4.5 DIRECTORY AND DISK STRUCTURE

- Directory & disk structure deals with how to store files. A disk can be **partitioned** into quarters, and each quarter can hold a separate file system.
- Partitioning is useful for limiting the sizes of individual file systems, putting multiple file-system types on the same device.
- Any entity containing a file system is generally known as a **volume**.
- Each volume that contains a file system must also contain information about the files in the system.
- This information is kept in entries in a **device directory** or **volume table of contents**.
- The device directory (more commonly known simply as the **directory**) records information—such as name, location, size, and type—for all files on that volume.

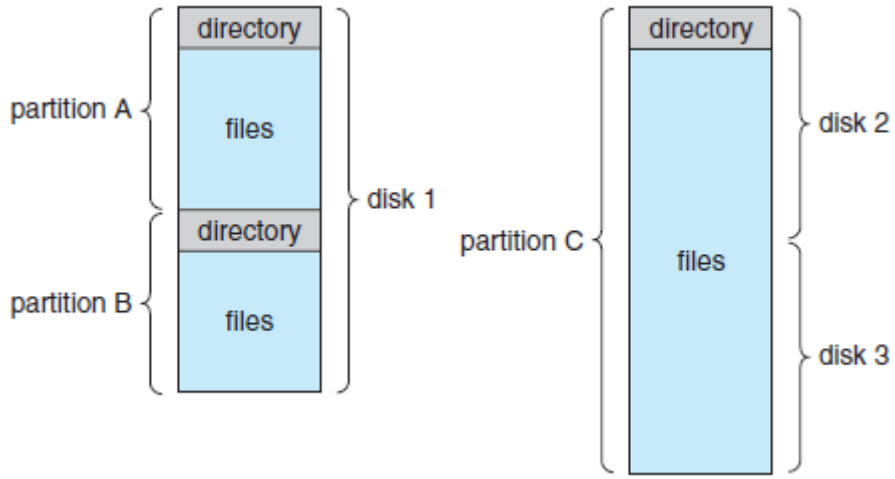


Fig: A typical file-system organization

Storage Structure

- A general-purpose computer system has multiple storage devices, and those devices can be sliced up into volumes that hold file systems.
- Computer systems may have zero or more file systems, and the file systems may be of varying types.
- For example, a typical Solaris system may have dozens of file systems of a dozen different types,

Consider the types of file systems in the Solaris

- **tmpfs**—a "temporary" file system that is created in volatile main memory and has its contents erased if the system reboots or crashes
- **lofs**—a "loop back" file system that allows one file system to be accessed in place of another one
- **procfs**—a virtual file system that presents information on all processes as a file system
- **ufs, zfs**—general-purpose file systems

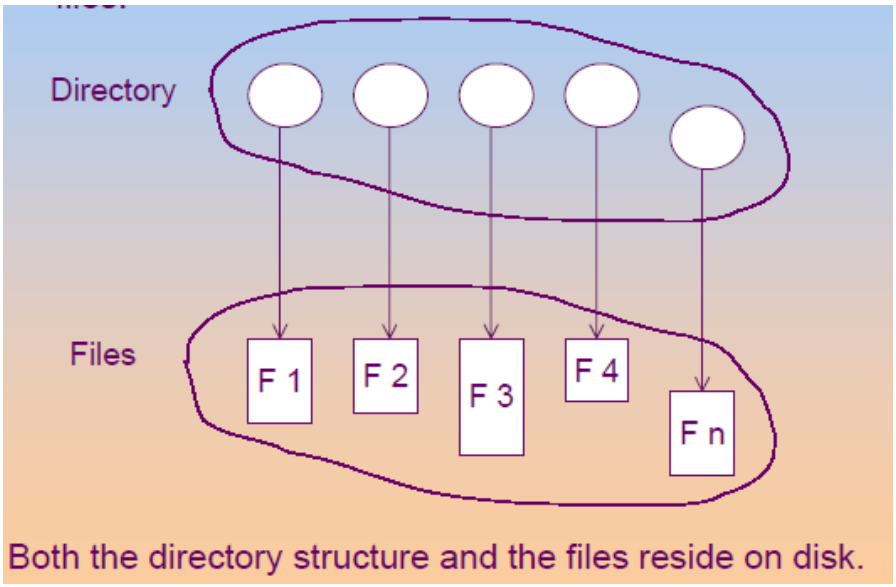
Directory Overview

Directory operations to be supported include:

- Search for a file
- Create a file - add to the directory
- Delete a file - erase from the directory
- List a directory - possibly ordered in different ways.
- Rename a file - may change sorting order
- Traverse the file system.

Directory Structure

A collection of nodes containing information about all files.



There are five directory structures. They are

1. Single-level directory

2. Two-level directory

3. Tree-Structured directory

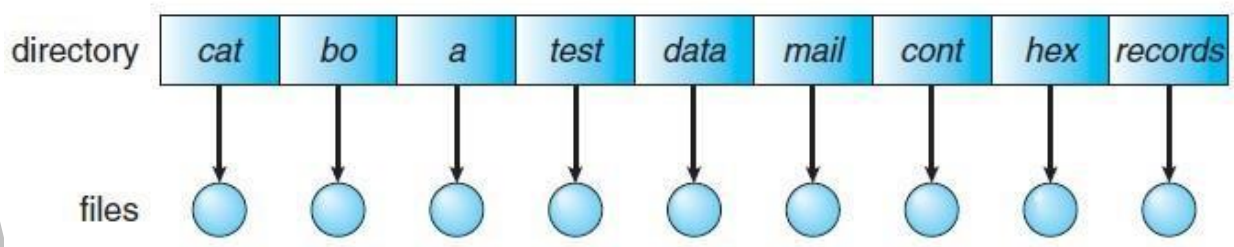
4. Acyclic Graph directory

5. General Graph directory

1. Single – Level Directory

- The simplest directory structure is the single- level directory.
- All files are contained in the same directory.
- Disadvantage:

When the number of files increases or when the system has more than one user, since all files are in the same directory, they must have unique names.

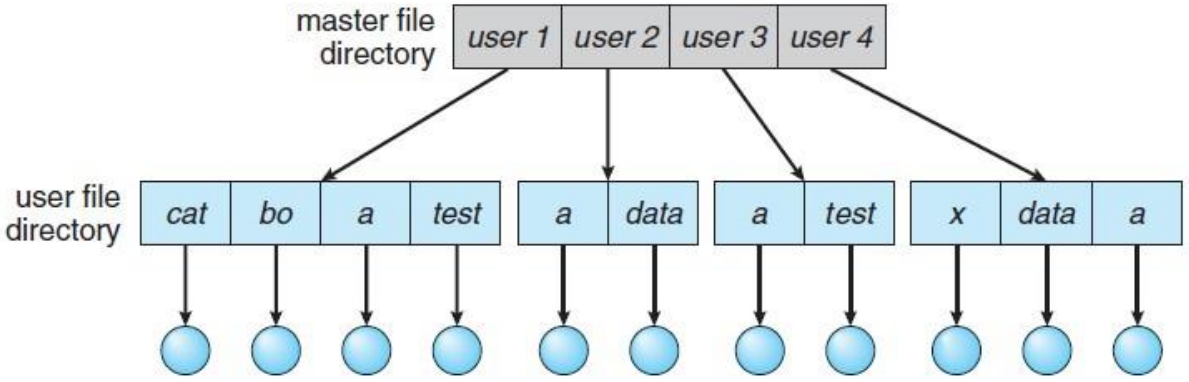


2. Two – Level Directory

- Separate directory for each user.
- In the two level directory structures, each user has her own user file directory (UFD).
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched.
- Thus, different users may have files with the same name.
- The two – level directory structure solves the name-collision problem

Disadvantage:

- Users cannot create their own sub-directories.

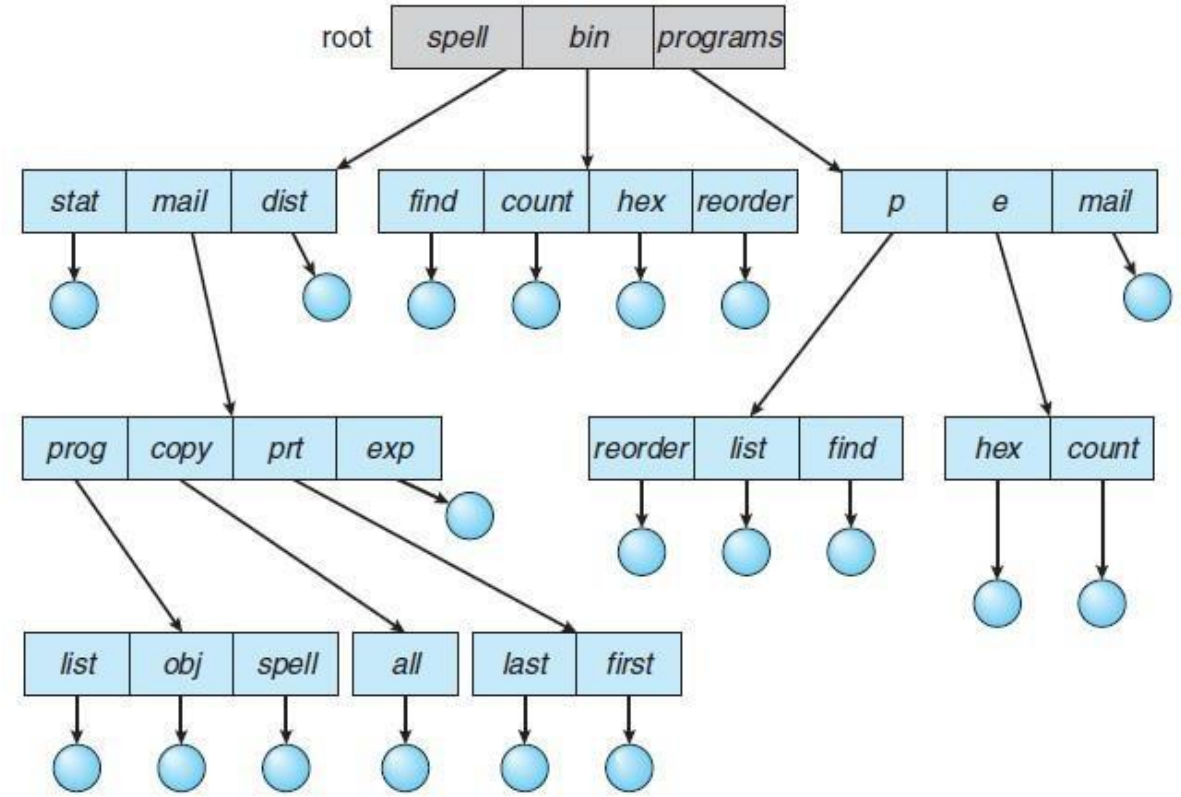


Path Name

- If a user can access another user's files, the concept of path name is needed.
- In two-level directory, this tree structure has MFD as root of path through UFD to user file name at leaf.
- Path name :: username + filename
- Standard syntax -- /user/file.ext

3. Tree – Structured Directory

- A tree is the most common directory structure.
- The tree has a root directory. Every file in the system has a unique path name.
- A **path name** is the path from the root, through all the subdirectories to a specified file.
- A directory (or sub directory) contains a set of files or sub directories.
- 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 path names or relative path names.
- An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A relative path name defines a path from the current directory.



4. Acyclic Graph Directory.

An acyclic graph is a graph with no cycles.

- To implement shared files and subdirectories this directory structure is used.
- An acyclic – graph directory structure is more flexible than is a simple tree structure, but it is also more complex.

Implementations of shared files or directories

1. Links

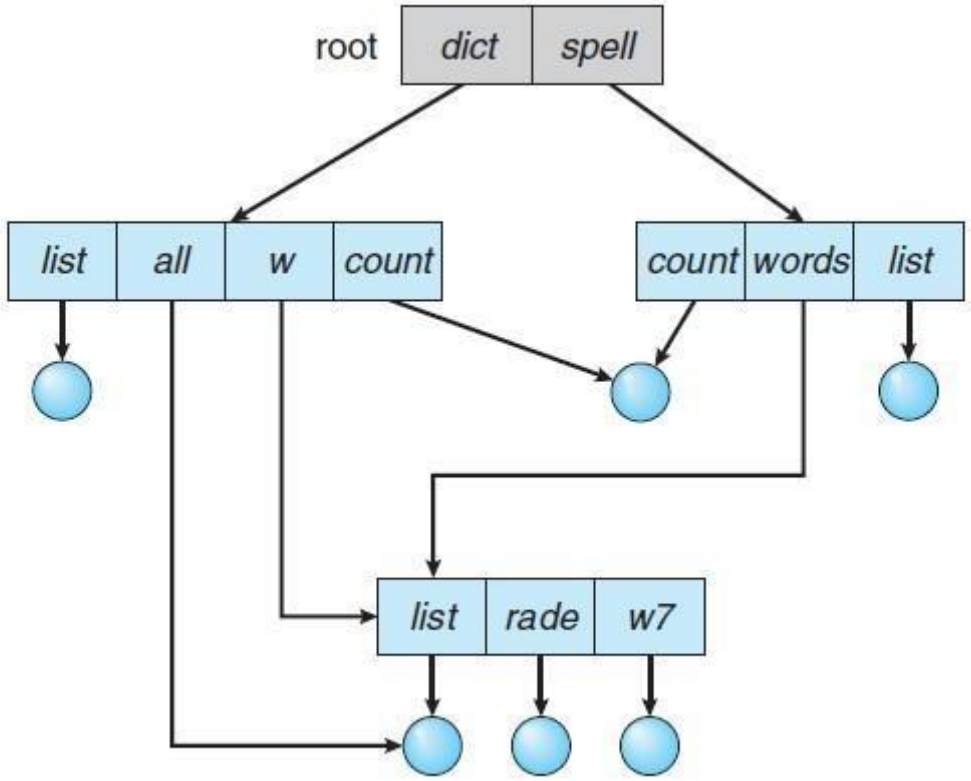
- A new type of directory entry is created. It is effectively a pointer to another file or subdirectory
- Links are implemented as an absolute or relative path name.
- The deletion of a link does not need to affect the original file; only the link is removed.

2. Duplicate all information in sharing directories

Problems to consider with link implementation:

- Upon traversal of file system, do not want to traverse shared structures more than once

- On deletion



Deletion:

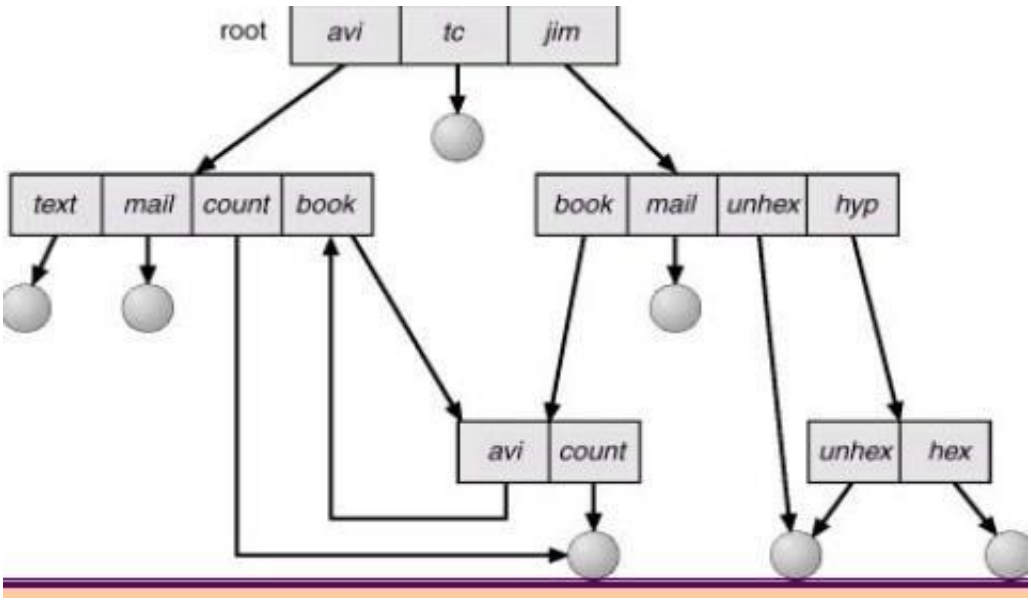
- The deletion of a link does not need to affect the original file; only the link is removed.
- Another approach to deletion is to preserve the file until all references to it are deleted. keep a count of the number of references.

  When count=0, file is deleted.
- Maintain a file reference list containing one entry for each reference to the file. On deletion just delete the entry from file reference list. File is deleted when this list becomes empty.

With a shared file only one actual file exists, so any changes made by one person are immediately visible to the other.

5. General Graph Directory

- When links are added to an existing tree-structured directory, a general graph structure can be created.



- A general graph can have cycles and cycles cause problems when searching or traversing file system.
- How do we guarantee no cycles?
  - Allow only links to files not subdirectories.
  - Use Garbage collection. {computationally expensive}
  - Every time a new link is added, use a cycle detection algorithm to determine whether a cycle now exists.

Disadvantage: Computationally expensive

An alternative approach is to bypass links during directory traversal.

4.9 DIRECTORY IMPLEMENTATION

1. Linear List

- The simplest method of implementing a directory is to use a linear list of file names with pointer to the data blocks.
- A linear list of directory entries requires a linear search to find a particular entry.
- This method is simple to program but time- consuming to execute. To create a new file, we must first search the but time – consuming to execute.
- The real disadvantage of a linear list of directory entries is the linear search to find a file.

2. Hash Table

- In this method, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- Therefore, it can greatly decrease the directory search time.
- Insertion and deleting are also fairly straight forward, although some provision must be made for collisions – situation where two file names hash to the same location.
- The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.

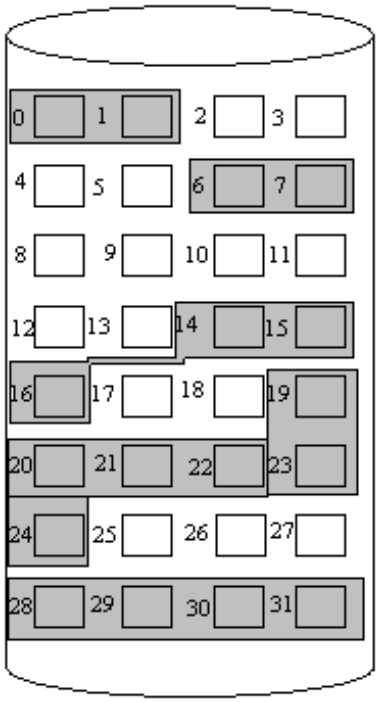
4.10 ALLOCATION METHODS

Allocation methods deals with how to allocate space to the files so that disk space is utilized effectively and files can be accessed quickly.
There are there major methods of allocating disk space:

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

1. Contiguous Allocation

The contiguous – allocation method requires each file to occupy a set of contiguous blocks on the disk.



Directory

file	start	length
Count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2,….,b+n-1.
- The directory entry for each file indicates the address of the starting block and the length allocated for this file.

Disadvantages:

1. Finding space for a new file.

It suffers from the problem of external fragmentation. Storage is fragmented into a number of holes, no one of which is large enough to store the needed data.

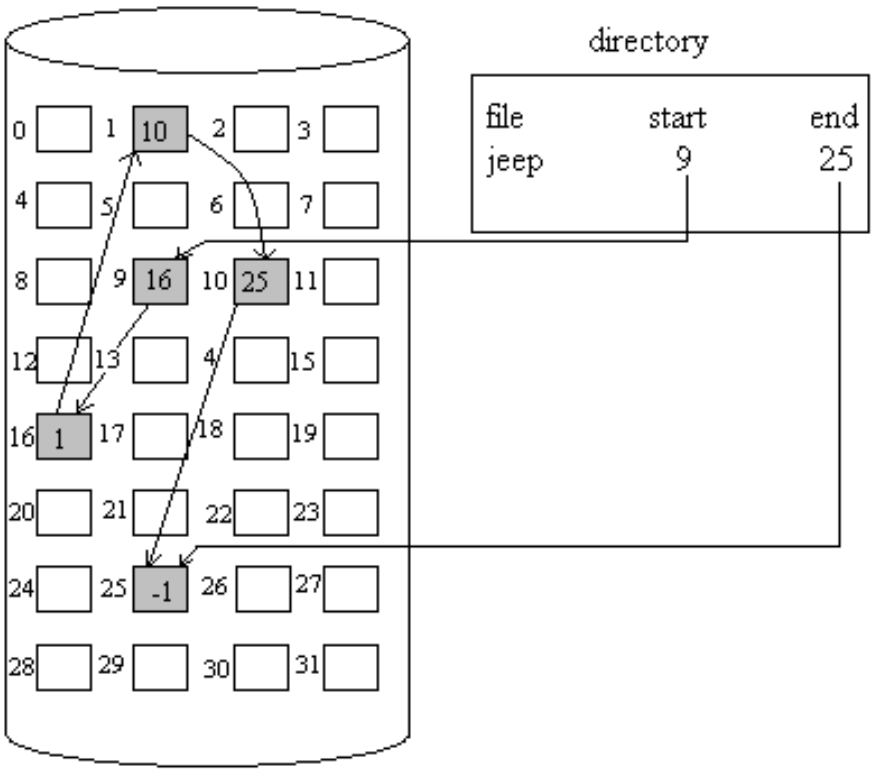
2. Determining how much space is needed for a file.

- When the file is created, the total amount of space it will need must be specified. It is not possible for all files.
- Even if the total amount of space needed for a file is known in advance pre-allocation may be inefficient.
- A file that grows slowly over a long period (months or years) must be allocated enough space for its final size therefore has an internal fragmentation.

To overcome these disadvantages: Use a modified contiguous allocation scheme, in which a contiguous chunk of space called as an **extent** is allocated initially and then, when that amount
is not large enough another chunk of contiguous space an extent is added to the initial allocation.

2. Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally bock 25.
- Each block contains a pointer to the next block. These pointers are not made available to the user.
- There is no external fragmentation with linked allocation, and any free block on the free space list can be used to satisfy a request.
- The size of a file does not need to be declared when that file is created.



Disadvantages:

1. Used effectively only for sequential access files.

To find the ith block of a file, we must start at the beginning of that file, and follow the pointers until we get to the ith block. It is inefficient to support a direct- access capability for linked allocation files.

2. Space required for the pointers

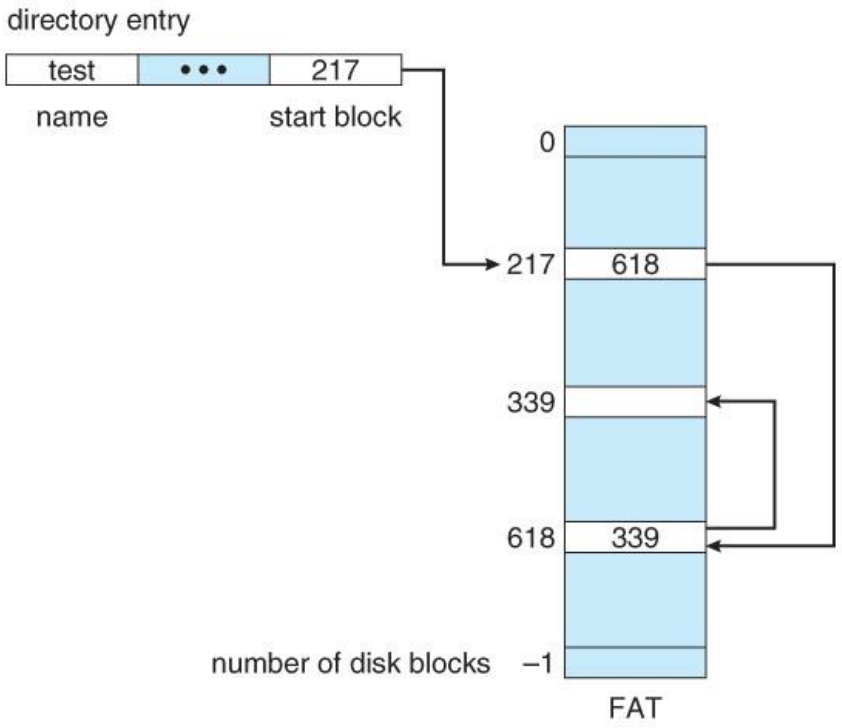
If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information. Solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate the clusters rather than blocks.

3. Reliability

Since the files are linked together by pointers scattered all over the disk hardware failure might result in picking up the wrong pointer. This error could result in linking into the free- space list or into another file.

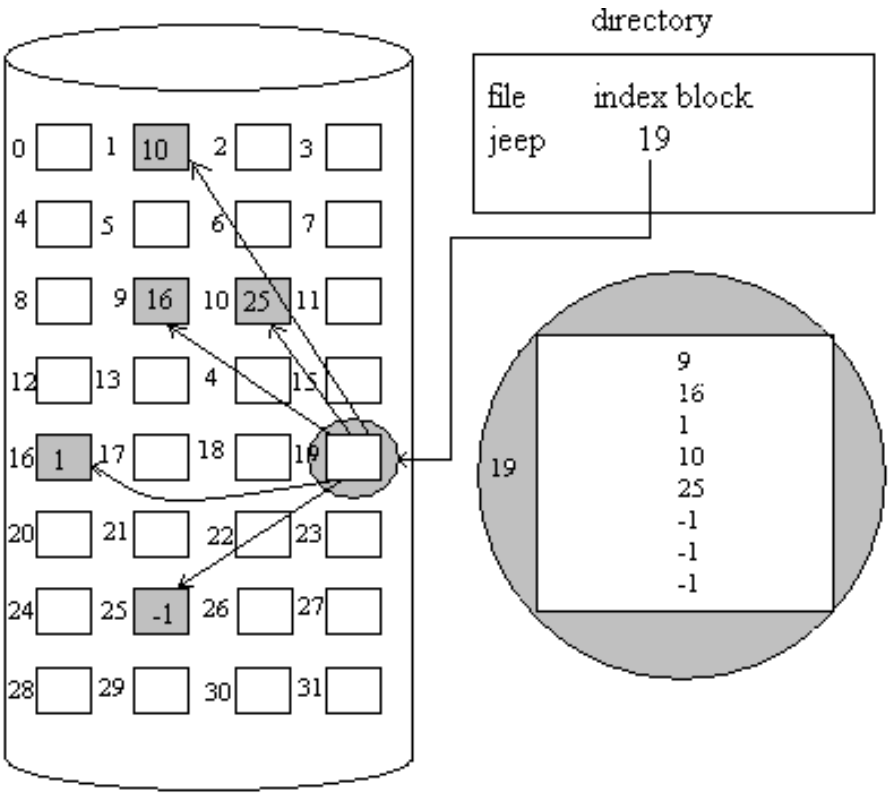
File Allocation Table(FAT)

- An important variation on the linked allocation method is the use of a file allocation table(FAT).
- This simple but efficient method of disk- space allocation is used by the MS-DOS and OS/2 operating systems.
- A section of disk at beginning of each partition is set aside to contain the table.
- The table has entry for each disk block, and is indexed by block number.
- The FAT is much as is a linked list.
- The directory entry contains the starting block number of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until the last block which has a special end – of – file value as the table entry.
- Unused blocks are indicated by a 0 table value.
- Allocating a new block file is a simple matter of finding the first 0 – valued table entry, and replacing the previous end of file value with the address of the new block.
- Example the FAT structure for a file consisting of disk blocks 217,618, and 339.



3. Indexed Allocation

- Linked allocation solves the external – fragmentation and size- declaration problems of contiguous allocation.
- Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered.
- Indexed allocation solves this problem by bringing all the pointers together into one location: the **index block**.
- Each file has its own index block, which is an array of disk – block addresses.
- The ith entry in the index block points to the ith block of the file.
- The directory contains the address of the index block .
- To read the ith block, we use the pointer in the ith index – block entry to find and read the desired block this scheme is similar to the paging scheme.



Disadvantages

1.Pointer Overhead

Indexed allocation suffers from wasted space. The pointer over head of the index block is generally greater than the pointer over head of linked allocation.

2. Size of Index block

If the index block is too small, however, it will not be able to hold enough pointers for a large file.

Linked Scheme: An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we may link together several index blocks.

Multilevel index: A variant of the linked representation is to use a first level index block to point to a set of second – level index blocks.

Combined scheme:

- Another alternative, used in the UFS, is to keep the first, say, 15 pointers of the index block in the file's inode.
- The first 12 of these pointers point to direct blocks; that is for small (no more than 12 blocks) files do not need a separate index block
- The next pointer is the address of a single indirect block.

- The single indirect block is an index block, containing not data, but rather the addresses of blocks that do contain data.
- Then there is a double indirect block pointer, which contains the address of a block that contains pointers to the actual data blocks. The last pointer would contain pointers to the actual data blocks.
- The last pointer would contain the address of a triple indirect block.

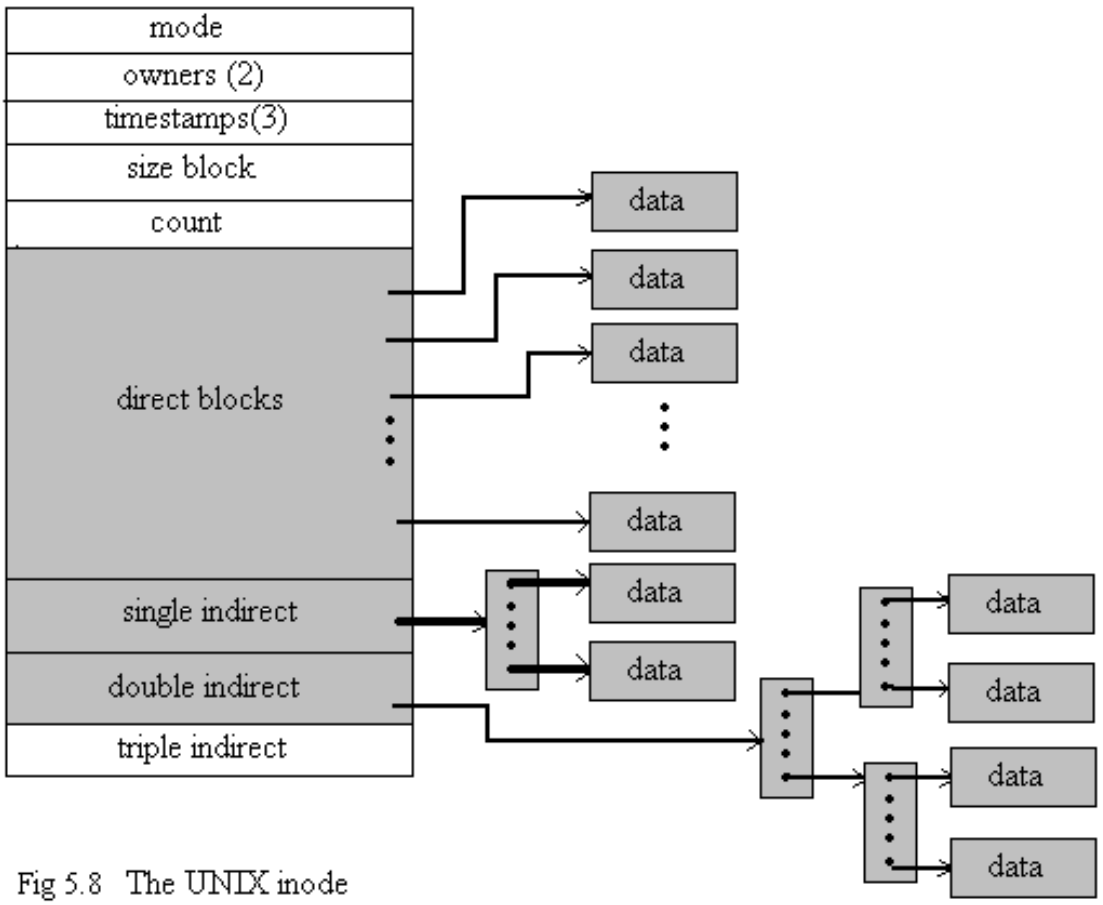


Fig 5.8 The UNIX inode

1. Disk initialization
2. Booting from disk
3. Bad-block recovery.

Disk Formatting:

Low-level formatting or physical formatting: Before a disk can store data, the sector is divided into various partitions. This process is called low-level formatting or physical formatting. It fills the disk with a special data structure for each sector. Data structure includes header, data area, and trailer.

The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC).**

This formatting enables the manufacturer to

1. Test the disk and
2. To initialize the mapping from logical block numbers

Logical Formatting

- After partitioning, then the file systems must be **logically formatted,** which involves laying down the master directory information, initializing free lists, and creating at least the root directory of the file system.

Boot Block

- The bootstrap is stored in read-only memory **(ROM).**
- The full bootstrap program is stored in a partition called the **boot blocks**, at a fixed location on the disk.
- A disk that has a boot partition is called a **boot disk or system disk**.
- **Bootstrap loader**: load the entire operating system from a non-fixed location on disk, and to start the operating system running.

Example:

Windows

- **Boot partition**—contains the operating system and device drivers.

The Windows system places its boot code in the first sector on the hard disk,

4.3 DISK MANAGEMENT:

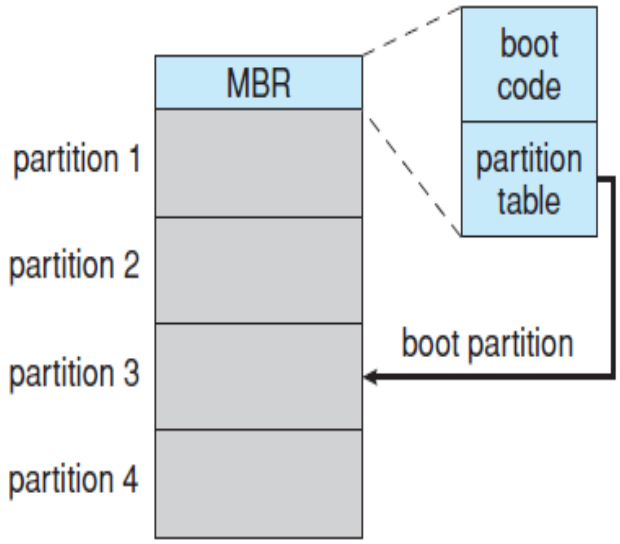
The operating system is responsible for several other aspects of disk management.

- which it terms the **master boot record**, or **MBR**.

Booting begins by running code that is resident in the system's ROM memory. This code directs the system to read the boot code from the MBR.

In addition to containing boot code, the MBR contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from.

Once the system identifies the boot partition, it reads the first sector from that partition (which is called the **boot sector**) and continues with the remainder of the boot process, which includes loading the various subsystems and system services.



Bad Blocks:

- The disk with defected sector is called as bad block.
- Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: "Handled manually"

If blocks go bad during normal operation, a **special program** must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

Method 2: "sector sparing or forwarding"

The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

Method 3: "sector slipping"

For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

4.2 DISK SCHEDULING

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives it means,

1. A fast access time and
2. High disk bandwidth.

The **access time** has two major components;

- The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- The **rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head.
- The disk **band width** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

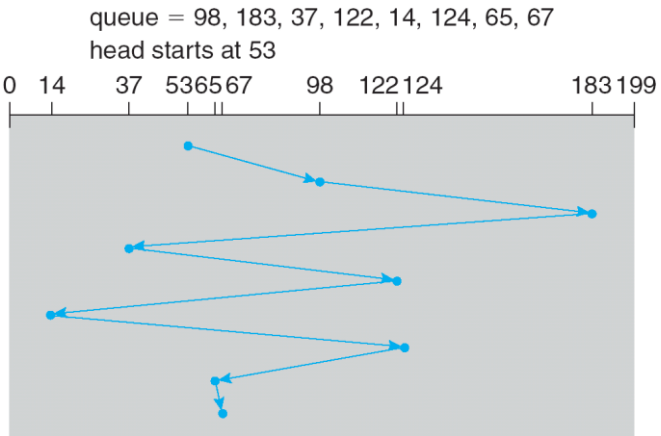
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

1. FCFS Scheduling:

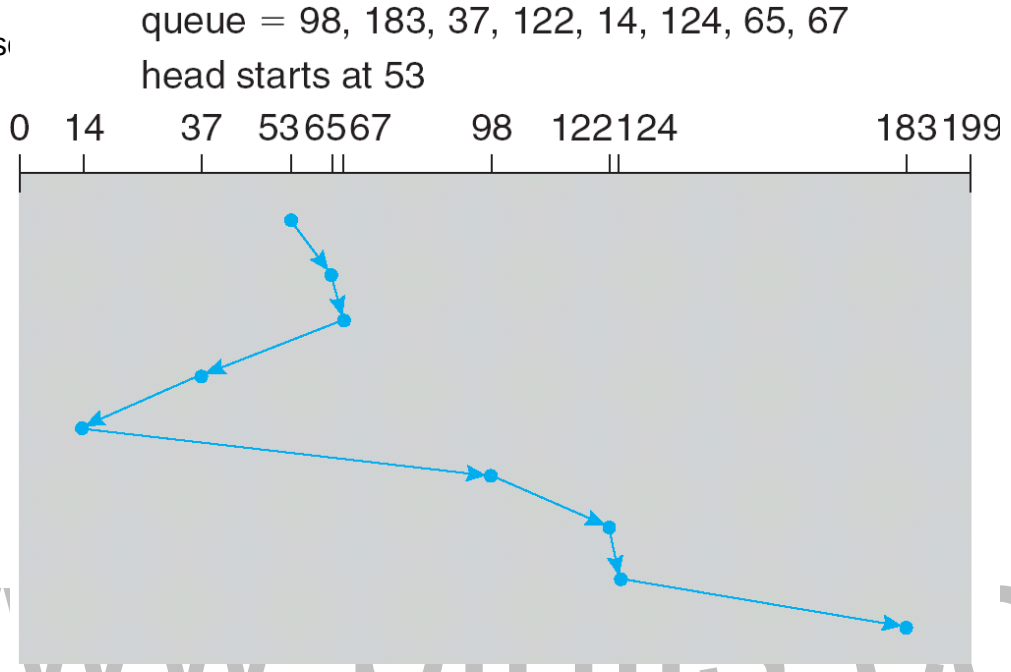
The simplest & fastest form of disk scheduling. I/O requests are serviced based on their arrival.



Total head movement = 640 cylinders

2. SSTF (shortest-seek-time-first)Scheduling

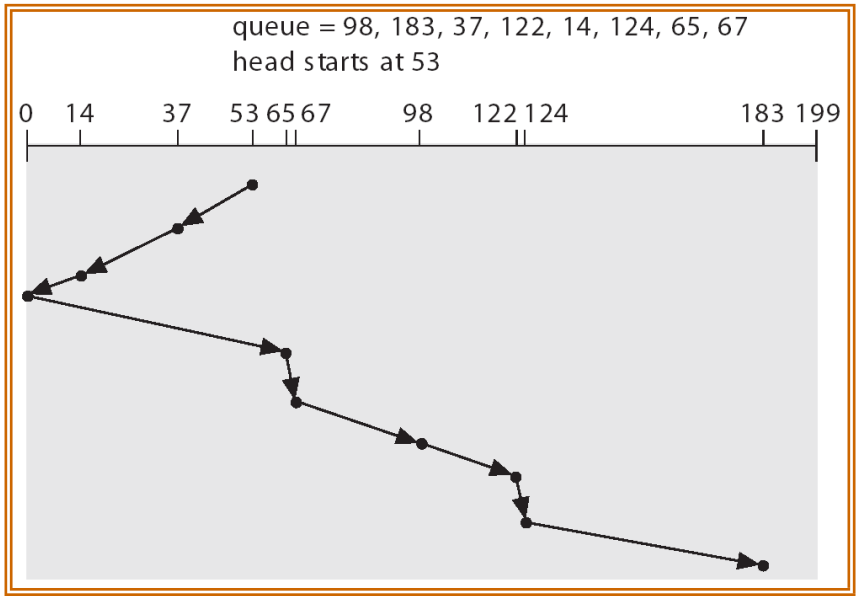
Selects the request with the minimum seek time from the current head position. SSTF scheduling is a form of SJF scheduling; may cause starvation of s



Total head movement = 236 cylinders

3. SCAN Scheduling

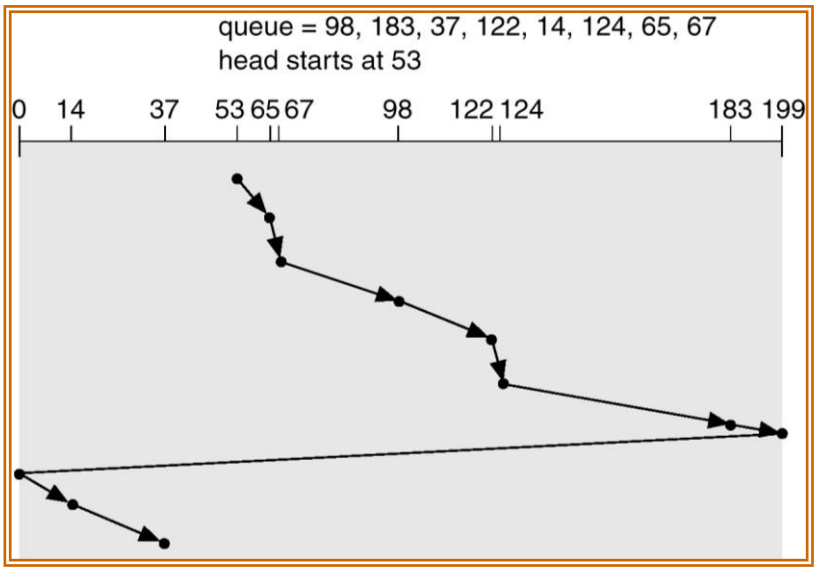
- The disk head starts at one end of the disk, and moves toward the other end, servicing requests, until it gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- **Elevator algorithm**: Sometimes the SCAN algorithm is called as the elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up, and then reversing to service requests the other way.



Total head movement of 208 cylinders

4. C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

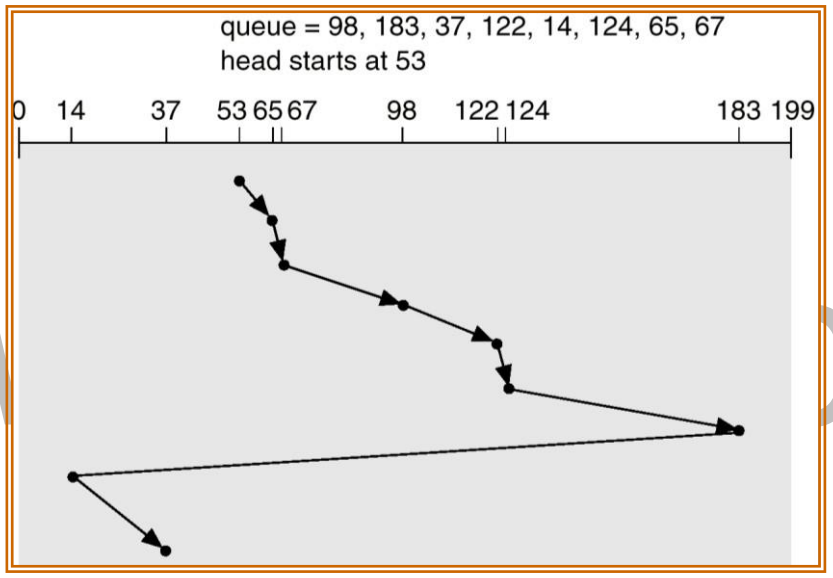


5. LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In this, the arm only moves up to the final request in each direction. Then, it reverses direction immediately, without reaching the end of the disk.

C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



Multiple Users

- On a multi-user system, more information needs to be stored for each file:
  - The owner (user) who owns the file, and who can control its access.
  - The group of other user IDs that may have some special access to the file.
  - What access rights are given to the owner (User), the Group, and to the rest of the world.
- Some systems have more complicated access control, allowing or denying specific accesses to specifically named users or groups.

Remote File Systems

- The advent of the Internet introduces issues for accessing files stored on remote computers
- The original method was ftp, allowing individual files to be transported across systems as needed. Ftp can be either account and password controlled, or **anonymous**, not requiring any user name or password.
- Various forms of **distributed file systems** allow remote file systems to be mounted onto a local directory structure, and accessed using normal file access commands.
- The WWW has made it easy once again to access files on remote systems without mounting their file systems, generally using (anonymous) ftp as the underlying file transport mechanism.

The Client-Server Model

- When one computer system remotely mounts a filesystem that is physically located on another system, the system which physically owns the files acts as a **server**, and the system which mounts them is the **client.**
- User IDs and group IDs must be consistent across both systems for the system to work properly. (I.e. this is most applicable across multiple computers managed by the same organization, shared by a common group of users.)

- **4.6 FILE SHARING**
- The same computer can be both a client and a server. (E.g. crosslinked file systems.)
- There are a number of security concerns involved in this model:
- Servers commonly restrict mount permission to certain trusted systems only. Spoofing (a computer pretending to be a different computer) is a potential security risk.

- Servers restrict which filesystems may be remotely mounted. Generally the information within those subsystems is limited, relatively public, and protected by frequent backups.
- The NFS (Network File System) is a classic example of such a system.

Distributed Information Systems

- The **Domain Name System, DNS,** provides for a unique naming system across all of the Internet.
- Domain names are maintained by the **Network Information System, NIS**, which unfortunately has several security issues. NIS+ is a more secure version, but has not yet gained the same widespread acceptance as NIS.
- Microsoft's **Common Internet File System, CIFS**, establishes a **network login** for each user on a networked system with shared file access. Older Windows systems used **domains**, and newer systems (XP, 2000), use **active directories.** User names must match across the network for this system to be valid.
- A newer approach is the **Lightweight Directory-Access Protocol, LDAP,** which provides a **secure single sign-on** for all users to access all resources on a network. This is a secure system which is gaining in popularity, and which has the maintenance advantage of combining authorization information in one central location.

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 response to fail.

However when a remote file is unavailable, there are many possible reasons, and whether or not it is unrecoverable is not readily apparent. Hence most remote access systems allow for blocking or delayed response, in the hopes that the remote system (or the network) will come back up eventually. Servers may restrict remote access to read- only.

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 otherusers see the changes?

UNIX Semantics

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

One implementation uses a shared location pointer, which is adjusted for all sharing users.

Session Semantics

- The Andrew File System, AFS uses the following semantics:
- Writes to an open file are not immediately visible to other users.
- When a file is closed, any changes made become available only to users who open the file at a later time.

Immutable-Shared-Files Semantics

Under this system, when a file is declared as **shared** by its creator, it becomes immutable and the name cannot be re-used for any other resource. Hence it becomes read-only, and shared access is simple.

4.7 PROTECTION

- Files must be kept safe for reliability (against accidental damage), and protection (against
- Deliberate malicious access.) The former is usually managed with backup copies.
- One simple protection scheme is to remove all access to a file.
- However this makes the file unusable, so some sort of controlled access must be arranged.

Types of Access

The following low-level operations are often controlled:

- Read - View the contents of the file
- Write - Change the contents of the file.
- Execute - Load the file onto the CPU and follow the instructions contained therein.
- Append - Add to the end of an existing file.
- Delete - Remove a file from the system.
- List -View the name and other attributes of files on the system.
- Higher-level operations, such as copy, can generally be performed through combinations of the above.

- **Access Control**
- One approach is to have complicated **Access Control Lists, ACL,** which specify exactly what access is allowed or denied for specific users or groups.
- The AFS uses this system for distributed access.
- Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown. (AFS allows some wild cards, so for example all users on a certain remote system may be trusted, or a given username may be trusted when accessing from any remote system.)
- UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others. The RWX bits control the following privileges for ordinary files and directories:

bit	Files	Directories
R	Read (view) file contents.	Read directory contents. Required to get a listing of the directory.
W	Write (change) file contents.	Change directory contents. Required to create or delete files.
X	Execute file contents as a program.	Access detailed directory information. Required to get a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access.

- In addition there are some special bits that can also be applied:
- The set user ID (SUID) bit and/or the set group ID (SGID) bits applied to executable files temporarily change the identity of whoever runs the program to match that of the owner / group of the executable program. Setting of these two bits is usually restricted to root, and must be done with caution, as it introduces a potential security leak.
- The sticky bit on a directory modifies write permission, allowing users to only delete files for which they are the owner. This allows everyone to create files in /tmp, for example, but to only delete files which they have created, and not anyone else's.
- The SUID, SGID, and sticky bits are indicated with an S, S, and T in the positions for execute permission for the user, group, and others, respectively. If the letter is lower

case, (s, s, t), then the corresponding execute permission is not also given. If it is upper case, (S, S, T), then the corresponding execute permission IS given.

Other Protection Approaches and Issues

- Some systems can apply passwords, either to individual files, or to specific subdirectories,
- or to the entire system. There is a trade-off between the number of passwords that must be maintained and the amount of information that is vulnerable to a lost or forgotten password.
- Older systems which did not originally have multi-user file access permissions must now be **retrofitted** if they are to share files on a network.
- Access to a file requires access to all the files along its path as well. In a cyclic directory structure, users may have different access to the same file accessed through different paths.
- Sometimes just the knowledge of the existence of a file of a certain name is a security (or privacy) concern. Hence the distinction between the R and X bits on UNIX directories.

4.8 FILE SYSTEM IMPLEMENTATION

FILE-SYSTEM STRUCTURE

Hard disks have two important properties that make them suitable for secondary storage of files in file systems:

(1) Blocks of data can be rewritten in place, and

(2) They are direct access, allowing any block of data to be accessed with only (relatively) minor movements of the disk heads and rotational latency.

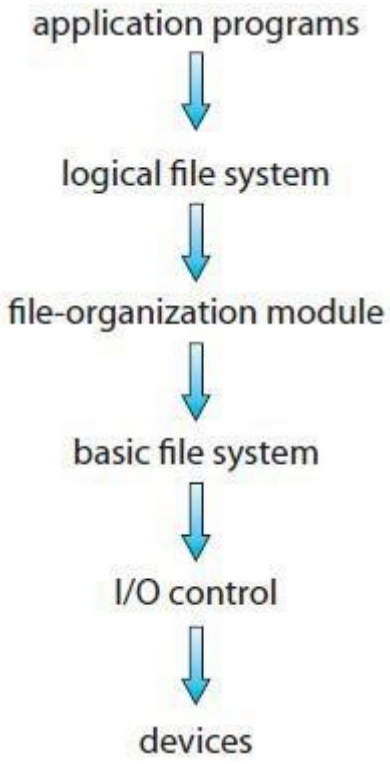
- Disks are usually accessed in physical blocks, rather than a byte at a time. Block sizes may range from 512 bytes to 4K or larger.
- File systems organize storage on disk drives, and can be viewed as a layered

design:

- At the lowest layer are the physical devices, consisting of the magnetic media, motors & controls, and the electronics connected to them and controlling them. Modern disk put more and more of the electronic controls directly on the disk driveitself, leaving relatively little work for the disk controller card to perform.

I/O Control consists of **device drivers**, special software programs (often written in assembly) which communicate with the devices by reading and writing special codes directly to and from memory addresses corresponding to the controller card's registers. Each controller card (device) on a system has a different set of addresses (registers, a.k.a. **ports**) that it listens to, and a unique set of command codes and results codes that it understands.

- The **basic file system** level works directly with the device drivers in terms of retrieving and storing raw blocks of data, without any consideration for what is in each block. Depending on the system, blocks may be referred to with a single block number, (e.g. block # 234234), or with head-sector-cylinder combinations.
- The **file organization module** knows about files and their logical blocks, and how they map to physical blocks on the disk. In addition to translating from logical to physical 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 fora wide variety of different file systems, and only certain layers need to be file system specific. 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 file systems ext2 and ext3 (among 40 others supported.)



4.4 FILE SYSTEM STORAGE

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- Data cannot be written to secondary storage unless they are within a

file. Examples of files:

- A text file is a sequence of characters organized into lines (and possibly pages).
- A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
- An object file is a sequence of bytes organized into blocks understandable by the system's linker.
- An executable file is a series of code sections that the loader can bring into memory and execute.

File Attributes

- **Name**: The symbolic file name is the only information kept in human readable form.
- **Identifier**: This unique tag, usually a number identifies the file within the file system. It is the non-human readable name for the file.
- **Type**: This information is needed for those systems that support different types.
- **Location**: This information is a pointer to a device and to the location of the file on that device.
- **Size**: The current size of the file (in bytes, words or blocks)and possibly the maximum allowed size are included in this attribute.
- **Protection**: Access-control information determines who can do reading, writing, executing and so on.
- **Time, date and user identification**: This information may be kept for creation, last modification and last use. These data can be useful for protection, security and usage monitoring.

File Operations

The file ADT supports many common operations:

1. **Creating a file**

Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

information to be written to the file. Given the name of the file, the system searches the directory to find the file's location.

The system must keep a **write pointer**. The write pointer must be updated whenever a write

occurs.

3. Reading a file

To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a **read pointer.** Once the read has taken place, the read pointer is updated.

4. Repositioning within a file

The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. This file operation is also known as a file **seek**.

5. Deleting a file Removing a file

6. 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 function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

Most OS requires that files be **opened** before access and **closed** after all access is complete. Normally the programmer must open and close files explicitly, but some rare systems open the file automatically at first access. Information about currently open files is stored in an **open file table**, containing for example:

- **File pointer** - records the current position in the file, for the next read or write access.
- **File-open count** - How many times has the current file been opened (simultaneously by different processes) and not yet closed? When this counter reaches zero the file can be removed from the table.
- **Disk location of the file.-**The information needed to locate the file on disk is kept in

7. 2. **Writing a file**

To write a file, we make a system call specifying both the name of the file and the memory so that the system does not have to read it from disk for each operation.

Access rights- Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny

subsequent I/O requests.

Some systems provide support for **file locking.**

- A **shared lock** is for reading only.
- A**exclusive lock** is for writing as well as reading.
- An **advisory lock** is informational only, and not enforced. (A "Keep Out" sign, which may be ignored.)
- A **mandatory lock** is enforced. (A truly locked door.) UNIX used advisory locks, and Windows uses mandatory locks.

File Types

• Windows (and some other systems) use special file extensions to indicate the type of each file:

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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

File Structure

- File types also can be used to indicate the internal structure of the file.
- Some files contain an internal structure, which may or may not be known to the OS.
- For the OS to support particular file formats increases the size and complexity of the OS.
- 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 resource fork contains information relating to the UI, such as icons and button images, and can be modified independently of the data fork, which

  contains the code or data as appropriate.

Internal File Structure

- Disk files are accessed in units of physical blocks, typically 512 bytes or some power-of-two multiple thereof. (Larger physical disks use larger block sizes, to keep the range of block numbers within the range of a 32-bit integer.)
- Internally files are organized in units of logical units, which may be as small as a single byte, or may be a larger size corresponding to some data record or structure size.
- The number of logical units which fit into one physical block determines its **packing**, and has an impact on the amount of internal fragmentation (wasted space) that occurs.
- As a general rule, half a physical block is wasted for each file, and the larger the block sizes the more space is lost to internal fragmentation.

4.11 FREE-SPACE MANAGEMENT

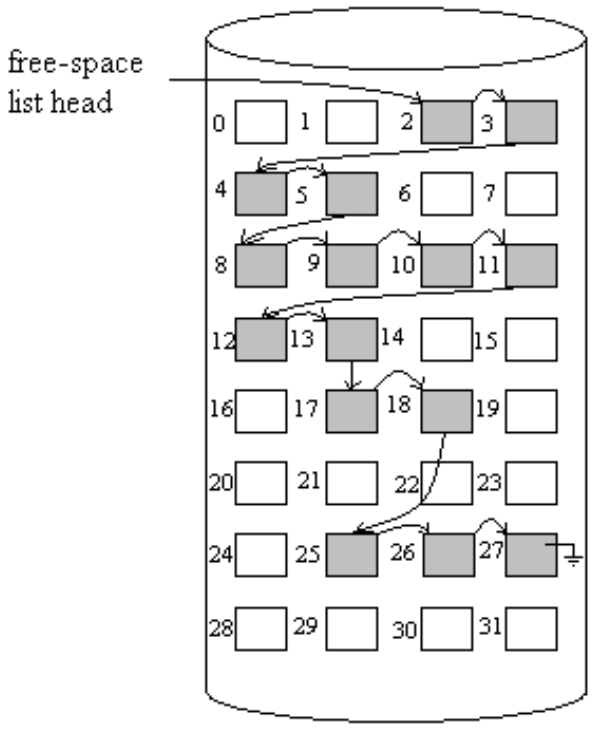
- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks – those not allocated to some file or directory.
- To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file.
- This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list.

1. Bit Vector

- The free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block 2,3,4,5,8 , and the rest of the block are allocated. The free space bit map would be 00111100100000 …
- The main **advantage** of this approach is it's simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

2. Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.
- In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.



3. Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- The first n-1 of these blocks are actually free.
- The last block contains the addresses of another n free blocks, and so on.
- The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

4. Counting

- We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.
- Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

5. Space Maps

- Oracle's **ZFS** file system (found in Solaris and other operating systems) was designed to encompass huge numbers of files, directories, and even file systems.

- In its management of free space, ZFS uses a combination of techniques to control the size of data structures and minimize the I/O needed to manage those structures.
- First, ZFS creates **meta slabs** to divide the space on the device into chunks of manageable size. A given volume may contain hundreds of meta slabs. Each meta slab has an associated space map.
- ZFS uses the counting algorithm to store information about free blocks. Rather than write counting structures to disk, it uses log-structured file-system techniques to record them.
- The space map is a log of all block activity (allocating and freeing), in time order, in counting format.
- When ZFS decides to allocate or free space from a meta slab, it loads the associated space map into memory in a balanced-tree structure (for very efficient operation), indexed by offset, and replays the log into that structure.
- The in-memory space map is then an accurate representation of the allocated and free space in the meta slab. ZFS also condenses the map as much as possible by combining contiguous free blocks into a single entry.
- Finally, the free-space list is updated on disk as part of the transaction-oriented operations of ZFS.

UNIT IV

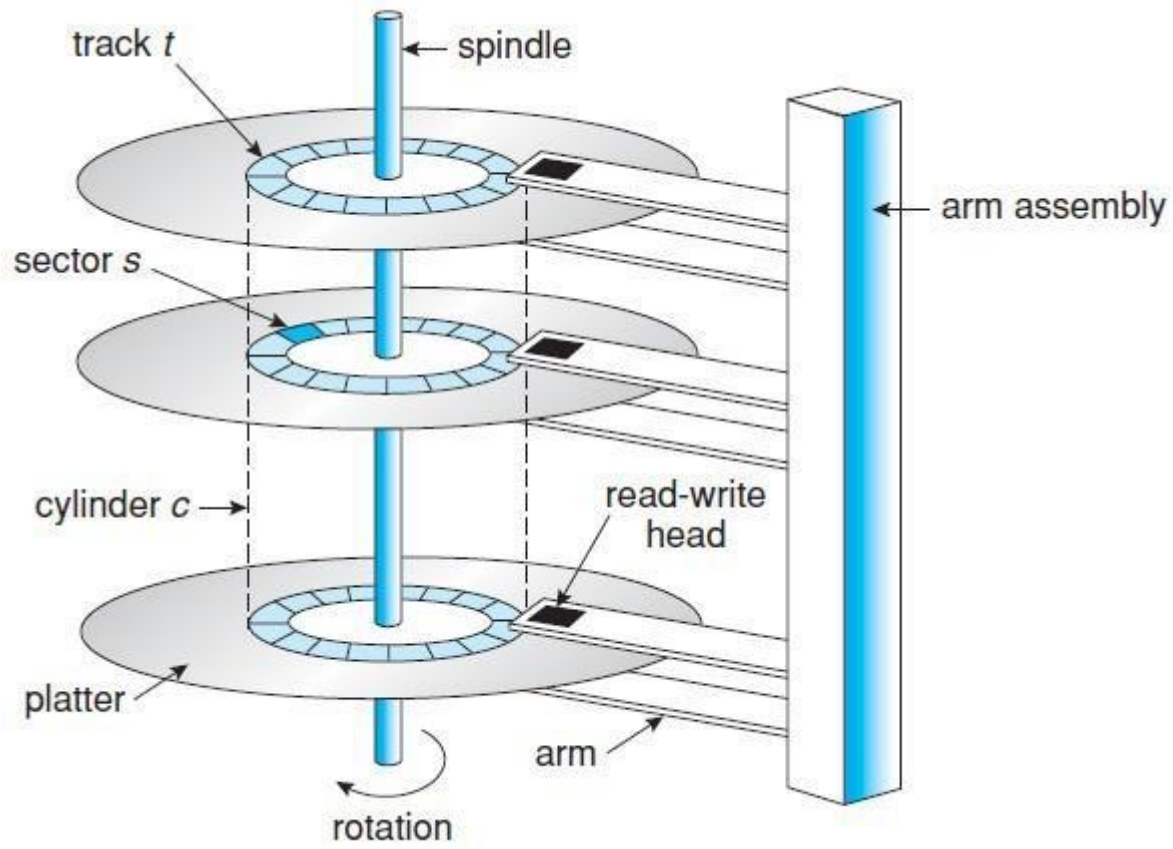
4.1 MASS STORAGE STRUCTURE

Overview of Mass-Storage Structure

Magnetic Disks :

Traditional magnetic disks have the following basic structure:

- One or more platters in the form of disks covered with magnetic media.
- Hard disk platters are made of rigid metal
- Each platter has two working surfaces.
- Each working surface is divided into a number of concentric rings called tracks.
- The collection of all tracks that are the same distance from the edge of the platter, is called a cylinder.
- Each track is further divided into sectors, traditionally containing 512 bytes of data each, Sectors also include a header and a trailer, including checksum information among other things.
- The data on a hard drive is read by read-write heads.
- The standard configuration uses one head per surface, each on a separate arm, and controlled by a common arm assembly which moves all heads simultaneously from one cylinder to another.
- The storage capacity of a traditional disk drive is equal to the number of heads, times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector.
- The **positioning time**, or **seek time** or **random access time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- The **rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head
- The transfer rate, which is the time required to move the data electronically from the disk to the computer.



ig: Moving-head disk mechanism

- Disk drives are connected to the computer via a cable known as the **I/O Bus.**
- Some of the common interface formats include Enhanced Integrated Drive Electronics, EIDE; Advanced Technology Attachment, ATA; Serial ATA, SATA, Universal Serial Bus, USB; Fiber Channel, FC, and Small Computer Systems Interface, SCSI.
- The **host controller** is at the computer end of the I/O bus, and the **disk controller** is built into the disk itself.
- The CPU issues commands to the host controller via I/O ports.

Solid-State Disks - New

- SSDs use memory technology as a small fast hard disk. Specific implementations may use either flash memory or DRAM chips protected by a battery to sustain the information through power cycles.
- Because SSDs have no moving parts they are much faster than traditional hard drives, and certain problems such as the scheduling of disk accesses simply do not apply.

- They are more expensive than hard drives, generally not as large, and may have shorter life spans.
- SSDs are especially useful as a high-speed cache. Another variation is a boot disk containing the OS and some application executable, but no vital user data.
- SSDs are also used in laptops to make them smaller, faster, and lighter.
- Because SSDs are so much faster than traditional hard disks, the throughput of the bus can become a limiting factor, causing some SSDs to be connected directly to the system PCI bus for example.

Magnetic Tapes

- Magnetic tapes were once used for common secondary storage before the days of hard disk drives, but today are used primarily for backups.
- Accessing a particular spot on a magnetic tape can be slow, but once reading or writing commences, access speeds are comparable to disk drives.
- Capacities of tape drives can range from 20 to 200 GB, and compression can double that capacity.

STREAM :

It is a full-duplex communication channel between a user-level process and a device in Unix System V and beyond

A STREAM consists of:

a) STREAM head interfaces with the user process

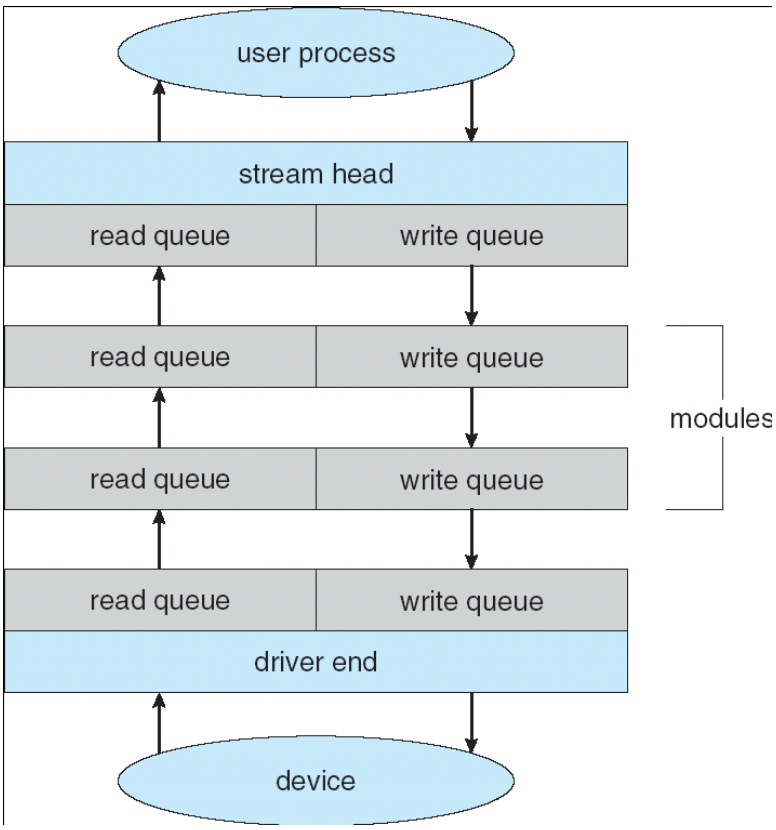
b) Driver end interfaces with the device

c) Zero or more STREAM modules between them.

Steps:

- The user process interacts with the ***stream head.***
- The device driver interacts with the ***device end.***
- Each module contains a read queue and a write queue
- Message passing is used to communicate between queues
- Modules provide the functionality of STREAMS processing and they are pushed onto a stream using the ioctl () system call.
- User processes communicate with the stream head using either read( ) and write( ) (or putmsg( ) and getmsg( ) for message passing.

The device driver **must** respond to interrupts from its device - If the adjacent module is not prepared to accept data and the device driver's buffers are all full, then data is typically dropped. Streams are widely used in UNIX, and are the preferred approach for device drivers. For example, UNIX implements sockets using streams.



Performance:

- I/O is a major factor in system performance:

We can employ several principles to improve the efficiency of I/O:

1. Reduce the number of context switches.

2. Reduce the number of times that data must be copied in memory while passing between device and application.

3. Reduce the frequency of interrupts .

4. Increase concurrency by using DMA-knowledgeable controllers .

5. Balance CPU, memory subsystem, bus, and I/O performance, because an overload in any one area will cause idleness in others.

Kernel I/O Subsystem

Kernels provide many services related to I/O.

- One way that the I/O subsystem improves the efficiency of the computer is by scheduling I/O operations.

- Another way is by using storage space in main memory or on disk, via techniques called buffering, caching, and spooling.

Services include:

1. I/O Scheduling:

To determine a good order in which to execute the set of I/O requests. Uses:

a) It can improve overall system performance,

b) It can share device access fairly among processes, and

c) It can reduce the average waiting time for 1/0 to complete.

Implementation: OS developers implement scheduling by maintaining a —queue of requestsll for each device.

1. When an application issues a blocking I/O system call,
2. The request is placed on the queue for that device.
3. The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

2. Buffering:

Buffer: A memory area that stores data while they are transferred between two devices or between a device and an application.

Reasons for buffering:

a) To cope with a speed mismatch between the producer and consumer of a data stream.

b) To adapt between devices that have different data-transfer sizes.

c) To support copy semantics for application I/O.

Copy semantics: Suppose that an application has a buffer of data that it wishes to write to disk. It calls the write () system call.After the system call returns, what happens if the application changes the contents of the buffer?

With copy semantics, the version of the data written to disk is guaranteed to be the version at the time of the application system call, independent of any subsequent changes in the application's buffer.

application. The disk write is performed from the kernel buffer, so that subsequent changes to the application buffer have no effect.

3. Caching

A cache is a region of fast memory that holds copies of data.

Access to the cached copy is more efficient than access to the original

Cache vs buffer: A buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere.

When the kernel receives a file I/O request,

1. The kernel first accesses the buffer cache to see whether that region of the file is already available in main memory.
2. If so, a physical disk I/O can be avoided or deferred. Also, disk writes are accumulated in the buffer cache for several seconds, so that large transfers are gathered to allow efficient write schedules.

4. Spooling and Device Reservation:

Spool: A buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.

A printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together

The os provides a control interface that enables users and system administrators ;

a) To display the queue,

b) To remove unwanted jobs before those jobs print,

c) To suspend printing while the printer is serviced, and

so on. Device reservation - provides exclusive access to a device

- System calls for allocation and de-allocation

- A simple way that the operating system can guarantee copy semantics is for the write()
- system call to copy the application data into a kernel buffer before returning control to the Watch out for deadlock

5. Error Handling

An operating system that uses protected memory can guard against many hardware kinds of and application errors.

- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

4.12 I/O SYSTEMS

I/O Hardware

- The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices.
- A device communicates with a computer system by sending signals over a cable or even through the air.

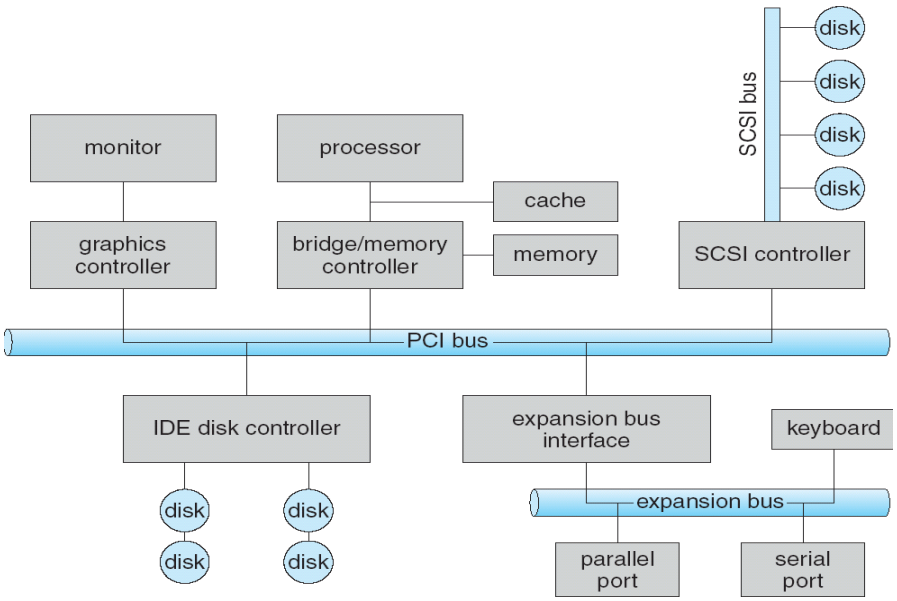
Port: The device communicates with the machine via a connection point (or port), for example, a serial port.

Bus: If one or more devices use a common set of wires, the connection is called a bus.

Daisy chain: Device 'A' has a cable that plugs into device 'B', and device 'B' has a cable that plugs into device 'C', and device 'C' plugs into a port on the computer, this arrangement is called a daisy chain. A daisy chain usually operates as a bus. Only one device is attached to computer directly.

PC bus structure:

- PCI bus- that connects the processor-memory subsystem to the fast devices
- Expansion bus that connects relatively slow devices such as the keyboard and serial and parallel ports
- A **controller or host adapter** is a collection of electronics that can operate a port, a bus, or a device.



- A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port.
- SCSI bus controller - contains a processor, microcode, and some private memory. How can the processor give commands and data to a controller to perform an I/O transfer?

- Direct I/O instructions -Use special I/O instructions that specify the transfer of a byte or word to an I/O port address. It triggers bus lines to select the proper device and to move bits into or out of a device register
- Memory-mapped I/O -The device-control registers are mapped into the address space

  of the processor. The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers.

An I/O port typically consists of four registers: status, control, data-in, and data-out registers.

- Status register- Read by the host to indicate states .
- Control register - Written by the host to start a command.
- data-in register - Read by the host to get input
- data-out register - Written by the host to send output

Polling

Interaction between the host and a controller

- The controller sets the busy bit when it is busy working, and clears the busy bit when it is ready to accept the next command.
- The host sets the command ready bit when a command is available for the controller to execute.

Coordination between the host & the controller is done by handshaking as follows:

- The host repeatedly reads the busy bit until that bit becomes clear.
- The host sets the write bit in the command register and writes a byte into the data- out register.
- The host sets the command-ready bit.
- When the controller notices that the command-ready bit is set, it sets the busy bit.
- The controller reads the command register and sees the write command. It reads the data-out register to get the byte, and does the I/O to the device.

- The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.
- In step 1, the host is —**busy-waiting or polling**: It is in a loop, reading the status register over and over until the busy bit becomes clear.

Interrupts

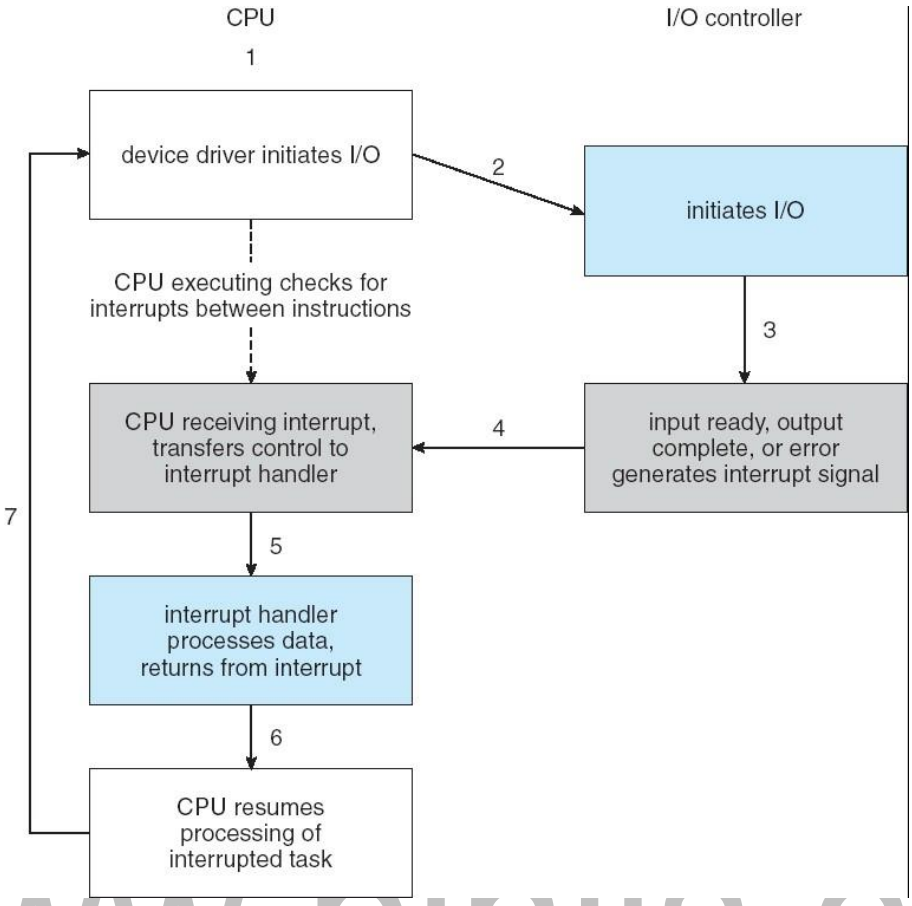
The CPU hardware has a wire called the —interrupt-request line.

The basic interrupt mechanism works as follows;

- The CPU has an ***interrupt-request line*** that is sensed after every instruction. A device's controller ***raises*** an interrupt by asserting a signal on the interrupt request line.
- The CPU then performs a state save, and transfers control to the ***interrupt handler*** routine at a fixed address in memory.
- The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a ***return from interrupt*** instruction to return control to the CPU.

Two interrupt request lines:

- **Nonmaskable interrupt**: which is reserved for events such as unrecoverable memory errors
- **Maskable interrupt**: Used by device controllers to request service. CPU can temporarily ignore this interrupts during critical processing.



Interrupt-driven I/O cycle.

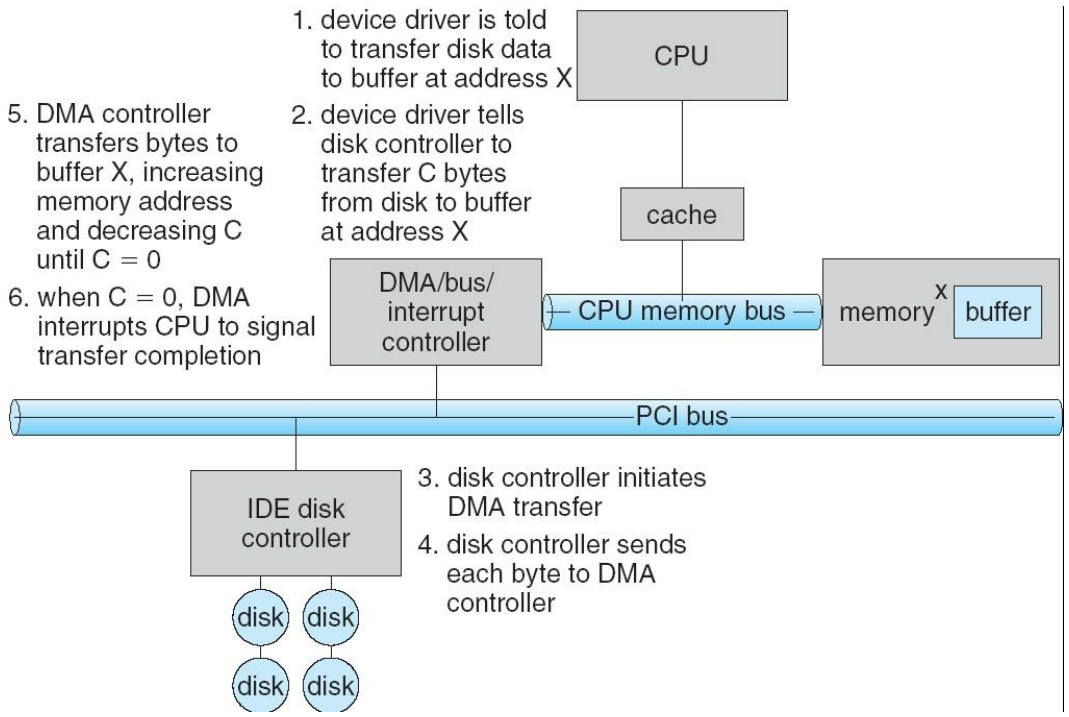
Direct Memory Access (DMA)

In general it is tough for the CPU to do the large transfers between the memory buffer & disk; because it is already equipped with some other tasks, then this will create overhead. So a special-purpose processor called a direct memory- access **(DMA)** controller is used.

Steps:

1. DMA controller transfers data without the intervention of CPU.DMA-request is initiated by device controller for getting DMA access.

2. Then DMA controller seizes bus and places memory address then send DMA-ACK. 3.Then DMA controller transfers data.



Application I/O Interface

- I/O system calls encapsulate device behaviours in generic classes
- Device-driver layer hides differences among I/O controllers from kernel Devices vary in many dimensions

1. Character-stream or block
2. Sequential or random-access
3. Sharable or dedicated
4. Speed of operation
5. read-write, read only, or write only

Types	Description	Example
Character-stream or block	A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.	Terminal, Disk
Sequential or random-access	A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.	Modem, CD-ROM
Sharable or dedicated	A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.	Tape, Keyboard
Speed of operation	Latency, seek time, transfer rate, delay between operations	
read-write, read only, or write only	Some devices perform both input and output, but others support only one data direction.	CD-ROM, Graphics controller, Disk

1. Block and Character Devices

- ***Block devices*** are accessed a block at a time. Operations supported include read( ), write( ), and seek( ).
- ***Character devices*** are accessed one byte at a time. Supported operations include get( ) and put( ).

2. Network Devices

- Because the performance and addressing characteristics of network I/O differ significantly from those of disk I/O, most operating systems provide a network I/O interface that is different from the read0 -write() -seek() interface used for disks.
  - Windows NT provides one interface to the network interface card, and a second interface to the network protocols.
  - In UNIX, we find half-duplex pipes, full-duplex FIFOs, full-duplex STREAMS, message queues and sockets.

3. Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

1. Give the current time
2. Give the elapsed time
3. Set a timer to trigger operation X at time T

These functions are used by the operating system & also by time sensitive applications.

Programmable interval timer: The hardware to measure elapsed time and to trigger operations is called a programmable interval timer.

Counter: The hardware clock is constructed from a high frequency counter. In some computers, the value of this counter can be read from a device register.

4. **Blocking and Non-blocking I/O (or) synchronous & asynchronous:**

Blocking I/O: When an application issues a blocking system call;

- The execution of the application is suspended.
- The application is moved from the operating system's run queue to a wait queue.
- After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution, at which time it will receive the values returned by the system call.

Non-blocking I/O: Some user-level processes need non-blocking I/O.

Examples:

1. User interface that receives keyboard and mouse input while processing and displaying data on the screen.

2. Video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display