Module IV

Implementation of File System · File Concept

- o **File Attributes**
- o **File Operations**
- o **File Types**
- o **File Structure**
- o **Internal File Structure**

· **Access Methods**

- o **Sequential Access**
- o **Direct Access**
- o **Other Access Methods – Indexed Access**

· **Directory and Disk Structure**

- o **Storage Structure**
- o **Directory Overview**
- o **Single-Level Directory**
- o **Two-Level Directory**
- o **Tree –Structured Directories**
- o **Acyclic –Graph Directories**
- o **General Graph Directory**

· **File System Mounting**
· **File Sharing**
- o **Multiple Users**
- o **Remote File Systems**
  - ▪ **The Client-Server Model**
  - ▪ **Distributed Information Systems**
  - ▪ **Failure Modes**
· **Consistency Semantics**
- o **Unix Semantics**
- o **Session Semantics**
- o **Immutable – Shared Files Semantics**
· **Protection – Types, Access Control, other approaches · File-System Structure**
· **File-System Implementation**
· **Directory Implementation**

· **Allocation Methods**
· **Free-Space Management**

File Concepts

The file system consists of two distinct parts: a collection of files, each storing related data, andadirectory structure, which organizes and provides information about all the files in the system. A file is a collection of related information that is recorded on secondary storage. Afileisa sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creatorand user.

a) File Attributes

A file is named, for the convenience of its human users, and is referred to by its name. Afile's attributes vary from one operating system to another .The attributes of a file are p- o **Name** - The symbolic file name is the only information kept in human readable form. Some special significance is given to names, and particularly extensions (.exe, .txt, etc.). o **Identifier** – It is a unique number, that identifies the file within the file system. o **Type** – Type of the file like text, executable, other binary, etc.

- o **Location** - . location of the file on that device.
- o **Size** - The current size of the file (in bytes, words, or blocks)
- o **Protection -** Access-control information (reading, writing, executing). o **Time, date, and user identification** –These data can be useful for protection, security, and usage monitoring.

b) File Operations

The operating system provides system calls to create, write, read, reposition, delete, and truncate files.

- o Creating a file - Two steps are necessary to create a file
  - ▪ Find space in the file system for the file.
  - ▪ Make an entry for the new file in the directory.
- o Writing a file - To write a file, the system call consists of both the name of thefileand the information to be written to the file. Given the name of the file, thesystem searches the directory to find the file's location. The systemmust keep a *write* pointer to the location in the file where the next writeisto take place. The write pointer must be updated whenever a write occurs.
- o Reading a file - To read from a file, the system call that specifies the name of thefile and where the next block of the file should be put. The directoryis searched for the file, and the system needs to keep a *read* pointer tothe location in the file where the next read is to take place. Once the readhastaken place, the read pointer is updated.
- o Repositioning within a file - The directory is searched for the file, andthefilepointer is repositioned to a given value. This file operation is alsoknown as a file seek.
- o Deleting a file – To delete a file, search the directory for the file. Release all filespace, so that it can be reused by other files, and erase the directoryentry.

Module IV- FileSystem

· Truncating a file - The user may want to erase the contents of a file but keep its attributes. Rather than

forcing the user to delete the file and then recreate it, this functionallowsall attributes to remain unchanged –except for file length. The file size is reset tozero.

· Information about currently open files is stored in an **open file table**. It containsinformationslike:

- o **File pointer** - records the current position in the file, for the next read or writeaccess.
- o **File-open count** - How many times has the current file been opened by different processes, at the same time and not yet closed? When this counter reaches zerothe file can be removed from the table.
- o **Disk location of the file** – The information needed to locate the file ondiskiskept in memory so that the system does not have to read it fromdiskfor eachopearation.
- o **Access rights** – The file access permissions are stored on the per-process tablesothat the

operating system can allow or deny subsequent I/O requests. · Some systems provide support for **file locking.**

- o A **shared lock** is for reading only.
- o A **exclusive lock** is for writing as well as reading.
- o An **advisory lock** , it is up to the software developers to ensure that locks areacquired or realeased.
- o A **mandatory lock** , prevents any other process from accessing the locked file.. (Atruly locked door.)
  - o UNIX uses advisory locks, and Windows uses mandatory locks. **c) File Types**

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Figure 10.2 Common file types.

Module IV- FileSystem

· File name consists of two parts: name and extension
· The user and the operating system can identify the type of a file using the name. · Most operating systems allow users to specify a file name as a sequenceofcharacters followed by a period and terminated by an extension. Example: resume.doc, threads.c etc.
· The system uses the extension to indicate the type of the file and the typeof operations that can be done on that file.

· For instance, only a file with a ".corn", ".exe", or ".bat", can be executed.

d) File Structure

· The study of different ways of storing files in secondary memory such that theycanbeeasily accessed.

· File types can be used to indicate the internal structure of the file. Certain files must beina particular structure that is understood by the operating system.

· For example, the operating system requires that an executable file have a specificstructure so that it can determine where in memory to load the file and the locationof thefirst instruction.

· UNIX treats all files as sequences of bytes, with no further consideration of the internal structure.

· Macintosh files have two **forks** - a **resource fork**, and a **data fork**. The resourceforkcontains information relating to the UI, such as icons and button images. The dataforkcontains the traditional file contents-program code or data.

e) Internal File Structure

· Disk systems typically have a well-defined block size determined by the sizeof asector. A group of sectors form a group

· All disk I/0 is performed in units of one block, and all blocks are the same size. · Logical records may even vary in length. Padding a number of logical records intophysical blocks is a common solution to this problem.

· The packing can be done either by the user's application programor bytheoperating system. In either case, the file may be considered a sequence of blocks. · All the basic I/O functions operate in terms of blocks.

· Disk space is always allocated in terms of blocks. Some portion of last block(while storing a file) is always wasted. This is called internal fragmentation.

10.2 Access Methods

The file information is accessed and read into computer memory. The information in the filecanbe accessed in several ways.

Module IV- FileSystem

a) Sequential Access

· Here information in the file is processed in order, one record after the other. · This mode of access is a common method; for example, editors and compilers usuallyaccess files in this fashion.

· A sequential access file emulates magnetic tape operation, and generally supportsafew operations:

　　o read next - read a record and advance the file pointer to the next position. o write next - write a record to the end of file and advance the file pointer tothenext position.

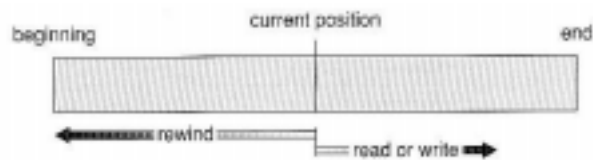　　o skip n records - May or may not be supported. 'n' may be limited to positivenumbers, or may be limited to +/- 1.

Figure 10.3 Sequential-access file.

b) Direct Access

A file is made up of fixed-length logical records that allow programs to read andwriterecords randomly. The records can be rapidly accessed in any order.

Direct access are of great use for immediate access to large amount of information. Eg : Database file. When a query occurs, the query is computed and only the selected rows areaccess directly to provide the desired information.

Operationssupported include:
- read n - read record number n. (position the cursor to n and then read the record) · write n - write record number n. (position the cursor to n and then writetherecord)
- jump to record n – move to nth record (n- could be 0 or the end of file) · If the record length is *L,* there is a request for record 'N'. Then the direct accesstothe starting byte of record 'N' is at L*(N-1)

Eg: if 3rdrecord is required and length of each record(L) is 50, then the starting position of 3rdrecord is L*(N-1)

$$Address = 50*(3-1) = 100.$$

c) Other Access Methods(Indexed method)

- These methods generally involve the construction of an ***index*** for the file called**indexfile**.

Module IV- FileSystem

- The index file is like an index page of a book, which contains key and address. Tofindarecord in the file, we first search the index and then use the pointer to access the recorddirectly and find the desired record.
- An indexed access scheme can be easily built on top of a direct access system. · For very large files, the index file itself is very large. The solution to this is to createanindex for index file. i.e. multi-level indexing.
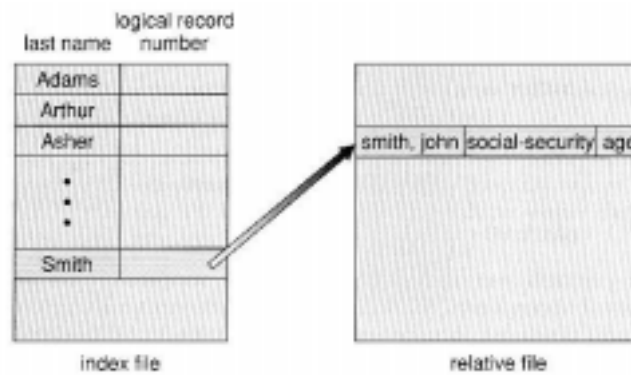
Figure 10.5 Example of index and relative files.

10.3 Directory Structure

Directory is a structure which contains filenames and information about the files likelocation, size, type etc. The files are put in different directories. Partitioning is useful for limitingthe sizes of individual file systems, putting multiple file-system types on the same device, orleaving part of the device available for other uses.

Partitions are also known as *slices* or *minidisks*. A file system can be created oneachofthese parts of the disk. Any entity containing a file system is generally known as a *volume*.
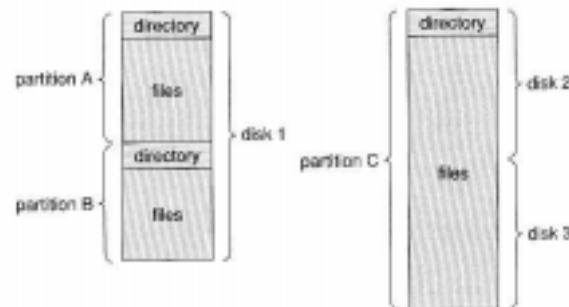

Figure 10.6 A typical file-system organization.

Module IV- FileSystem

Directory Overview

The directory can be viewed as a symbol table that translates file names into their directoryentries.

Directory operations to be supported include:

- o Search for a file - search a directory structure to find the entryfor aparticular file.
  - o Create a file – create new files and add to the directory
- o Delete a file - When a file is no longer needed, erase it fromthe directoryo List a directory - list the files in a directory and the contents of thedirectory entry.
- o Rename a file – Change the name of the file. Renaming a file mayalsoallow its position within the directory structure to be changed.

o Traverse the file system - Access every directory and every file withinadirectory structure.

Directory Structures -
a) Single-Level Directory
·It is the simplest directory structure.
· All files are contained in the same directory, which is easy to support and understand.

The limitations of this structure is that -
· All files are in the same directory must have unique names.
· Even a single user on a single-level directory may find it difficult to remember the namesof all the files as the number of files increases.

Figure 10.7 Single-level directory.

b) Two-Level Directory
· Each user gets their own directory space - *user file directory(UFD)* · File names only need to be unique within a given user's directory.
· A **master file directory(MFD)** is used to keep track of each users directory, andmust be maintained when users are added to or removed fromthe system. · When a user refers to a particular file, only his own UFD is searched. · All the files within each UFD are unique.
· To create a file for a user, the operating system searches only that user's UFDtoascertain whether another file of that name exists.

Module IV- FileSystem

· 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. Theuserdirectories themselves must be created and deleted as necessary.
· This structure **isolates** one user from another. Isolation is an advantage whentheusers are completely independent but is a disadvantage when the users *want* tocooperate on some task and to access one another's files.
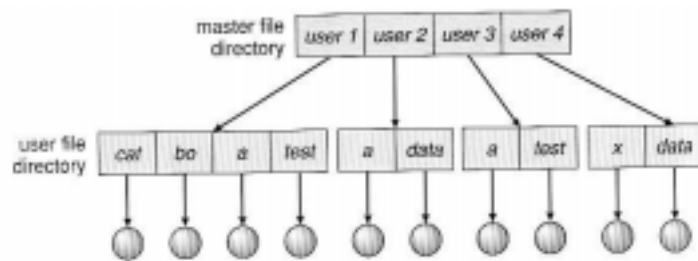
Figure 10.8 Two-level directory structure.

c) Tree-Structured Directories

- A tree structure is the most common directory structure.
- The tree has a root directory, and every file in the system has a unique path name. • A directory (or subdirectory) contains a set of files or subdirectories. • 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.
- Path names can be of two types: *absolute* and *relative.* An absolute path begins at theroot and follows a down to the specified file, giving the directory names on thepath. A relative path defines a path from the current directory.
- For example, in the tree-structured file system of figure below if the current directoryis *root/spell/mail,* then the relative path name is *prt/first* and the files absolutepathname *root/spell/mail/prt/jirst.*

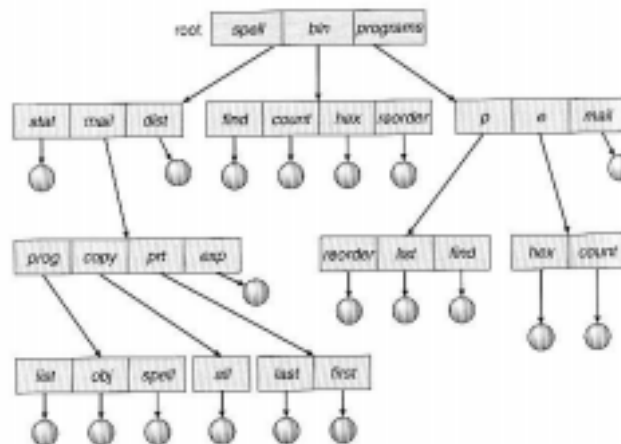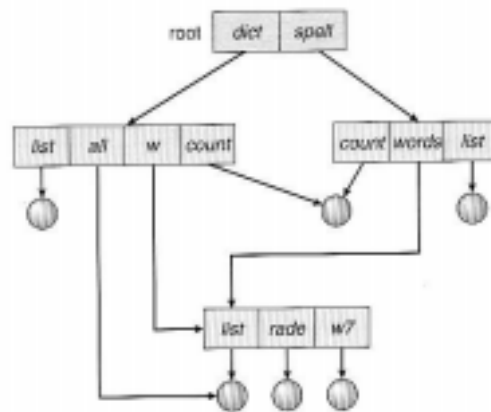**Figure 10.9** Tree-structured directory structure.

Module IV- FileSystem

- Directories are stored the same as any other file in the system, except there is abit that identifies them as directories, and they have some special structure that theOSunderstands.
- One question for consideration is whether or not to allow the removal of directoriesthat are not empty - Windows requires that directories be emptied first, andUNIXprovides an option for deleting entire sub-trees.

d) Acyclic-Graph Directories

· When the same files need to be accessed in more than one place in the directorystructure (e.g. because they are being shared by more than one user), it can be useful to provide an acyclic-graph structure. (Note the **directed** arcs fromparent tochild.)o UNIX provides two types of **links** (pointer to another file)for implementingtheacyclic-graph structure.

    o A **hard link** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in thesamefilesystem.

    o A **symbolic link**, that involves a special file, containing information about whereto find the linked file. Symbolic links may be used to link directories and/or filesin other filesystems, as well as ordinary files in the current filesystem.

· Windows only supports symbolic links, termed **shortcuts.**

· Hard links require a **reference count**, or **link count** for each file, keepingtrackofhow many directory entries are currently referring to this file. Whenever one of thereferences is removed the link count is reduced, and when it reaches zero, thediskspace can be reclaimed.

Figure 10.10 Acyclic-graph directory structure.

· For symbolic links there is some question as to what to do with the symbolic linkswhen the original file is moved or deleted:

    o One option is to find all the symbolic links and adjust them also.

Module IV- FileSystem

    o Another is to leave the symbolic links dangling, and discover that theyarenolonger valid the next time they are used.

    o What if the original file is removed, and replaced with another file havingthesame name before the symbolic link is next used?

Another approach to deletion is to preserve the file until all references to it are deleted. Toimplement this approach, we must have some mechanism for determining that thelast reference to the file has been deleted.

When a link or a copy of the directory entry is established, a new entry is added tothefile reference list. When a link or directory entry is deleted, we remove its entry on the list. Thefile is deleted when its file-reference list is empty.

e) General Graph Directory

· If cycles are allowed in the graphs, then several problems can arise: o Search algorithms can

go into infinite loops. One solution is to not followlinksinsearch algorithms. (Or not to follow symbolic links, and to only allowsymboliclinksto refer to directories)

  o Sub-trees can become disconnected from the rest of the tree and still not havetheir reference counts reduced to zero. Periodic garbage collection is requiredtodetect and resolve this problem. (chkdsk in DOS and fsck in UNIXsearch for theseproblems, among others, even though cycles are not supposed to be allowed ineithersystem. Disconnected disk blocks that are not marked as free are added backtothefile systems with made-up file names, and can usually be safely deleted.)
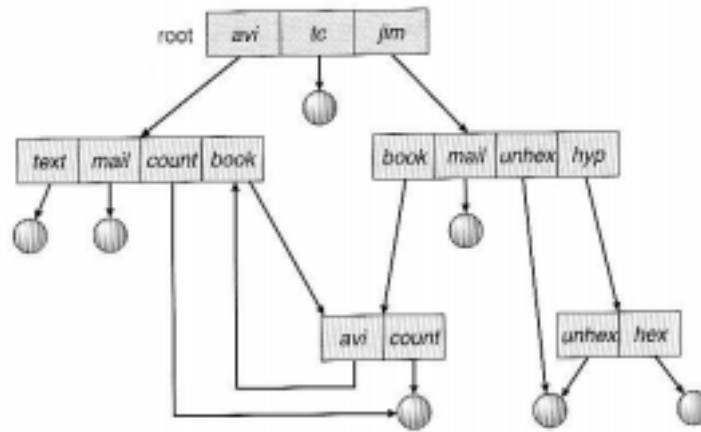

Figure 10.11 General graph directory.

Module IV- FileSystem

10.4 File-System Mounting

· The basic idea behind mounting file systems is to combine multiple file systems intoonelarge tree structure.
· The mount command is given a filesystem to mount and a **mount point** (directory) onwhich to attach it.
· Once a file system is mounted onto a mount point, any further references to that directoryactually refer to the root of the mounted file system.
· Any files (or sub-directories) that had been stored in the mount point directoryprior tomounting the new filesystem are now hidden by the mounted filesystem, andarenolonger available. For this reason some systems only allow mounting ontoemptydirectories.
· Filesystems can only be mounted by root, unless root has previously configuredcertainfilesystems to be mountable onto certain pre-determined mount points. (E.g. root mayallow users to mount floppy filesystems to /mnt or something like it) Anyone canrunthemount command to see what filesystems are currently mounted.
    · Filesystems may be mounted read-only, or have other restrictions imposed.
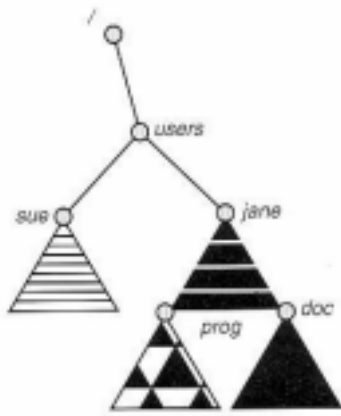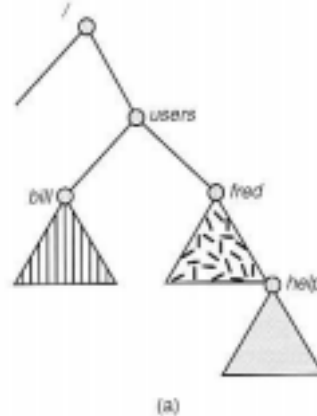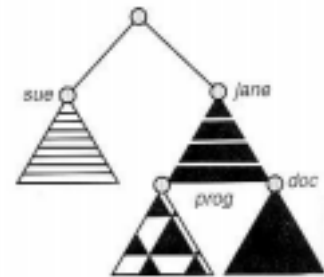
Figure 10.13 Mount point.

(a)

(b)

Figure 10.12 File system. (a) Existing system. (b) Unmounted volume.

The traditional Windows OS runs an extended two-tier directory structure, where the first tier of the structure separates volumes by drive letters, and a tree structure is implemented belowthat level.

- Macintosh runs a similar system, where each new volume that is found is automaticallymounted and added to the desktop when it is found.
- More recent Windows systems allow filesystems to be mounted to any directoryinthefilesystem, much like UNIX.

10.5 File Sharing

10.5.1 Multiple Users

- On a multi-user system, more information needs to be stored for each file: o The owner (user) who owns the file, and who can control its access. o The group of other user IDs that may have some special access to the file. o What access rights are afforded to the owner (**U**ser), the **G**roup, and to the rest ofthe world (the universe, a.k.a. **O**thers.)
  o Some systems have more complicated access control, allowing or denyingspecific accesses to specifically named users or groups.

10.5.2 Remote File Systems

- The advent of the Internet introduces issues for accessing files stored onremotecomputers
  o The original method was ftp, allowing individual files to be transportedacrosssystems as needed. Ftp can be either account and password controlled, or **anonymous**, not requiring any user name or password.
  o Various forms of **distributed file systems** allow remote file systems to be mountedonto a local directory structure, and accessed using normal file access commands. (The actual files are still

transported across the network as needed, possibly usingftpas the underlying transport mechanism.)
- o The WWW has made it easy once again to access files on remote systems without mounting their filesystems, generally using (anonymous) ftp as the underlyingfile transport mechanism.

a) The Client-Server Model

- · When one computer system remotely mounts a filesystemthat is physicallylocated on another system, the system which physically owns the files actsasa **server**, and the system which mounts them is the **client.**
- · User IDs and group IDs must be consistent across both systems for the systemtowork properly. (I.e. this is most applicable across multiple computers managedby the same organization, shared by a common group of users.)
- · The same computer can be both a client and a server. (E.g. cross-linkedfilesystems.)
- · There are a number of security concerns involved in this model:
  - o Servers commonly restrict mount permission to certain trusted systems only. Spoofing (a computer pretending to be a different computer) isapotential security risk.
  - o Servers may restrict remote access to read-only.
    - o Servers restrict which filesystems may be remotely mounted. Generallythe information within those subsystems is limited, relatively public, andprotected by frequent backups.

Module IV- FileSystem

- · The NFS (Network File System) is a classic example of such a system. **b) Distributed**

Information Systems

- · The **Domain Name System, DNS,** provides for a unique naming systemacrossall of the Internet.
- · Domain names are maintained by the **Network Information System, NIS**, whichunfortunately has several security issues. NIS+ is a more secure version, but hasnot yet gained the same widespread acceptance as NIS.
- · Microsoft's **Common Internet File System, CIFS**, establishes a **networklogin** for each user on a networked system with shared file access. OlderWindows systems used **domains**, and newer systems (XP, 2000), use **activedirectories.** User names must match across the network for this systemtobevalid.
- · A newer approach is the **Lightweight Directory-Access Protocol, LDAP,** whichprovides a **secure single sign-on** for all users to access all resources onanetwork. This is a secure system which is gaining in popularity, and whichhasthe maintenance advantage of combining authorization informationinone central location.

c) Failure Modes

- · When a local disk file is unavailable, the result is generally known immediately, and is generally non-recoverable. The only reasonable response is for the responseto fail.
- · However when a remote file is unavailable, there are many possible reasons, andwhether or not

it is unrecoverable is not readily apparent. Hence most remoteaccess systems allow for blocking or delayed response, in the hopes that theremote system (or the network) will come back up eventually.

10.5.3 Consistency Semantics

· **Consistency Semantics** deals with the consistency between the views of sharedfiles on a networked system. When one user changes the file, when do other userssee the changes?
· The series of accesses between the open() and close() operations of a file is calledthe **file session**.

Examples of consistency semantics -

a) UNIX Semantics

· The UNIX file system uses the following semantics:
  o Writes to an open file are immediately visible to any other user who has thefileopen.

  o One implementation uses a shared location pointer, which is adjustedfor all sharing users.
· There is a single copy of the file, which may delay some accesses.

b) Session Semantics

· The Andrew File System, AFS uses the following semantics:
  o Writes to an open file are not immediately visible to other users.
  o When a file is closed, any changes made become available only to users whoopenthe file at a later time.
· According to these semantics, a file can be associated with multiple (possibly different)views. Almost no constraints are imposed on scheduling accesses. No user is delayedin reading or writing their personal copy of the file.
· AFS file systems may be accessible by systems around the world. Access control ismaintained through (somewhat) complicated access control lists, which maygrant access to the entire world (literally) or to specifically named users accessingthefiles from specifically named remote environments.

c) Immutable-Shared-Files Semantics

· Under this system, when a file is declared as **shared** by its creator, then the namecannot be re-used by any other process and it cannot be modified.

10.6 Protection

The information in a computer system must be stored safely without any physical damage(theissue of *reliability)* and improper access (the issue of *protection).*
Reliability is generally provided by duplicate copies of files. Many computers havesystems programs that automatically copy disk files to tape at regular intervals (once per dayorweek or month). The damage could be

due to hardware problems (such as errors in readingorwriting), power surges or failures, head crashes, dirt, temperature extremes, and vandalism.

10.6.1 Types of Access
Systems that do not permit access to the files of other users do not need protection. Thus, wecould provide complete protection by prohibiting access. Alternatively, we could providefreeaccess with no protection. Several different types of operations may be controlled: **· Read-** Read from the file.
· **Write-** Write or rewrite the file.
· **Execute** - Load the file into memory and execute it.
· **Append-** Write new information at the end of the file.
· **Delete** - Delete the file and free its space for possible reuse.
· **List** - List the name and attributes of the file.

Module IV- FileSystem

10.6.2 Access Control

To make access to files depending on the identity of the user. Different users may need different types of access to a file or directory. The most general scheme to implement dependent accessisto associate with each file and directory an access-control list (ACL) specifying user names andthe types of access allowed for each user. When a user requests access to a particular file, theoperating system checks the access list associated with that file. If that user is listedfor therequested access, the access is allowed. Otherwise, a protection violation occurs, and the user jobis denied access to the file.

If we want to allow everyone to read a file, we must list all users with read access. Thistechnique has two undesirable consequences:
· Constructing such a list may be a tedious, if we do not know in advance the list of usersin the system.
· The directory entry, must be of variable size, as the list grows, resulting inmorecomplicated space management.

These problems can be resolved by use of a condensed version of the access list. To condensethelength of the access-control list, many systems recognize three classifications of usersinconnection with each file:
· **Owner-** The user who created the file is the owner.
· **Group -** A set of users who are sharing the file and need similar access is a group, or work group.
· **Universe-** All other users in the system constitute the universe.
The most common recent approach is to combine access-control lists with owner, group, anduniverse access control scheme.

10.6.3 Other Protection Approaches
Associate a password with each file.
Using of password is effective, but has a few disadvantages:
· The number of passwords that a user needs to remember maybe large · If one password is used, then once the password is discovered, all the files canbeaccessed.

Some system allow users to associate a password to a subdirectory, rather than only to file.

10.7 File-System Structure

Disks provide the bulk of secondary storage on which a file system is maintained. Thetwocharacteristics that make them a convenient medium for storing multiple files: 1. A disk can be rewritten in place; it is possible to read a block fromthe disk, modifytheblock, and write it back into the same place.
2. A disk can access directly any given block of information it contains. Thus, it is simpletoaccess any file either sequentially or randomly, and switching fromone file toanotherrequires only moving the read-write heads and waiting for the disk to rotate.

· To improve I/O efficiency, I/O transfers between memory and disk are performedinunitsof *blocks.* Block sizes may range from 512 bytes to 4K or larger.(Rather thantransferring a byte at a time,)
· A file system poses two quite different design problems.
  · The first problem is defining how the file system should look to the user. Thistask involves defining a file and its attributes, the operations allowed on a file, andthe directory structure for organizing files.
  · The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

· File systems organize storage on disk drives, and can be viewed as a layered design: ○ At the lowest layer are the physical devices, consisting of the magnetic media, motors & controls, and the electronics connected to them and controllingthem. Modern disk put more and more of the electronic controls directly on the diskdriveitself, leaving relatively little work for the disk controller card to perform. ○ Lowest level*, I/O Control* consists of ***device drivers***, which communicate withthedevices by reading and writing special codes directly to and frommemory addressescorresponding to the controller card's registers. Each controller card (device) onasystem has a different set of addresses (registers, ***ports***) that it listens to, andaunique set of command codes and results codes that it understands.
  ○ The ***basic file system*** level works directly with the device drivers in termsofretrieving and storing raw blocks of data, without any consideration for what isineach block.
  ○ The ***file organization module*** knows about files and their logical blocks, andhowthey map to physical blocks on the disk. In addition to translating fromlogical tophysical blocks, the file organization module also maintains the list of free blocks, and allocates free blocks to files as needed.
  ○ The ***logical file system*** deals with all of the meta data associated with a file (UID, GID, mode, dates, etc), i.e. everything about the file except the data itself. This level manages the directory structure and the mapping of file names to ***file control blocks, FCBs***, which contain all of the meta data as

well as block number information for
finding the data on the disk.
· The layered approach to file systems means that much of the code can be used uniformly for a wide variety of different file systems, and only certain layers need to be filesystem specific.

· When a layered structure is used for file-system implementation, duplication of code is minimized. The I/O control and sometimes the basic file-system code can be used by multiple file systems. · Common file systems in use include the UNIX file system, UFS, the Berkeley Fast File System, FFS, Windows systems FAT, FAT32, NTFS, CD-ROM systems ISO 9660, and for Linux the extended filesystems ext2 and ext3 .
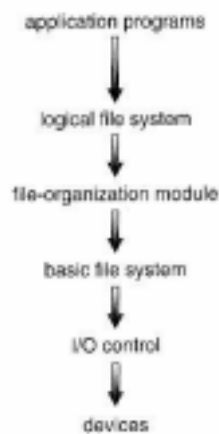

Figure 11.1 Layered file system.

10.8 File-System Implementation

On disk, the file system may contain information about how to boot an operating system stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.

File systems store several important data structures on the disk:

· A **boot-control block**, (per volume) can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty. It is typically the first block of a volume In UFS, it is called the **boot block**; in NTFS, it is the partition **boot sector.**
· A **volume control block,** (per volume) contains volume (or partition) details, such as the number of blocks in the partition, size of the blocks, free block count and free-block pointers, and free FCB count and FCB pointers. In UFS, this is called a **superblock;** in NTFS, it is stored in. the **master file table.**
   o A directory structure (per file system), containing file names and pointers to corresponding FCBs. UNIX uses inode numbers, and NTFS uses a **master filetable.**
   o The **File Control Block, FCB,** (per file) containing details about ownership, size, permissions, dates, etc. UNIX stores this information in inodes, and NTFS in the master file table as a relational database structure.

Module IV- FileSystem

· There are also several key data structures stored in memory:
- o An in-memory mount table contains information about each mounted volume.. o An in-memory directory cache of recently accessed directory information. o *A system-wide open file table*, containing a copy of the FCB for every currentlyopen file in the system, as well as some other related information.
- o *A per-process open file table,* containing a pointer to the systemopen file tableaswell as some other information. (For example the current file positionpointermay be either here or in the system file table, depending on the implementationand whether the file is being shared or not.)
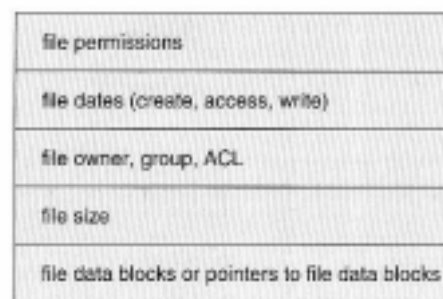

| file permissions |
| file dates (create, access, write) |
| file owner, group, ACL |
| file size |
| file data blocks or pointers to file data blocks |

Figure 11.2 A typical file-control block.

· Figure 11.3 illustrates some of the interactions of file system components when files arecreated and/or used:
- o When a new file is created, a new FCB is allocated and filled out with important information regarding the new file.
- o When a file is accessed during a program, the open( ) systemcall reads intheFCB information from disk, and stores it in the system-wide open file table. Anentry is added to the per-process open file table referencing the system-widetable, and an index into the per-process table is returned by the open( ) systemcall. UNIX refers to this index as a *file descriptor*, and Windows refers to it as a*filehandle*.
- o If another process already has a file open when a new request comes infor thesame file, and it is sharable, then a counter in the system-wide tableisincremented and the per-process table is adjusted to point to the existingentryinthe system-wide table.
- o When a file is closed, the per-process table entry is freed, and the counter inthesystem-wide table is decremented. If that counter reaches zero, then the systemwide table is also freed. Any data currently stored in memory cache for this fileiswritten out to disk if necessary.
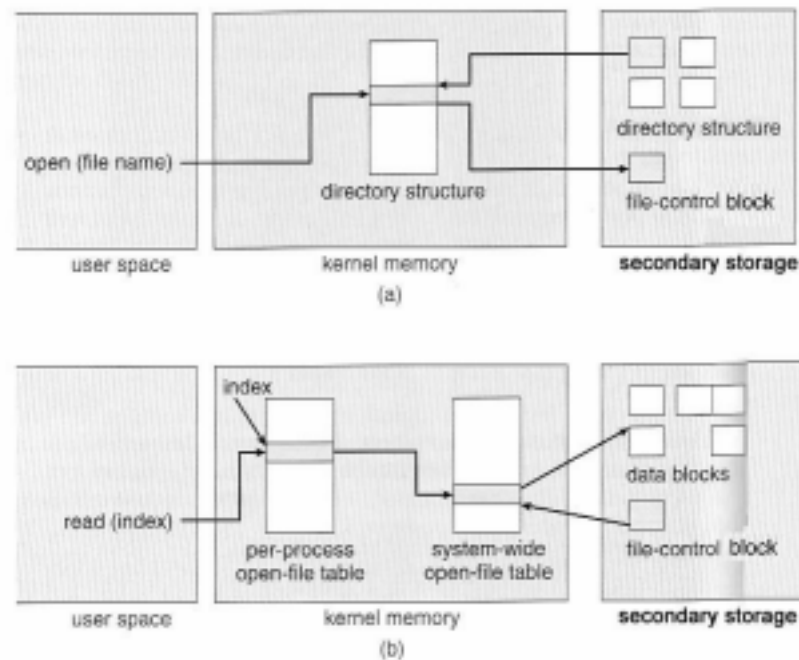
Figure 11.3 In-memory file-system structures. (a) File open. (b) File read.

10.8.1 Partitions and Mounting

· Physical disks are commonly divided into smaller units called partitions. Theycanalso be combined into larger units, but that is most commonly done for RAIDinstallations and is left for later chapters.

· Partitions can either be used as raw devices (with no structure imposeduponthem), or "cooked;' containing a file system. they can be formatted toholdafilesystem (i.e. populated with FCBs and initial directory structures asappropriate.) Raw partitions are generally used for swap space, and mayalsobeused for certain programs such as databases that choose to manage their owndiskstorage system. Partitions containing filesystems can generally only be accessedusing the file system structure by ordinary users, but can often be accessedasaraw device also by root.

· Boot information can be stored in a separate partition. Again, it has its ownformat, because at boot time the system does not have file-systemdevice driversloaded and therefore cannot interpret the file-system format.

· The boot block is accessed as part of a raw partition, by the boot programprior toany operating system being loaded. Modern boot programs understand multipleOSes and filesystem formats, and can give the user a choice of which of several available systems to boot.

· The ***root partition*** contains the OS kernel and at least the key portions of theOSneeded to complete the boot process. At boot time the root partition is mounted, and control is transferred from the boot program to the kernel found there. (Oldersystems required that the root partition lie completely within the first 1024cylinders of the disk, because that was as far as the boot programcouldreach.

Module IV- FileSystem

Once the kernel had control, then it could access partitions beyond the 1024cylinder boundary.)
· Continuing with the boot process, additional filesystems get mounted, addingtheir information into the appropriate mount table structure. As a part of themounting process the file systems may be checked for errors or inconsistencies, either because they are flagged as not having been closed properly the last timethey were used, or just for general principals. Filesystems may be mountedeitherautomatically or manually. In UNIX a mount point is indicated by settinga flaginthe in-memory copy of the inode, so all future references to that inode get re-directed to the root directory of the mounted filesystem.

10.8.2 Virtual File Systems

· ***Virtual File Systems, VFS***, provide a common interface to multiple different filesystem types. In addition, it provides for a unique identifier (vnode) for filesacross the entire space, including across all filesystems of different types. (UNIXinodes are unique only across a single filesystem, and certainly do not carryacross networked file systems.)
· The VFS in Linux is based upon four key object types:
   o The ***inode*** object, representing an individual file
   o The ***file*** object, representing an open file.
   o The ***superblock*** object, representing a filesystem.
   o The ***dentry*** object, representing an individual directory entry.
· Linux VFS provides a set of common functionalities for each filesystem, usingfunction pointers accessed through a table. The same functionality is accessedthrough the same table position for all filesystem types, though the actual functions pointed to by the pointers may be filesystem-specific. Common
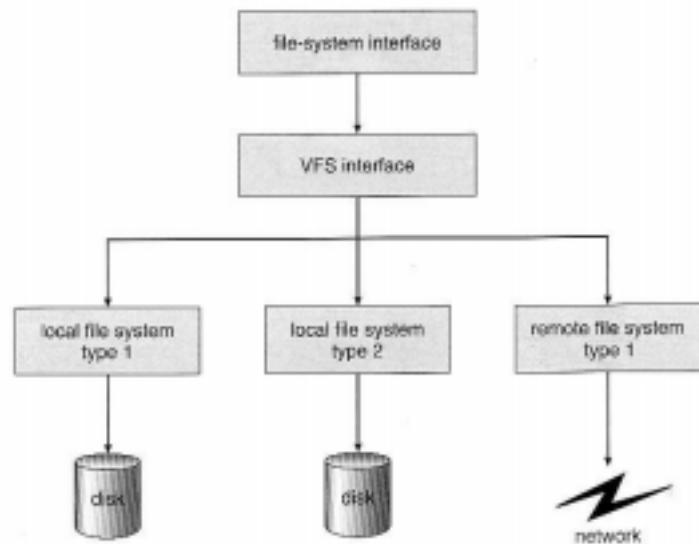   operations provided include open( ), read( ), write( ), and mmap( ).

Figure 11.4 Schematic view of a virtual file system.

Module IV- FileSystem

Figure 11.4. The first layer is the file-system interface, based on the open(), read(), write(), andclose() calls and on file descriptors. The second layer is called the virtual file system(VFS)layer; it serves two important functions:

1. It separates file-system-generic operations from their implementation by definingacleanVFS interface. Several implementations for the VFS interface may coexist on thesamemachine, allowing transparent access to different types of file systems mounted locally.
2. The VFS provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a vnode, that containsanumerical designator for a network-wide unique file. (UNIX inodes are unique withinonly a single file system.) This network-wide uniqueness is required for support ofnetwork file systems.

The kernel maintains one vnode structure for each active node (file or directory).

10.9 Directory Implementation

The selection of directory-allocation and directory-management algorithms significantlyaffectsthe efficiency, performance, and reliability of the file system. Directories need to be fast tosearch, insert, and delete, with a minimum of wasted disk space.

a) Linear List

- · A linear list is the simplest and easiest directory structure to set up, but it does havesome drawbacks.
- · The disadvantage of a linear list of directory entries is that finding a file requiresalinear search.
- · To overcome this, a software cache is implemented to store the recently accesseddirectory structure.
- · Deletions can be done by moving all entries, flagging an entry as deleted, or bymoving the last entry into the newly vacant position.

- A sorted list allows a binary search and decreases the average search time. However, the requirement that the list be kept sorted may complicate creating and deletingfiles, • A linked list makes insertions and deletions into a sorted list easier, with overheadforthe links.
- An advantage of the sorted list is that a sorted directory listing can be producedwithout a separate sort step.

b) Hash Table

- With this method, a linear list stores the directory entries, but a hash data structure is alsoused.
- The hash table takes a value computed from the file name and returns a pointer tothefilename in the linear list. Therefore, it can greatly decrease the directory search time.

Module IV- FileSystem

- Here **collisions** may occur. Collision is the situation where two file names hashtothesame location.
- Alternatively, a chained-overflow hash table can be used. Each hash entry can be a linkedlist instead of an individual value, and we can resolve collisions by adding the newentryto the linked list.
- The major disadvantage with a hash table are its generally fixed size and the dependenceof the hash function on that size. For example, assume that we make a linear-probinghash table that holds 64 entries. The hash function converts file names into integers from0 to 63, for instance, by using the remainder of a division by 64. If we later try tocreatea65th file, we must enlarge the directory hash table—say, to 128 entries. As a result, weneed a new hash function that must map file names to the range 0 to 127, and wemust reorganize the existing directory entries to reflect their new hash-function values.

10.10 Allocation Methods

The main problem is how to allocate space to these files so that disk space is utilized effectivelyand files can be accessed quickly. Three major methods of allocating disk space are in wideuse: contiguous, linked, and indexed.

a) Contiguous Allocation

- *Contiguous Allocation* requires that all blocks of a file be kept togethercontiguously.
- Performance is very fast, because reading successive blocks of the samefilegenerally requires no movement of the disk heads, or at most one small steptothenext adjacent cylinder.
- Storage allocation is done by using one of the algorithms(first fit, best fit, worst fit).
- The allocation of blocks contiguous leads to external fragmentation.
- Problems can arise when files grow, or if the exact size of a file is unknownat creation time:
    - Over-estimation of the file's final size increases external fragmentation and wastes disk space.
    - Under-estimation may require that a file be moved or a process abortedifthe file grows beyond its originally allocated space.

o If a file grows slowly over a long time period and the total final spacemust be allocated initially, then a lot of space becomes unusable beforethefile fills the space.
· A variation is to allocate file space in large contiguous chunks, called *extents.* When a file outgrows its original extent, then an additional oneblock is allocated. A pointer points from last block of contiguous memory allocation to the extended chunk.

Module IV- FileSystem
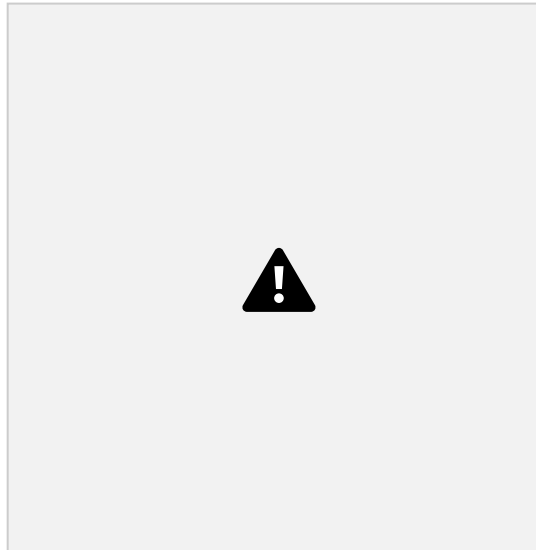
b) Linked Allocation

· Disk files can be stored as linked lists, with the expense of the storage spaceconsumed by each link. (E.g. a block may be 508 bytes instead of 512.) · Linked allocation involves no external fragmentation, does not require pre-knownfile sizes, and allows files to grow dynamically at any time.
· Unfortunately linked allocation is only efficient for sequential access files, asrandom access requires starting at the beginning of the list for each newlocationaccess.
· Allocating *clusters* of blocks reduces the space wasted by pointers, at the cost ofinternal fragmentation.
· Another big problem with linked allocation is reliability if a pointer is lost ordamaged. Doubly linked lists provide some protection, at the cost of additional overhead and wasted space.
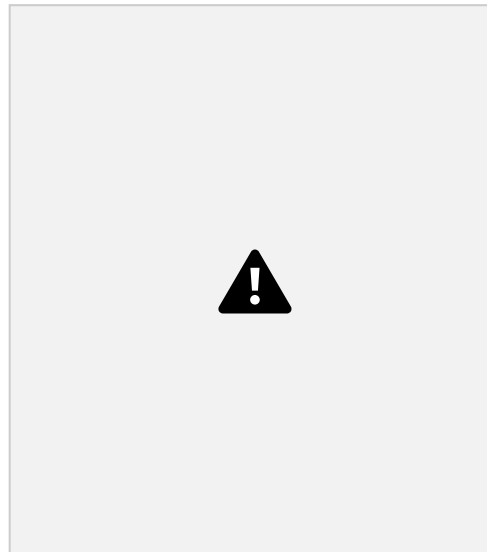
Module IV- FileSystem

- The ***File Allocation Table, FAT,*** used by DOS is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the disk. Thebenefit of this approach is that the FAT table can be cached in memory, greatlyimproving random access speeds.
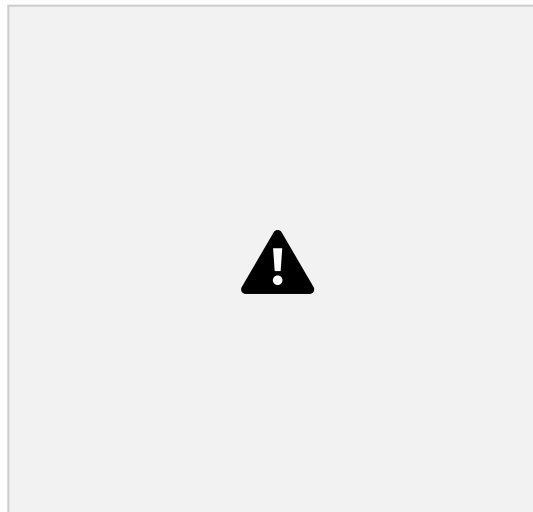
c) Indexed Allocation

- ***Indexed Allocation*** combines all of the indexes(block numbers) for accessingeach file into a common block (for that file).
- Each file will have a common block called the index block.
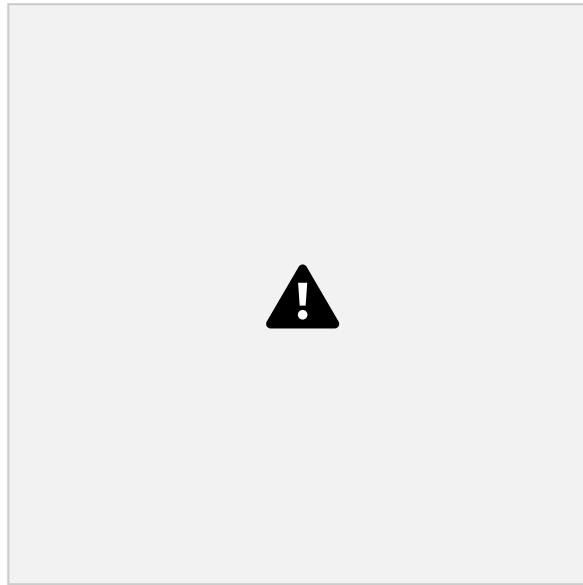
∙ Some disk space is wasted (relative to linked lists or FAT tables) becauseanentire index
block must be allocated for each file, regardless of howmanydata
blocks the file contains. This leads to questions of how big the index blockshouldbe, and
how it should be implemented. There are several approaches:

Module IV- FileSystem

- o **Linked Scheme -** An index block is one disk block, which canbereadand written in a
single disk operation. The first index block contains someheader information, the first N
block addresses, and if necessary a pointerto additional linked index blocks.
  - o **Multi-Level Index -** The first index block contains a set of pointersto
secondary index blocks, which in turn contain pointers to the actual data
blocks.
  - o **Combined Scheme -** This is the scheme used in UNIXinodes, inwhich
the first 12 entries data block pointers are stored directly in the inode, andthen
singly, doubly, and triply indirect pointers provide access tomoredata blocks as
needed. The advantage of this scheme is that for small files
(files stored in less than 12 blocks), the data blocks are readily accessible(up to 48K
with 4K block sizes); files up to about 4144K(using4K
blocks) are accessible with only a single indirect block (whichcanbe
cached), and huge files are still accessible using a relatively small numberof disk
accesses (larger in theory than can be addressed by a 32-bit address, which is why some
systems have moved to 64-bit file pointers.)

Performance

· The optimal allocation method is different for sequential access files thanforrandom access files, and is also different for small files than for large files. · Some systems support more than one allocation method, which mayrequirespecifying how the file is to be used (sequential or random access) at the timeit is allocated. Such systems also provide conversion utilities.
· Some systems have been known to use contiguous access for small files, andautomatically switch to an indexed scheme when file sizes surpass a certainthreshold.

Module IV- FileSystem

· And of course some systems adjust their allocation schemes (e.g. block sizes) tobest match the characteristics of the hardware for optimum performance.

10.11 Free-Space Management:

The space created after deleting the files can be reused. Another important aspect of diskmanagement is keeping track of free space in memory. The list which keeps track of freespacein memory is called the free-space list. To create a file, search the free-space list for the requiredamount of space and allocate that space to the new file. This space is then removed fromthefree-space list. When a file is deleted, its disk space is added to the free-space list. The free-spacelist, is implemented in different ways as explained below.

a) Bit Vector

· Fast algorithms exist for quickly finding contiguous blocks of a given size · One simple approach is to use a *bit vector*, in which each bit represents adiskblock, set to 1 if free or 0 if allocated.

For example, consider a disk where blocks 2,3,4,5,8,9, 10,11, 12, 13, 17and 18 are free, andthe rest of the blocks are allocated. The free-space bit map would be 0011110011111100011

· Easy to implement and also very efficient in finding the first free blockor 'n'consecutive free blocks on the disk.
· The down side is that a 40GB disk requires over 5MB just to store the bitmap.

b) Linked List
· A linked list can also be used to keep track of all free blocks.
· Traversing the list and/or finding a contiguous block of a given size are not easy, but fortunately are not frequently needed operations. Generally the systemjust adds and removes single blocks from the beginning of the list.
· The FAT table keeps track of the free list as just one more linked list on the table.

Module IV- FileSystem
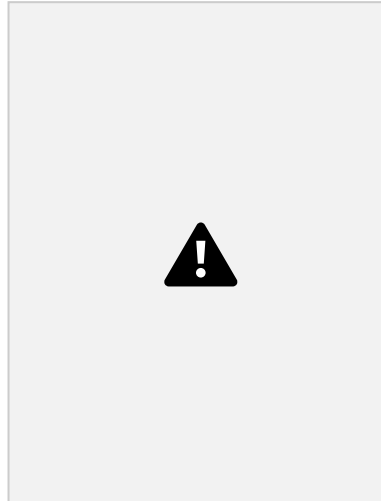
c) Grouping

· A variation on linked list free lists. It stores the addresses of n free blocks inthefirst free block. The first n-1 blocks are actually free. The last block contains theaddresses of another n free blocks, and so on.
· The address of a large number of free blocks can be found quickly. **d) Counting**

· When there are multiple contiguous blocks of free space then the systemcankeeptrack of the starting address of the group and the number of contiguous freeblocks.
· Rather than keeping al list of n free disk addresses, we can keep the addressoffirst free block and the number of free contiguous blocks that followthefirst block.
· Thus the overall space is shortened. It is similar to the extent method of allocatingblocks.

e) Space Maps (New)

· Sun's ZFS file system was designed for huge numbers and sizes of files, directories, and even file systems.
· The resulting data structures could be inefficient if not implemented carefully. For example, freeing up a 1 GB file on a 1 TB file system could involve updatingthousands of blocks of free

list bit maps if the file was spread across the disk.

· ZFS uses a combination of techniques, starting with dividing the diskupinto(hundreds of) *metaslabs* of a manageable size, each having their own space map. · Free blocks are managed using the counting technique, but rather than writetheinformation to a table, it is recorded in a log-structured transactionrecord. Adjacent free blocks are also coalesced into a larger single free block.

· An in-memory space map is constructed using a balanced tree data structure, constructed from the log data.

· The combination of the in-memory tree and the on-disk log provide for veryfast and efficient management of these very large files and free blocks.