Unit v os - Unit 5

Operating Systems (Dr. A.P.J. Abdul Kalam Technical University)

Unit- V I/O Management & Disk Scheduling

1. Mass – Storage Structure:

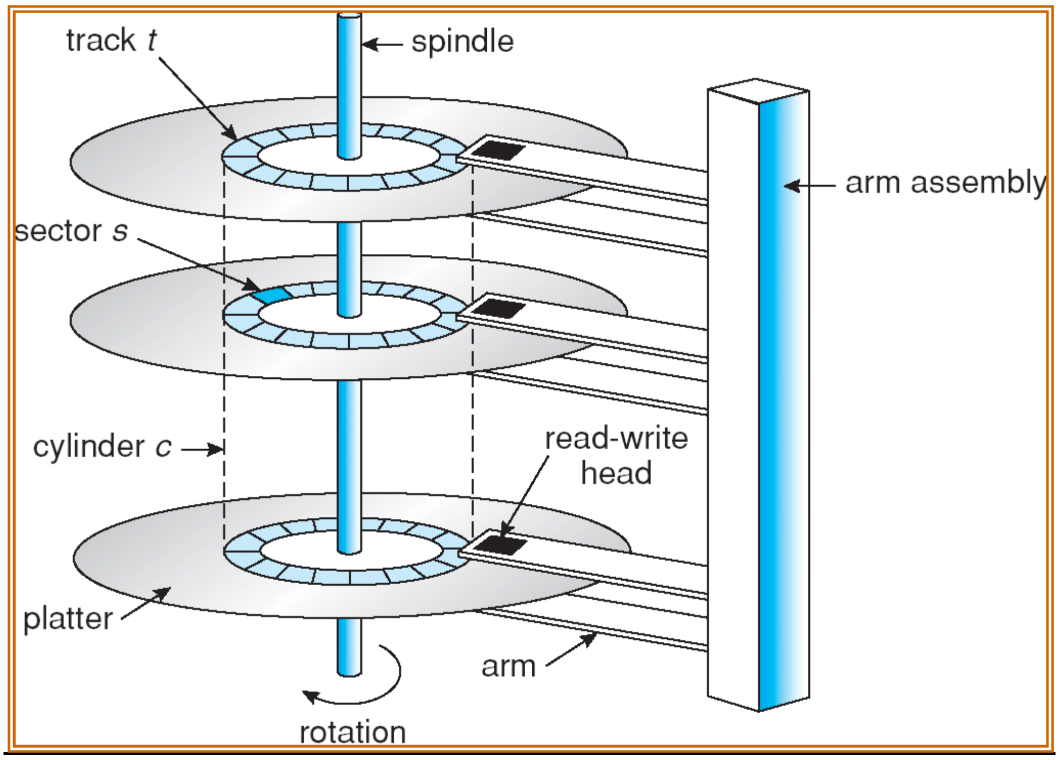
1.1 Disk Structure:

A. Magnetic disks provide bulk of secondary storage of modern computers

- Drives rotate at 60 to 200 times per second
- Transfer rate is rate at which data flow between drive and computer
- Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency).
- Head crash results from disk head making contact with the disk surface
  - That's bad

B. Disks can be removable

C. Drive attached to computer via I/O bus



1.2 Disk Scheduling:

A. The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.

B. Access time has two major components

- *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
- *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.

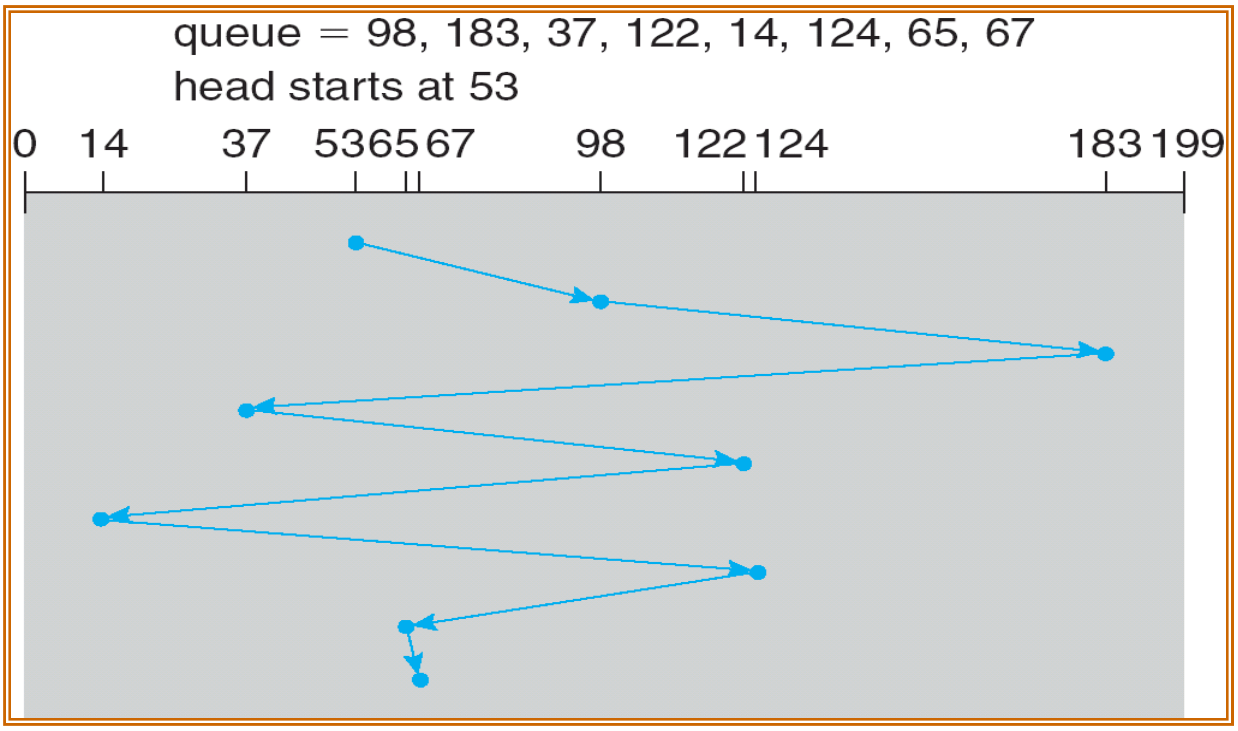
C. Minimize seek time

D. Seek time ≈ seek distance

E. Disk bandwidth 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.

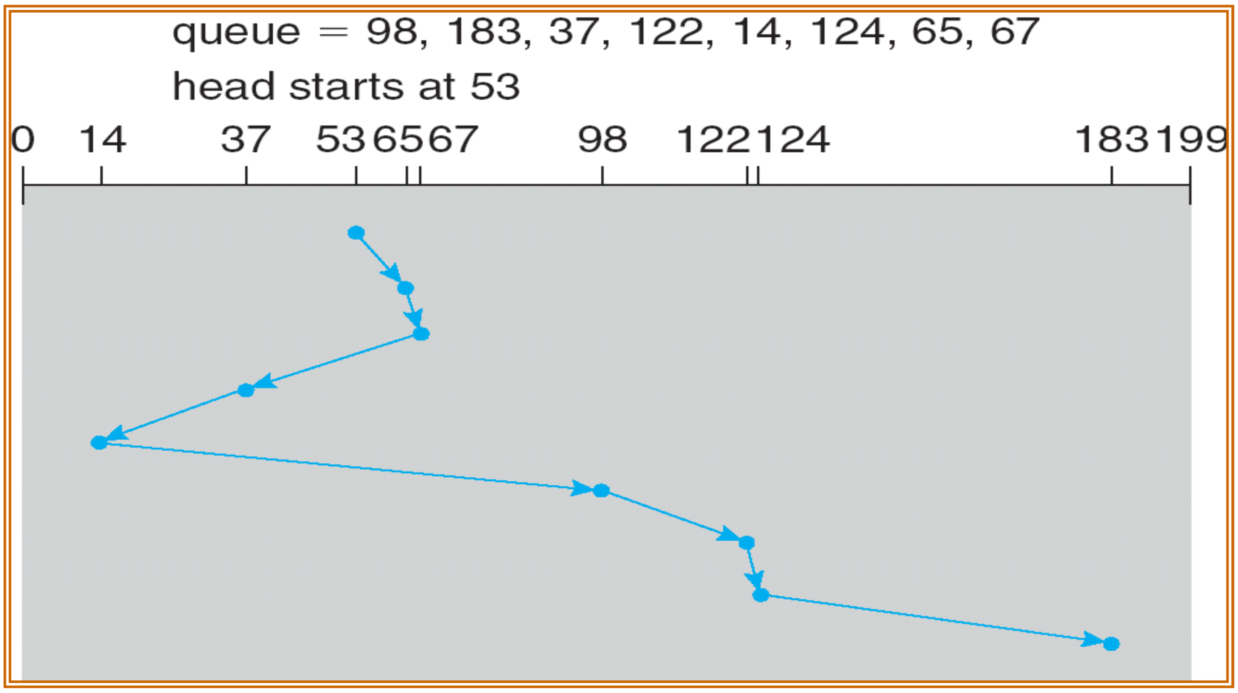
Several algorithms exist to schedule the servicing of disk I/O requests.

1.2.1 FCFS Scheduling (First Come First Served):

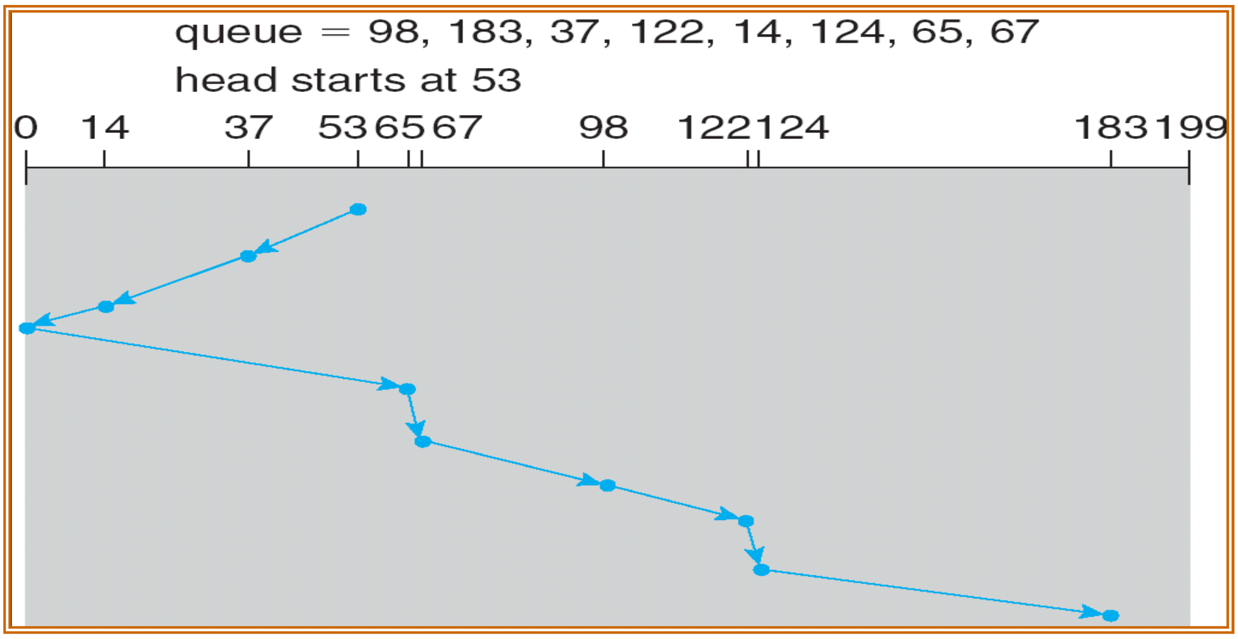


1.2.2 SSTF Scheduling (Shortest-Seek-Time-First)

- 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 some requests.

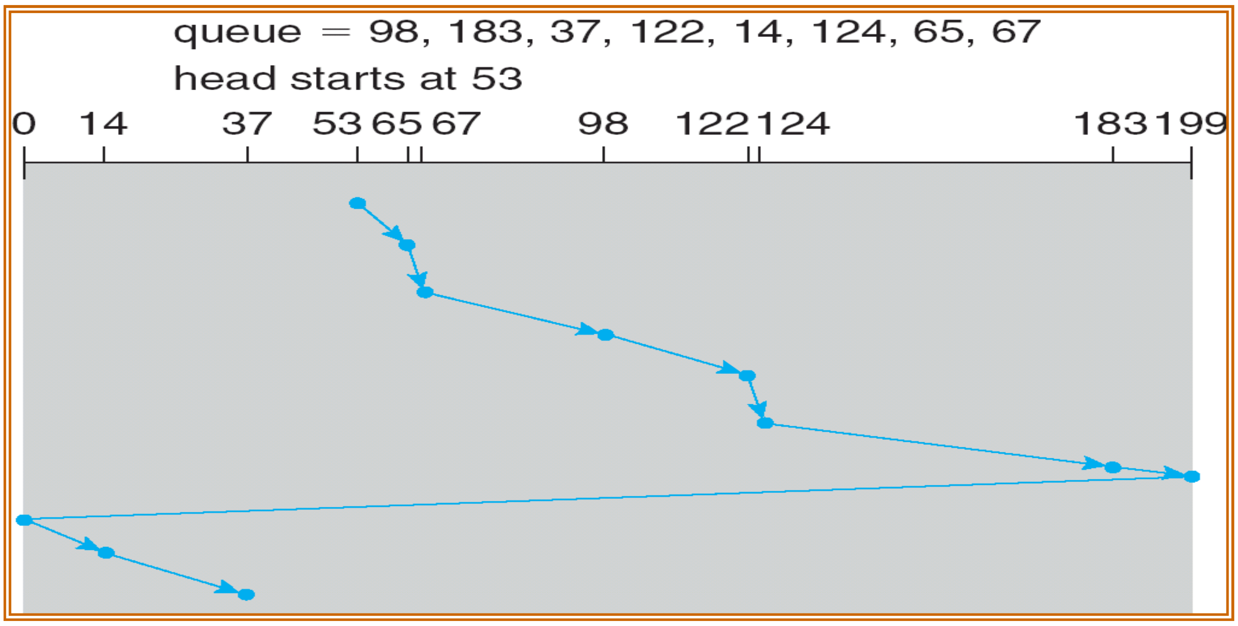


1.2.3 SCAN Scheduling: The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. Sometimes called the ***elevator algorithm*.**



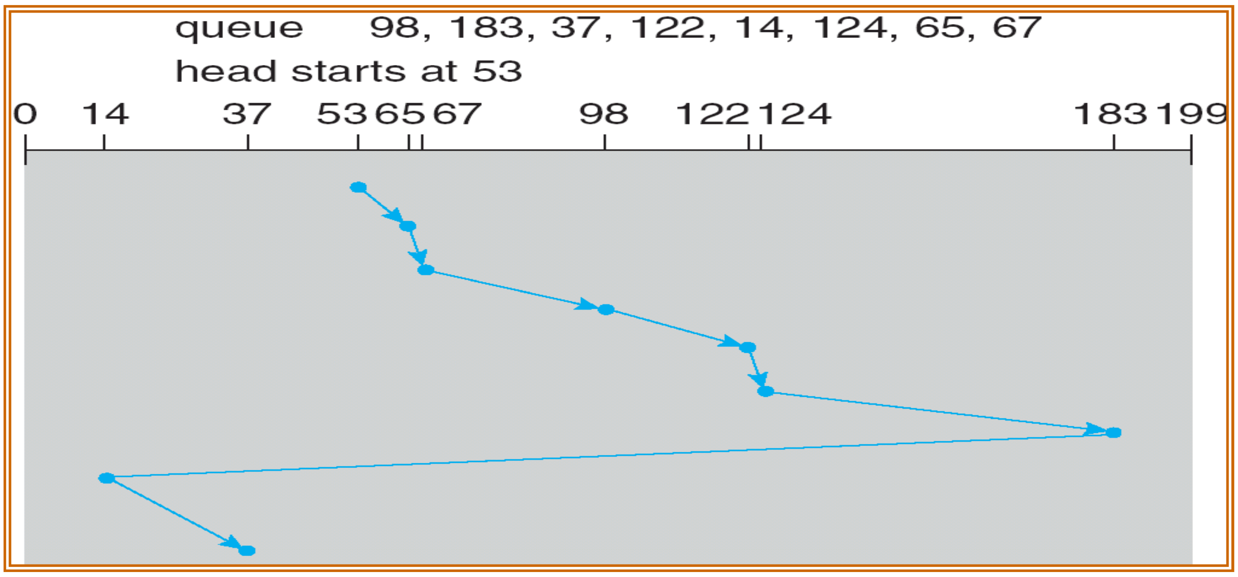
1.2.4 CSCAN Scheduling: Provides a more uniform wait time than SCAN.

- 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.



1.2.5 C LOOK Scheduling: 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.**



Criteria for Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

1.3 RAID Structure (Redundant Array of Inexpensive Disks):

- RAID – multiple disk drives provides reliability via redundancy.
- Several improvements in disk-use techniques involve the use of multiple disks **working cooperatively.**
- Disk striping uses a group of disks as one storage unit.
- RAID schemes improve performance and improve the reliability of the storage system by **storing redundant data.**
    - ***Mirroring* or *shadowing*** keeps duplicate of each disk.
    - ***Block interleaved parity*** uses much less redundancy.

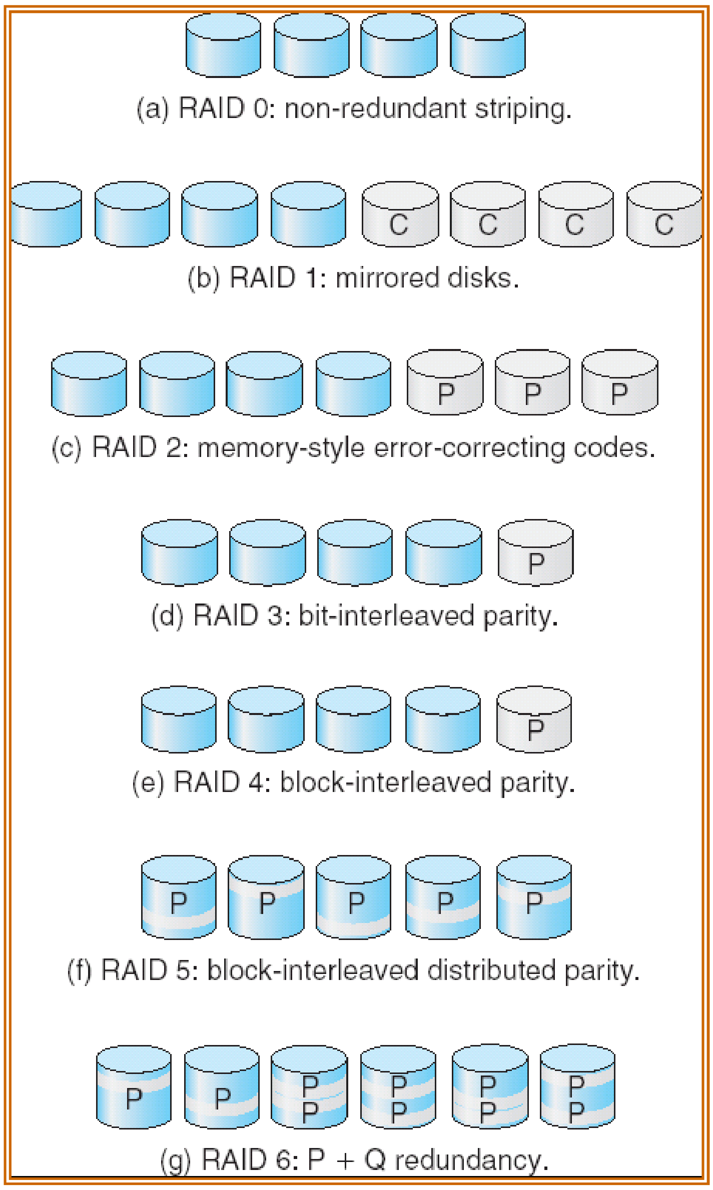
✓ RAID is arranged into **six different levels.**

i. **RAID Level 0:**

Level 0 refers to disk arrays with striping at the level of blocks but without any redundancy (such as mirroring or parity bits), as shown in Figure (a).

ii. **RAID Level 1:**

Level 1 refers to disk mirroring. Figure (b) shows a mirrored organization.



(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.

iii. RAID Level 2:

RAID level 2 is also known as **memory-style error-correctingcode (ECC) organization.** Memory systems have long detected certain errors by using parity bits. Each byte in a memory system may have a parity bit associated with it that records whether the number of bits in the byte set to 1 is even (parity = 0) or odd (parity = 1). If one of the bits in the byte is damaged (either a 1 becomes a 0, or a 0 becomes a 1), the parity of the byte changes and thus will not match the stored parity.

iv. RAID Level 3:

RAID level 3, or **bit-interleaved parity organization**, improves on level 2 by taking into account the fact that, unlike memory systems, disk controllers can detect whether a sector has been read correctly, so a single parity bit can be used for error correction as well as for detection. If one of the sectors is damaged, we know exactly which sector it is, and we can figure out whether any bit in the sector is a 1 or a 0 by computing the parity of the corresponding bits from sectors in the other disks. If the parity of the remaining bits is equal to the stored parity, the missing bit is 0; otherwise, it is 1.

RAID level 3 has **two advantages** over level 1. First, the storage overhead is reduced because only one parity disk is needed for several regular disks, whereas one mirror disk is needed for every disk in level 1. Second, since reads and writes of a byte are spread out over multiple disks with A/-way striping of data, the transfer rate for reading or writing a single block is *N* times as fast as with RAID level 1. On the negative side, RAID level 3 supports fewer I/Os per second, since every disk has to participate in every I/O request.

v. RAID Level 4:

RAID level 4, or block-interleaved parity organization, uses block-level striping, as in RAID 0, and in addition keeps a parity block on a separate disk for corresponding blocks from A! other disks.

vi. RAID Level 5:

RAID level 5, or block-interleaved distributed parity, differs from level 4 by spreading data and parity among all *N* + 1 disks, rather than storing data in *N* disks and parity in one disk. For each block, one of the disks stores the parity, and the others store data.

vii. RAID Level 6:

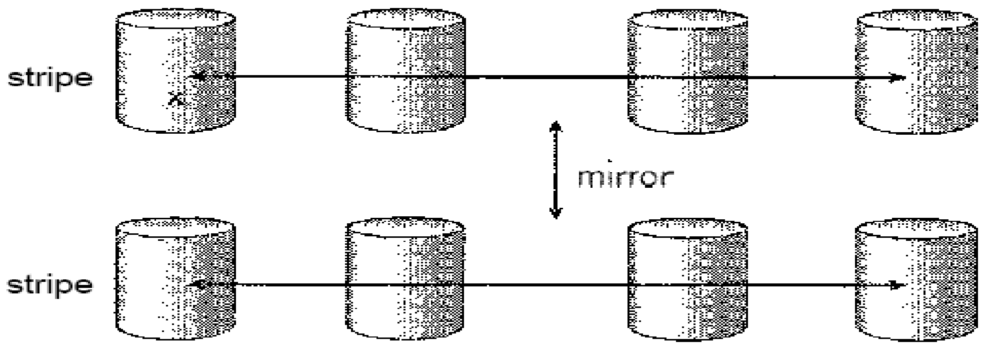
RAID level 6, also called the P + Q redundancy scheme, is much like RAID level 5 but stores extra redundant information to guard against multiple disk failures. Instead of parity, error-correcting codes such as the Reed-Solomon codes are used. In the scheme shown in Figure (g), 2 bits of redundant data are stored for every 4 bits of data— compared with 1 parity bit in level 5—and the system can tolerate two disk failures.

Que. Write short note on RAID 0+1 and RAID 1+0

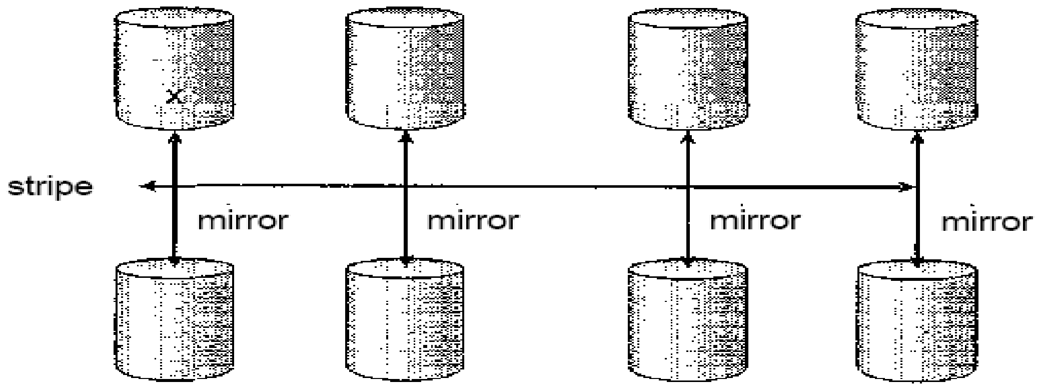
Ans.

RAID Level 0 + 1. RAID level 0 + 1 refers to a combination of RAID levels 0 and 1. RAID 0 provides the performance, while RAID 1 provides the reliability. Generally, this level provides better performance than RAID 5. It is common in environments where both performance and. reliability
are important. Unfortunately, it doubles the number of disks needed for storage, as does RAID 1, so it is also more expensive, in RAID 0 - 1, a set of disks are striped, and then the stripe is mirrored to another, equivalent stripe.

Another RAID option that is becoming available commercially is RAID level 1 + 0, in which disks are mirrored in pairs, and then the resulting mirror pairs are striped. This RAID has some theoretical advantages over RAID 0 + 1. For example, if a single disk fails in RAID 0 + 1, the entire stripe is inaccessible, leaving only the other stripe available. With a failure in RAID 1+0, the single disk is unavailable, but its mirrored pair is still available, as are all the rest of the disks



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

2. File System

2.1 File Concept (File Organization):

- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, *the file.* Files are mapped by the operating system onto physical devices.
- It contains data in type of numeric, character & binary etc..
- It has **File control block** – storage structure consisting of information about a file.
- It has two types of record structures:

Simple record structure: Lines, Fixed length, Variable length

Complex Structures: Formatted document, Relocatable load file

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

File Control Block

We will discuss about File Attributes, File Operations & File Types:

2.1.1 File Attributes:

i. Name – only information kept in human-readable form

ii. Identifier – unique tag (number) identifies file within file system

iii. Type – needed for systems that support different types

iv. Location – pointer to file location on device

v. Size – current file size

vi. Protection – controls who can do reading, writing, executing

vii. Time, date, and user identification – data for protection, security, and usage monitoring

viii. Information about files are kept in the directory structure, which is maintained on the disk

2.1.2 File Operations:

i. File is an abstract data type

ii. Create

iii. Write

iv. Read

v. Reposition within file

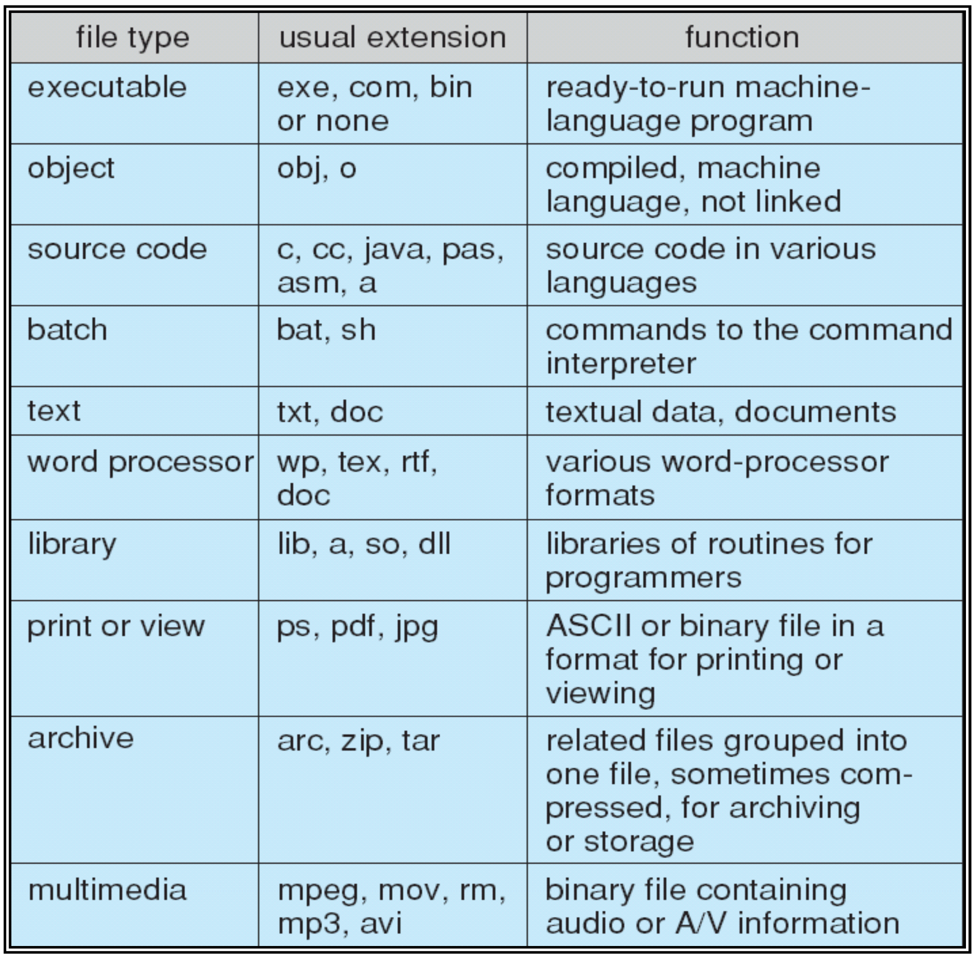
vi. Delete

vii. Truncate

viii. *Open(F_i)* – search the directory structure on disk for entry F_i, and move the content of entry to memory

ix. *Close (F_i)* – move the content of entry F_i in memory to directory structure on disk

2.1.3 File Types:

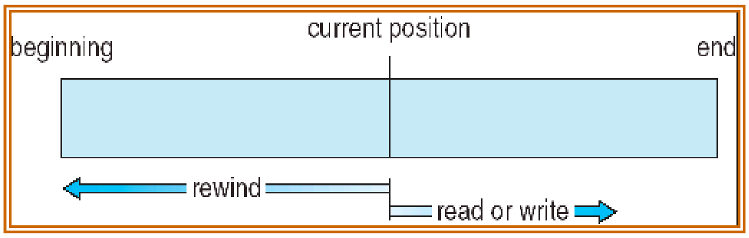


file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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 com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

2.2 Access methods of a file:

2.2.1 Sequential Access:

The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.



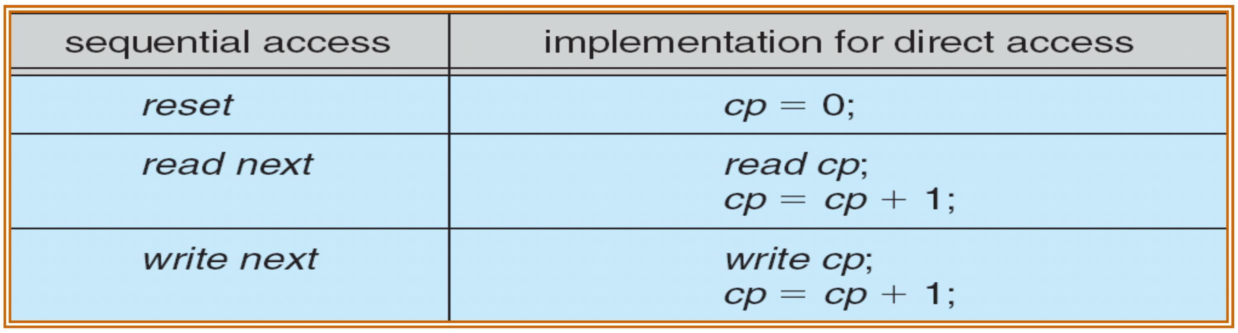
read next
write next
reset
no read after last write
(rewrite)

2.2.2 Direct Access:

Another method is **direct access** (or **relative** access). A file is made up of fixed length **logical records** that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records.

There are no restrictions on the order of reading or writing for a direct-access file.

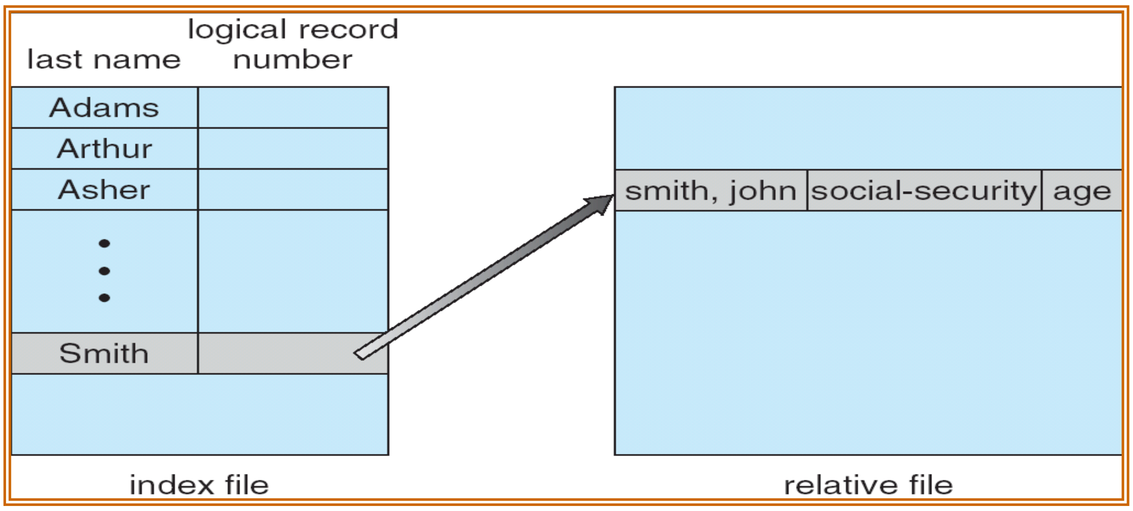
2.2.3 Simulation of Sequential Access on a Direct Access File:



sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

2.2.4 Indexed Sequential Access Method (ISAM):

indexed sequential-access method (ISAM) uses a small master index that points to disk blocks of a secondary index. The secondary index blocks point to the actual file blocks. The file is kept sorted on a defined key. To find a particular item, we first make a binary search of the master index, which provides the block number of the secondary index. This block is read in, and again a binary search is used to find the block containing the desired record.

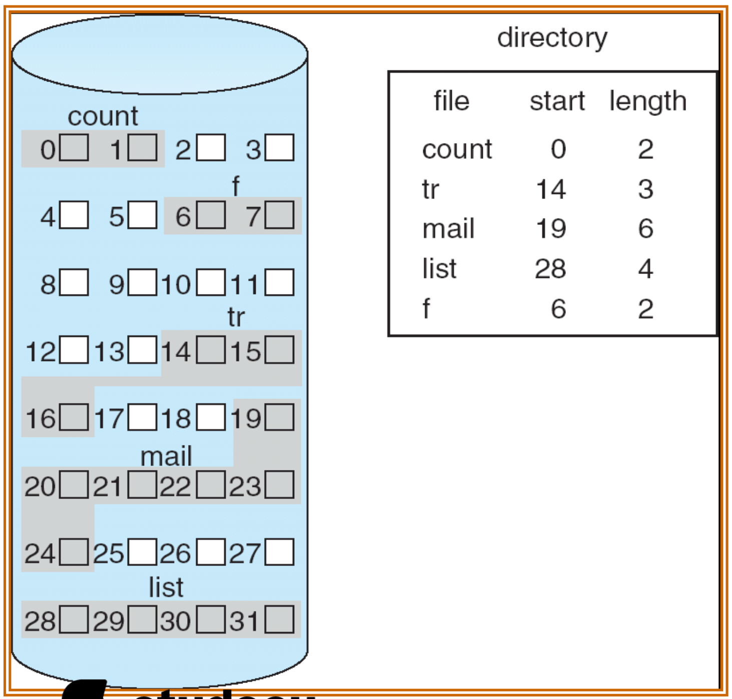


2.3 Allocation Methods:
An allocation method refers to how disk blocks are allocated for files:

(i) Contiguous allocation (ii) Linked allocation (iii) Indexed allocation

2.3.1 Contiguous Allocation

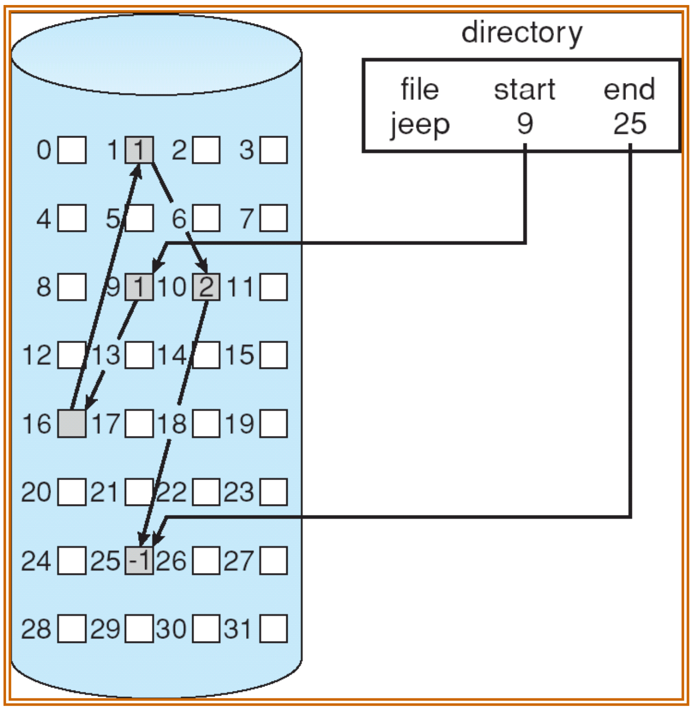
- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow



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

2.3.2 Linked Allocation

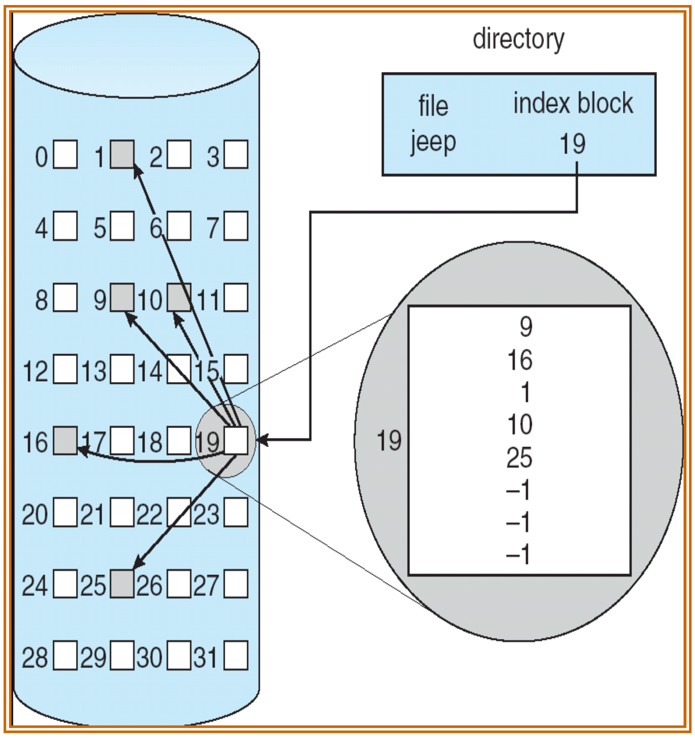
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping
- File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.



2.3.3 Indexed Allocation

- Brings all pointers together into the *index block.*
- Need index table
- Random access

- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.



2.4 File Sharing:

2.4.1 Multiple Users:

- User IDs identify users, allowing permissions and protections to be per-user.
- Group IDs allow users to be in groups, permitting group access rights.

2.4.2 Remote File Systems:

- Uses networking to allow file system access between systems
  - ➢ Manually via programs like FTP
  - ➢ Automatically, seamlessly using distributed file systems
  - ➢ Semi automatically via the world wide web
- Client-server model allows clients to mount remote file systems from servers
  - ➢ Server can serve multiple clients
  - ➢ Client and user-on-client identification is insecure or complicated
  - ➢ NFS is standard UNIX client-server file sharing protocol
  - ➢ CIFS is standard Windows protocol
  - ➢ Standard operating system file calls are translated into remote calls

Failure Modes:

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

2.5 File Protection & Access Control:

Protection mechanisms provide controlled access **by limiting the types of file access** that can be made (**Controlled Access**). Access is permitted or denied depending on several factors, one of which is the type of access requested. 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 tree its space for possible reuse.
• **List.** List the name and attributes of the file.

i. The most common approach to the protection problem is to make access dependent on the **identity of the user.**

ii. The most general scheme to implement identity dependent access is to associate with each file and directory an **access-control list (ACL)** specifying user names and the types of access allowed for each user.

iii. The main **problem with access lists is their length**. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

- **a.** Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
- **b.** The directory entry, previously of fixed size, now needs to be of variable size, resulting in more complicated space management.

iv. These problems can be resolved by use of a **condensed version of the access list**. To condense the length of the access-control list, many systems **recognize three classifications of users** in connection with each file:

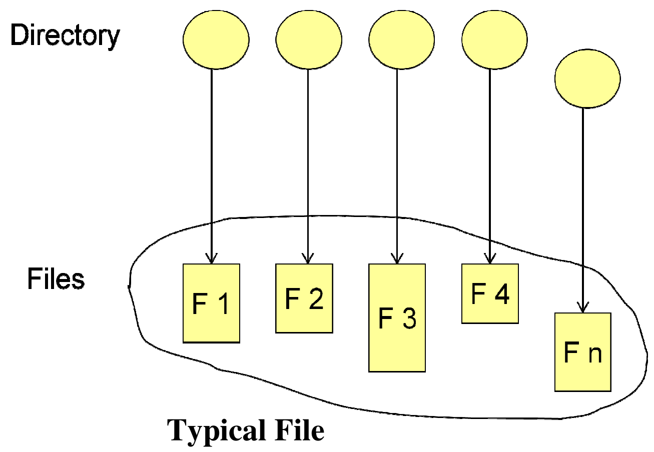
• **Owner.** The user who created the file is the owner.
• **Group.** A set of users who are sharing the file and need similar access is a group, or work group.
• **Universe.** All other users in the system constitute the universe.

2.6 Recovery

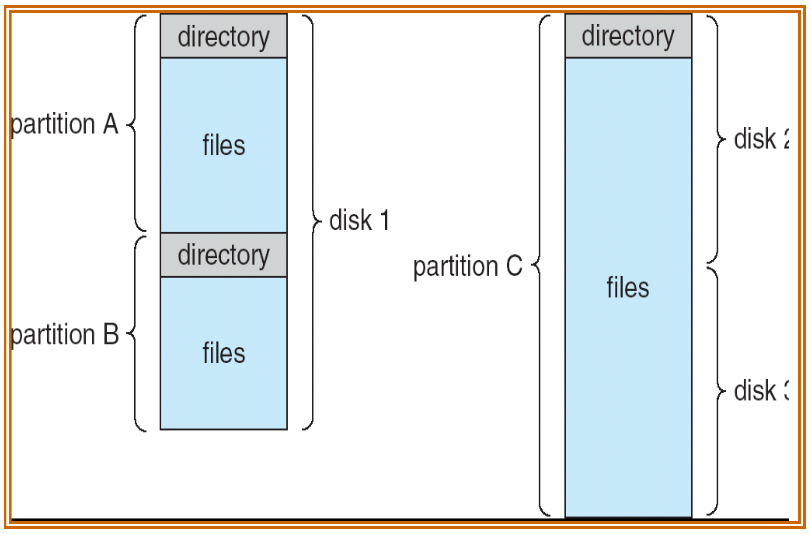
- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to **back up data** from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by restoring data from backup

2.7 Directory Structure:

- To manage all the data we need to organize them. This organization is usually **done in two parts.**
- First, **disks are split into one or more partitions**, also known as minidisks or **volumes** in the PC.
- Second, each partition contains information about files within it. This information is kept in entries in a **device directory** or volume table of contents.
- A collection of nodes containing information about all files



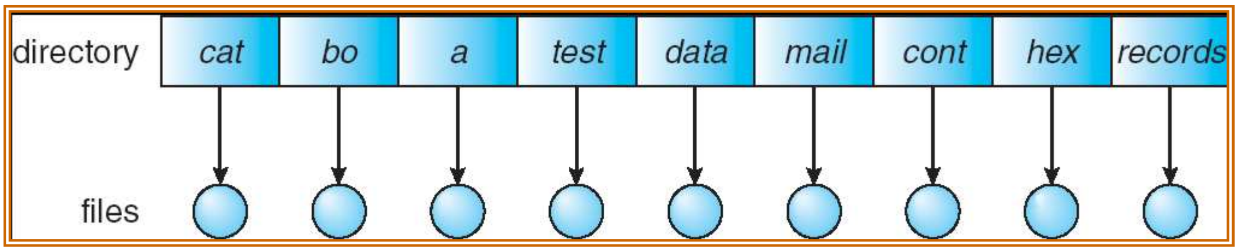
Typical File



System Organization

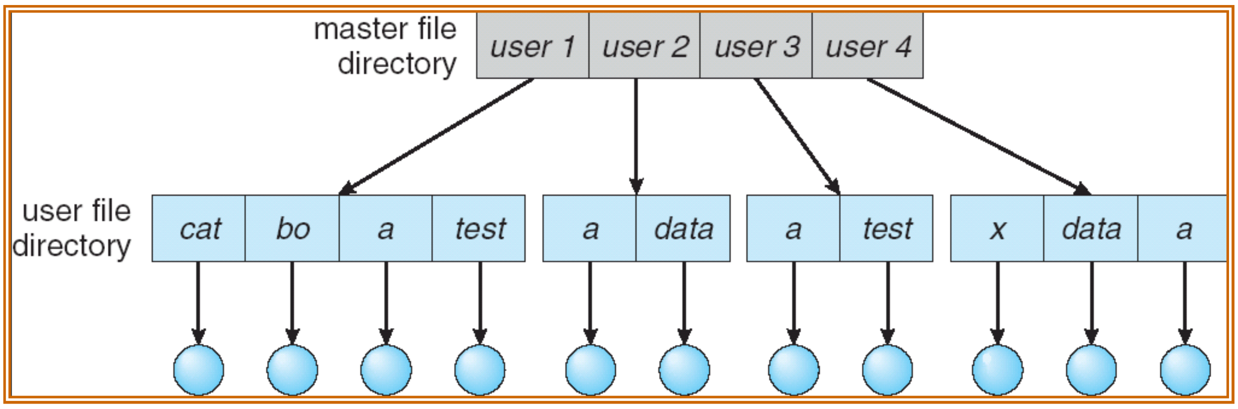
2.7.1 Single-Level Directory

- The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand.
- A single-level directory has significant limitations, Since all files are in the same directory, they must have unique names.
- It is difficult to remember the names of all the files as the number of files increases.



2.7.2 Two-Level Directory

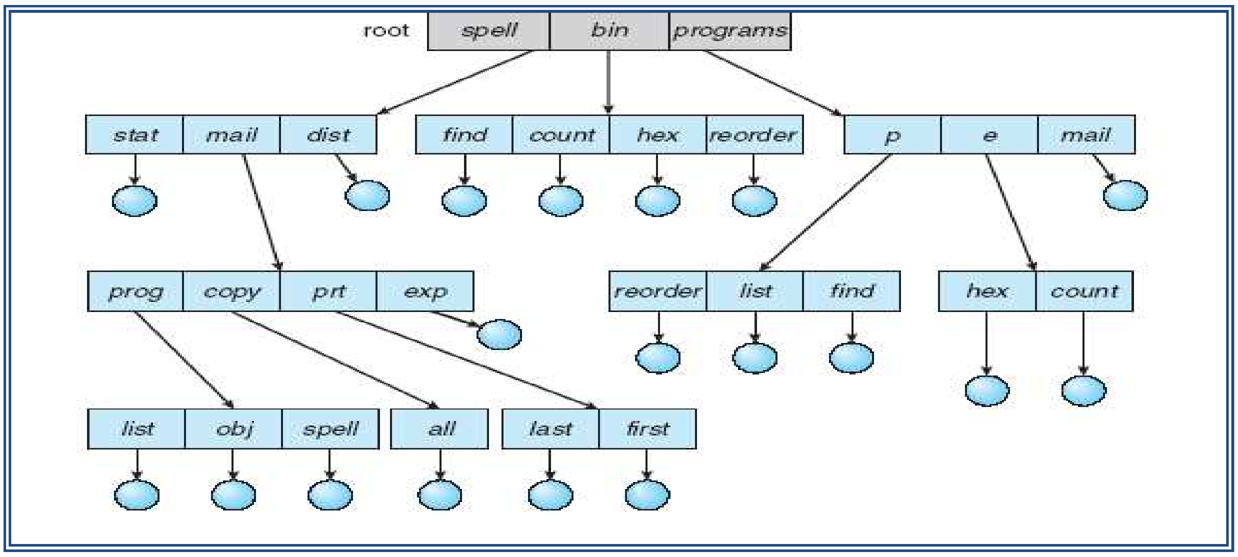
- In the two-level directory structure, **each user has his own user file directory (UFD)**. The UFDs have similar structures, but each lists only the files of a single user.
- When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.



2.7.3Tree-Structured Directory:

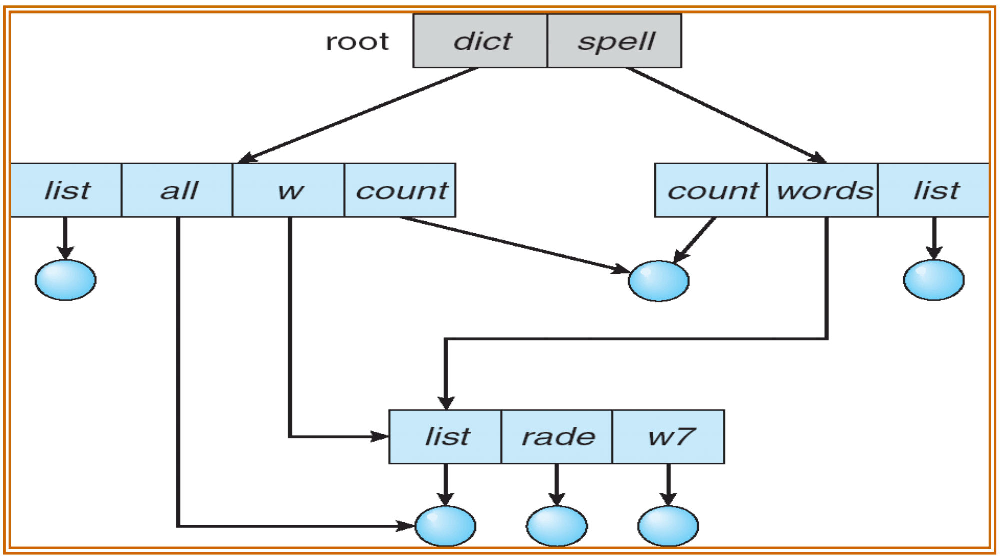
- The **natural generalization** is to extend the directory structure to a tree of arbitrary height. **This generalization allows users to create their own subdirectories and to organize their files accordingly.**

- A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. **All directories have the same internal format.**



2.7.4 Acyclic Graph Directory:

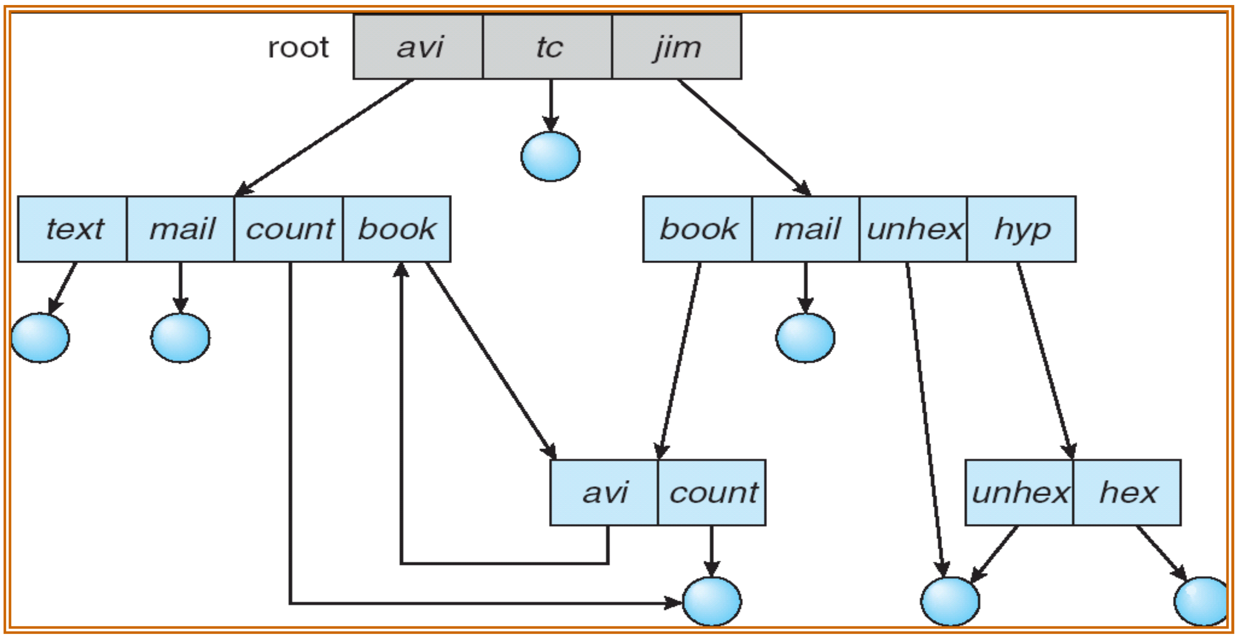
A tree structure prohibits the sharing of files or directories. An **acyclic graph** —that is, a graph with no cycles—**allows directories to share subdirectories and files** The *same* file or subdirectory may be in two different directories. **The acyclic graph is a natural generalization of the tree-structured directory scheme.**



2.7.5 General Graph Directory:

A serious problem with using an acyclic-graph structure is ensuring that there are no cycles. If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results. It should be fairly easy to see that simply adding new files and subdirectories to an existing tree-structured

directory preserves the tree-structured nature. However, **when we add links to an existing tree-structured directory, the tree structure is destroyed, resulting in a simple graph structure.**



2.8 Directory Implementation:

2.8.1 Linear List: The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute. To create a new file., we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory.

2.8.2 Hash Table: linear list with hash data structure.

- It decreases directory search time
- collisions – situations where two file names hash to the same location
- fixed size

Q. Explain Bit Vector & Linked List implementation for Free Space Management.

Ans. 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.

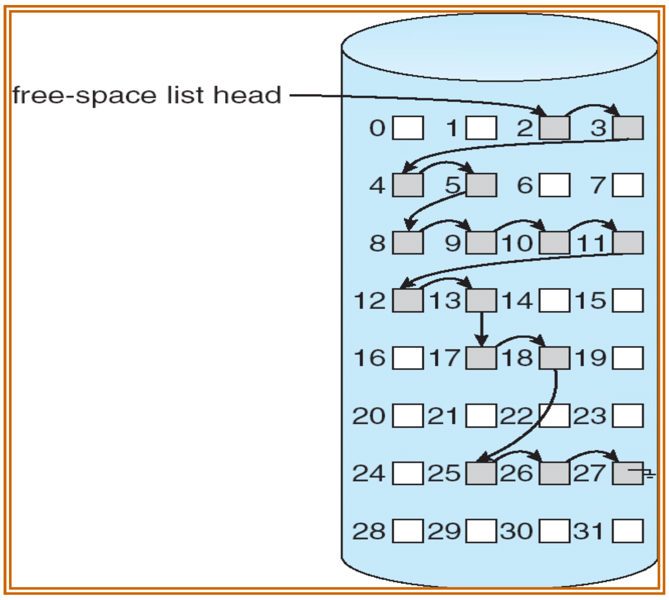
I. Bit Vector (Bit Map):

- 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 blocks **2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25,26, and 27** are free and the rest of the blocks are allocated. The free-space bit map would be

001111001111110001100000011100000

II. 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.



3. I/O management:

3.1 I/O Hardware:

Computers operate a great many kinds of devices. Most fit into the general categories of **storage devices** (disks, tapes), **transmission devices** (network cards, modems), and **human-interface devices** (screen, keyboard, mouse).

Other devices are more **specialized**, such as the steering of a military **fighter jet or a space shuttle**. In these aircraft, a human gives input to the flight computer **via a joystick** and foot pedals, and the computer sends output commands that cause motors to move rudders, flaps, and thrusters.

Introduction:

- The device communicates with the machine via a connection point (or **port).**
- If devices use a common set of wires, the connection is called a *bus.* A **bus** is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.
- In **terms of the electronics**, the messages are conveyed by **patterns of electrical voltages** applied to the wires with defined timings. When 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.
- A **typical PC bus structure** is shown in figure. This figure shows a **PCI bus** (Peripheral Component Interconnect, the common PC system bus) that connects the processor-memory subsystem to the **fast devices** . **An expansion bus that connects relatively slow devices** such as the keyboard and serial and parallel ports.
- A **controller** 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 (or portion of a chip) in the computer that controls the signals on the wires of a serial port. By contrast, a **SCSI bus controller is not simple**. Because the SCSI protocol is complex, the SCSI bus controller is often implemented as a separate circuit board (or a **host** adapter) that plugs into the computer.

Q. How can the processor give commands and data to a controller to accomplish an I/O transfer?

Ans. Controller has one or more registers for data and control signals. The **processor communicates** with the controller by **reading and writing bit patterns in these registers**.

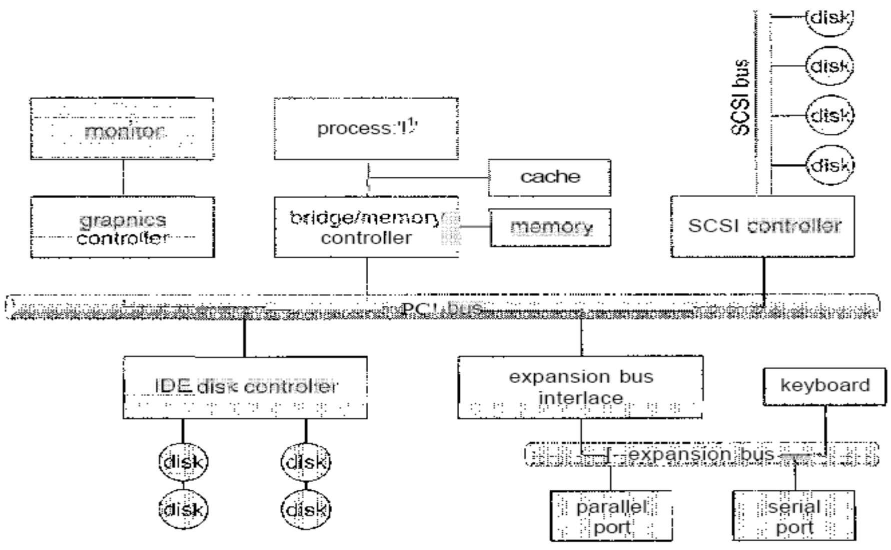
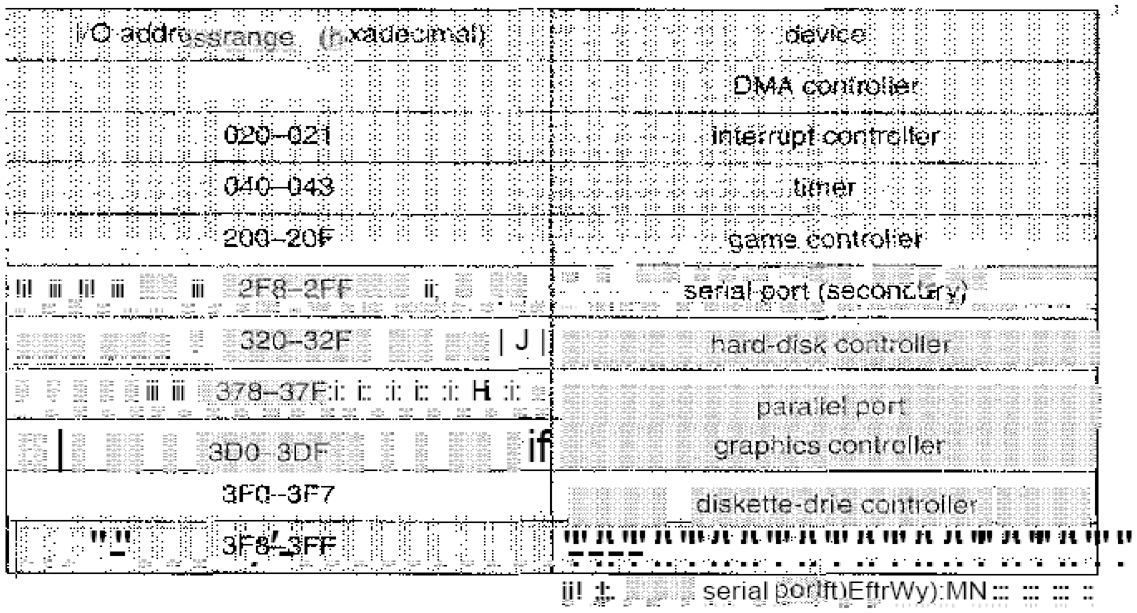


Figure **13.1** A typical PC bus structure.

The processor communicates with the controller by reading and writing bit patterns in these registers. One way in which this communication can occur is through the use of special I/O instructions that specify the **transfer of a byte or word to an I/O port address**. The **I/O instruction triggers bus lines to select the proper device** and to move bits into or out of a device register. Alternatively, the device controller can support memory-mapped I/O. In this case, 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, called the (1) status, (2) control, (3) data-in, and (4) data-out registers.

- The **data-in** register is read by the host to get input.
- The **data-out** register is written by the host to send output.
- The **status** register contains bits that can be **read by the host**. These bits indicate states, such as whether the **current command has completed**, whether a byte is available to be read from the data-in register, and whether a device error has occurred.
- The **control register can be written by the host to start a command or to change the mode of a device**. For instance, a certain bit in the control register of a serial port chooses between full-duplex and half-duplex communication, another bit enables parity checking, a third bit sets the word length to 7 or 8 bits, and other bits select one of the speeds supported by the serial port.



I/O address range (hexadecimal)	device
	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drie controller
3F8–3FF	serial port (primary)

Figure 13.2 Device I/O port locations on PCs (partial).

3.1.1 Polling:

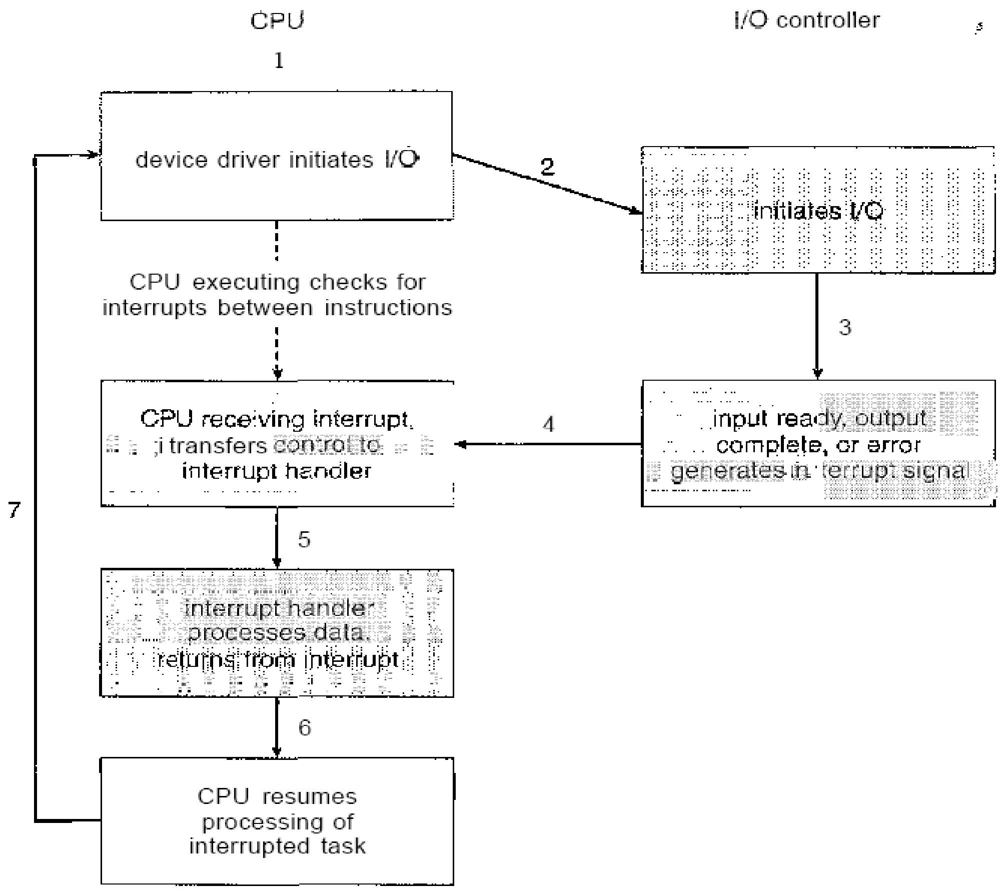
The complete protocol for interaction between the host and the controller (Handshaking Notion):

1. The host repeatedly reads the ***busy*** **bit until that bit becomes clear.**
2. The host sets the *write,* bit in the *command* register and writes **a byte** into the ***data-out*** **register.**
3. The host sets the ***command-ready*** bit.
4. When the controller notices that the *command-ready* bit is set, it sets the ***busy*** **bit.**
5. 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**.
6. 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.

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,

3.1.2 Interrupts:

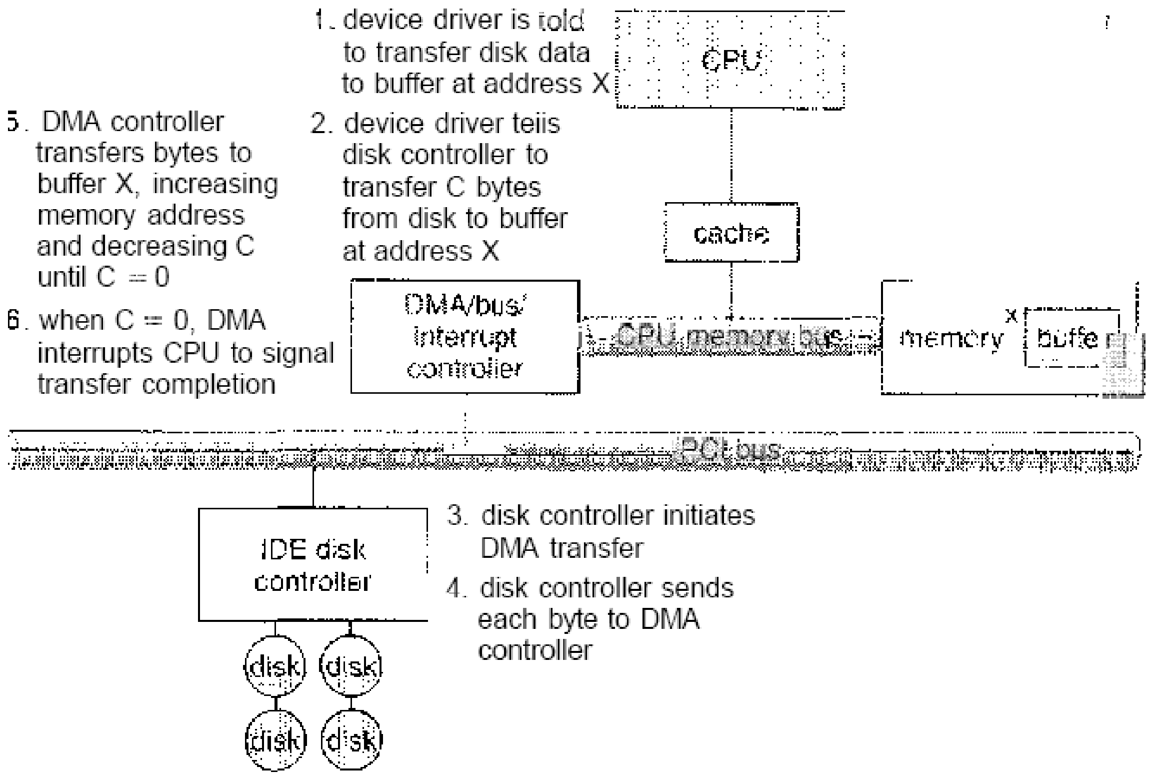
- The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction.
- When the CPU detects that a controller has asserted a signal on the interrupt request line, the CPU performs a state save and jumps to the **interrupt handler** routine at a fixed 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 the CPU to the execution state prior to the interrupt.
- We say that the device controller ***raises*** **an interrupt by asserting a signal** on the interrupt request line, the CPU *catches* the interrupt and *dispatches* it to the interrupt handler, and the handler ***clears*** **the interrupt by servicing the device.**


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

3.1.3 Direct Memory Access (DMA)

For a device that does large transfers, such as a disk drive, it seems wasteful to use an expensive general-purpose processor to watch status bits and to feed data into a controller register one byte at a time—a process termed **programmed I/O (PIO).** Many computers avoid burdening the main CPU with PIO by offloading some of this work to a special-purpose processor called a **direct-memory-access (DMA)** controller.

To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred. All the steps are explained in the figure:

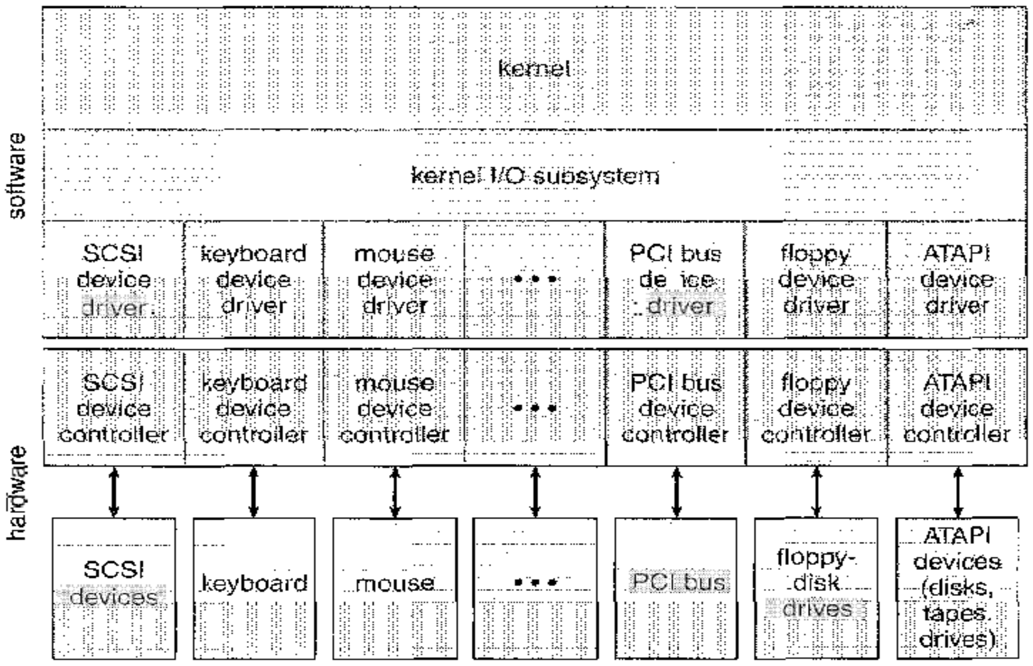

1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
DMA/bus/ interrupt controller
CPU memory bus
memory
X
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

Steps in a DMA transfer.

Handshaking between the DMA controller and the device controller is performed via a pair of wires called DMA-request and DMA-acknowledge.

3.2 Application I/O Interface:

In this section, we discuss structuring techniques and interfaces for the operating system that enable I/O devices to be treated in a standard, uniform way. We explain, for instance, how an application can open a file on a disk without knowing what kind of disk it is and how new disks and other devices can be added to a computer without disruption of the operating system.

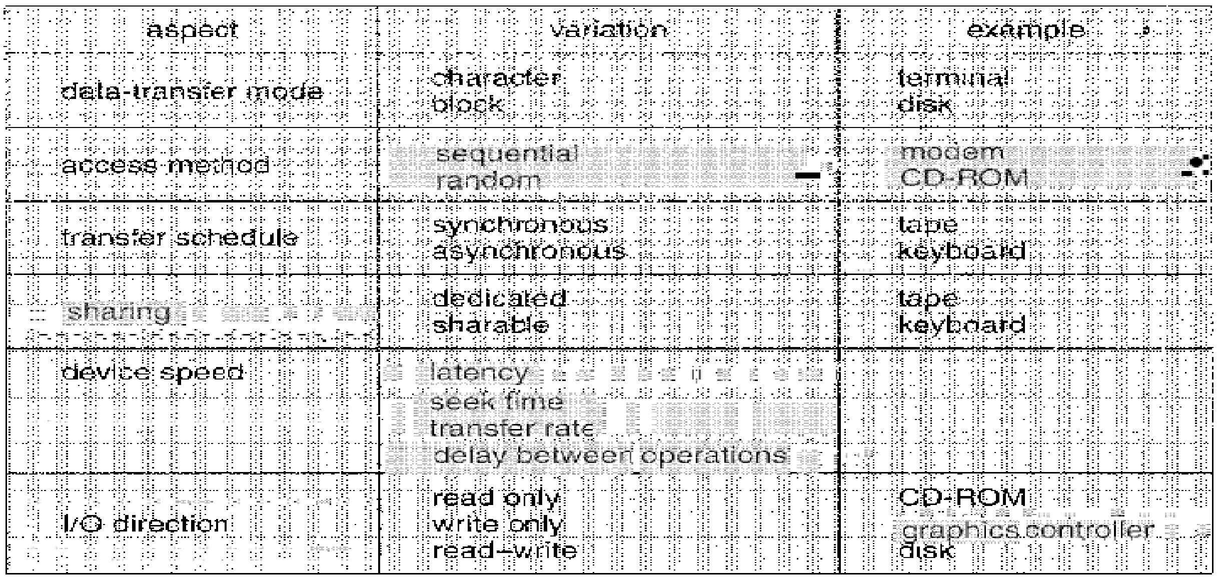


A kernel I/O structure.

3.2.1 Block & Character Devices:

The **block-device interface** captures all the aspects necessary for accessing disk drives and other block-oriented devices. The device is expected to understand commands such as **read ()** and **write ();** if it is a random-access device, it is also expected to have a seek() command to specify which block to transfer next.

A keyboard is an example of a device that is accessed through a c**haracterstream interface.** The basic system calls in this interface enable an application to **get()** or **put()** one character.



aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Characteristics of I/O devices.

3.2.2 Network Devices:

- Network **socket interface** is used for network I/O interface.
- The system calls in the socket interface enable an application to create a socket, to connect a local socket to a remote address to the system calls in the socket interface enable an application to create a socket, to connect a local socket to a remote address .
- To support the implementation of servers, the socket interface also provides a function called **select ()** that manages **a set of sockets**.

3.2.3 Clocks and Timers:

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

• Give the current time.

• Give the elapsed time.

• Set a timer to trigger operation X at time *T*.

These functions are used heavily by the operating system, as well as by time sensitive applications.

Unfortunately, the system calls that implement these functions are not standardized across operating systems. The hardware to measure elapsed time and to trigger operations is called a **programmable interval timer.**

In most computers, the hardware clock is constructed from a high frequency counter.

3.2.4 Blocking & Non Blocking I/O:

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.

Nonblocking I/O: One example is a user interface that receives keyboard and mouse input while processing and displaying data on the screen. Another example is a video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display.

The difference between nonblocking and asynchronous system calls is that a nonblocking readQ returns immediately with whatever data are available — the full number of bytes requested, fewer, or none at all. An asynchronous read() call requests a transfer that will be performed in its entirety but that will complete at some future time.

3.3 Kernel I/O Subsystem:

Kernels provide many services related to I/O: Several services—scheduling, buffering, caching, spooling, device reservation, and error handling.

3.3.1 I/O Scheduling:

To schedule a set of I/O requests means to determine a good order in which to execute them. **Example:** Application 1 requests a block near the end of the disk, application 2 requests one near the beginning, and application 3 requests one in the middle of the disk. The operating system can reduce the distance that the disk arm travels by serving the applications in the order 2, 3,1.

The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

3.3.2 Buffering

A **buffer** is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons.

i. **One reason is to cope with a speed mismatch between the producer and consumer of a data stream.** Suppose, for example, that a file is being received via modem for storage on the hard disk. The modem is about a thousand times slower than the hard disk. So a buffer is created in main memory to accumulate the bytes received from the modem.

ii. **A second use of buffering is to adapt between devices that have different data-transfer sizes.** Such disparities are especially common in computer networking, where buffers are used widely for fragmentation and reassembly of messages. At the sending side, a large message is fragmented into small network packets. The packets are sent over the network, and the receiving side places them in a reassembly buffer to form an image of the source data.

iii. **A third use of buffering is to support copy semantics for application I/O.**

3.3.3 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. For instance, the instructions of the currently running process are stored on disk, cached in physical memory, and copied gain in the CPU's secondary and primary caches. **The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, just holds a copy on faster storage of an item that resides elsewhere.**

3.3.4 Spooling & Device Reservation

A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. Although 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 operating system solves this problem by intercepting all output to the printer.

3.3.5 Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors, so that a complete system failure is not the usual result of each minor mechanical glitch.

For instance, a failure of a **SCSI device** is reported by the **SCSI protocol in three levels of detail:**

i. A sense key
ii. A additional sense code
iii. Additional sense code qualifier