major extensions provided by Joliet are:

1. Long file names.
2. Unicode character set.

3. Directory nesting deeper than eight levels.
4. Directory names with extensions

The MS-DOS File System

- The MS-DOS file system is the one the first IBM PCs came with. It was the main file system up through Windows 98 and Windows ME. It is still supported on Windows 2000, Windows XP, and Windows Vista, although it is no longer standard on new PCs now except for floppy disks. However, it and an extension of it (FAT-32) have become widely used for many embedded systems. Most digital cameras use it. Many MP3 players use it exclusively. The popular Apple iPod uses it as the default file system, although knowledgeable hackers can reformat the iPod and install a different file system.
- In this way the number of electronic devices using the MS-DOS file system is vastly larger now than at any time in the past, and certainly much larger than the number using the more modern NTFS file system. For that reason alone, it is worth looking at in some detail. To read a file, an MS-DOS program must first make an open system call to get a handle for it. The open system call specifies a path, which may be either absolute or relative to the current working directory. The path is looked up component by component until the final directory is located and read into memory. It is then searched for the file to be opened.

The UNIX V7 File System

- The early versions of UNIX had a fairly sophisticated multiuser file system since it was derived from MULTICS. Below we will talk about the V7 file system, the one for the PDP-11 that made UNIX famous. The file system is in the form of a tree starting at the root directory, with the addition of links, forming a directed acyclic graph. File names are up to 14 characters and can contain any ASCII characters except / (because that is the separator between components in a path) and NUL (because that is used to pad out names shorter than 14 characters). NUL has the numerical value of 0.

- A UNIX directory entry includes one entry for each file in that directory. Each entry is very simple because UNIX uses the i-node scheme. A directory entry includes only two fields: the file name (14 bytes) and the number of the i-node for that file (2 bytes). These parameters limit the number of files per file system to 64K.