Unit 4: storage management (8)

4.1 overview of mass-storage structure

4.2 Disk structure

4.3 Disk scheduling

4.4 RAID structure

4.5 File concept and access methods

4.6 Directory and disk structure

4.7 File system structure

4.8 File system implementation

4.9 Directory Implementation

4.10 I/O system overview

4.11 I/O hardware

Lab Work (12)

- Demonstrate directory and File attributes
- Simulate Disk scheduling algorithm and file management techniques

# Mass Storage Structure

The basic idea of Mass Storage is to create a Data Backup or Data Recovery System.

Along with computer systems, definitions of mass storage technologies and tactics have changed. The earliest and most basic mass storage techniques date back to the era of main frame supercomputers, according to experts. Punch cards, Hollerith cards, and other relatively similar manual storage medium are examples of this Mass Storage Media these days. Today, mass storage may include several kinds of hard disks or solid-state storage devices, as well as tape drives and other physical data storage devices.

The concepts of data backup and data recovery are frequently linked to mass storage media. The biggest Business Companies will make plans for recording, storing, and backing up all accessible data, which calls for a lot more mass storage media than what factory-direct gear can provide. This suggests a method for handling continuous mass storage that uses tape or other media. Other kinds of mass storage could function well as a data storage plan for a big network or a bunch of mobile distant devices. To backup data on a portable tablet that doesn't have a lot of internal capacity, for instance, mass storage for a tablet might require the usage of flash or Universal Serial Bus media (USB Media).

The Mass Storage Structure Devices are:

1. Magnetic Disks
2. Solid State Disks
3. Magnetic Tapes

## Disk Structure

- The traditional head-sector-cylinder, HSC numbers are mapped to linear block addresses by numbering the first sector on the first head on the outermost track as sector 0. Numbering proceeds with the rest of the sectors on that same track, and then the rest of the tracks on the same cylinder before proceeding through the rest of the cylinders to the center of the disk. In modern practice these linear block addresses are used in place of the HSC numbers for a variety of reasons:
    1. The linear length of tracks near the outer edge of the disk is much longer than for those tracks located near the center, and therefore it is possible to squeeze many more sectors onto outer tracks than onto inner ones.
    2. All disks have some bad sectors, and therefore disks maintain a few spare sectors that can be used in place of the bad ones. The mapping of spare sectors to bad sectors in managed internally to the disk controller.
    3. Modern hard drives can have thousands of cylinders, and hundreds of sectors per track on their outermost tracks. These numbers exceed the range of HSC numbers for many ( older ) operating systems, and therefore disks can be configured for any convenient combination of HSC values that falls within the total number of sectors physically on the drive.
- There is a limit to how closely packed individual bits can be placed on a physical media, but that limit is growing increasingly more packed as technological advances are made.
- Modern disks pack many more sectors into outer cylinders than inner ones, using one of two approaches:
    - With ***Constant Linear Velocity, CLV,*** the density of bits is uniform from cylinder to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders, causing the rate of bits passing under the read-write head to remain constant. This is the approach used by modern CDs and DVDs.
    - With ***Constant Angular Velocity, CAV,*** the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders. ( These disks would have a constant number of sectors per track on all cylinders. )

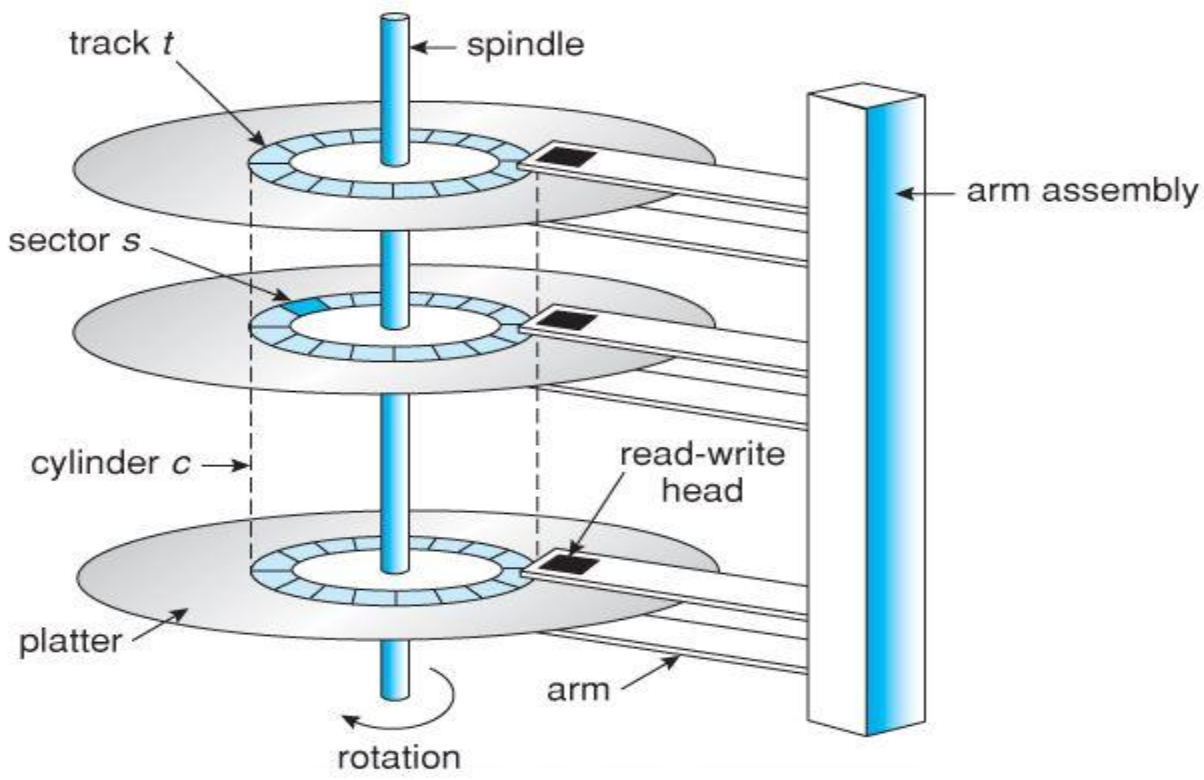



**Figure 10.1 - Moving-head disk mechanism.**

## *What is Disk Scheduling Algorithm?*

A Process makes the I/O requests to the operating system to access the disk. Disk Scheduling Algorithm manages those requests and decides the order of the disk access given to the requests.

### *Important Terms related to Disk Scheduling Algorithms*

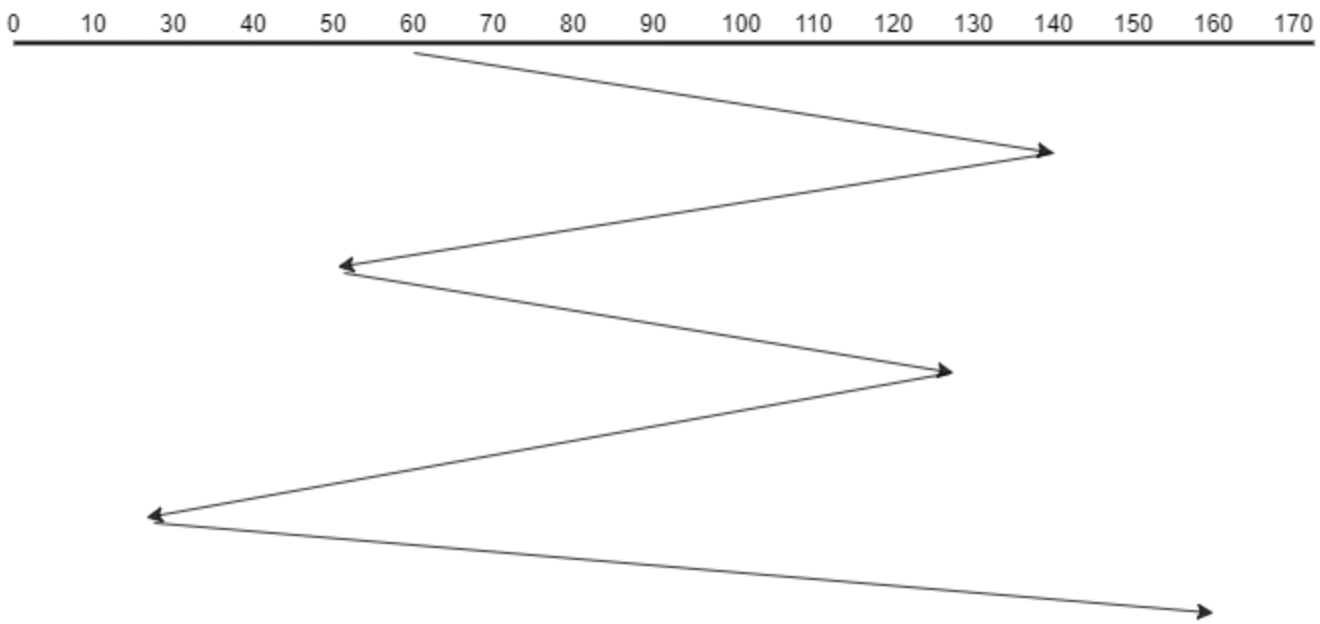
- **Seek Time -** It is the time taken by the disk arm to locate the desired track.
- **Rotational Latency -** The time taken by a desired sector of the disk to rotate itself to the position where it can access the Read/Write heads is called Rotational Latency.
- **Transfer Time -** It is the time taken to transfer the data requested by the processes.
- **Disk Access Time -** Disk Access time is the sum of the Seek Time, Rotational Latency, and Transfer Time.

### *Disk Scheduling Algorithms*

1. First Come First Serve (FCFS)

In this algorithm, the requests are served in the order they come. Those who come first are served first. This is the simplest algorithm.

Eg. Suppose the order of requests are 70, 140, 50, 125, 30, 25, 160 and the initial position of the Read-Write head is 60.

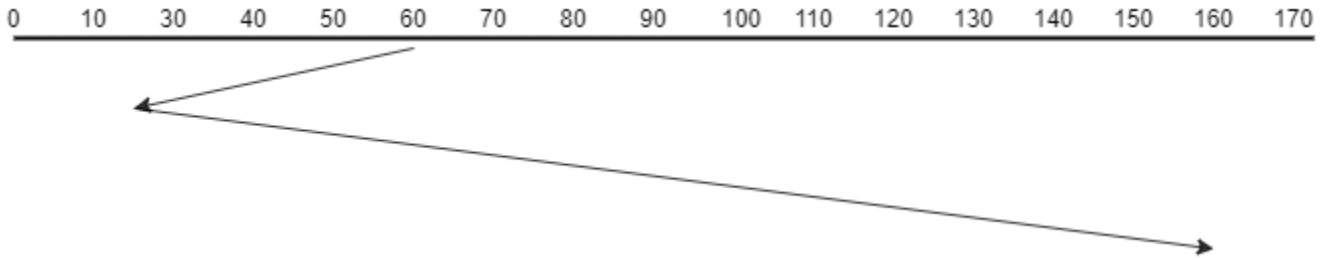



Seek Time = Distance Moved by the disk arm = (140-70)+(140-50)+(125-50)+(125-30)+(30-25)+(160-25)=480

### 2. Shortest Seek Time First (SSTF)

In this algorithm, the shortest seek time is checked from the current position and those requests which have the shortest seek time is served first. In simple words, the closest request from the disk arm is served first.

Eg. Suppose the order of requests are 70, 140, 50, 125, 30, 25, 160 and the initial position of the Read-Write head is 60.

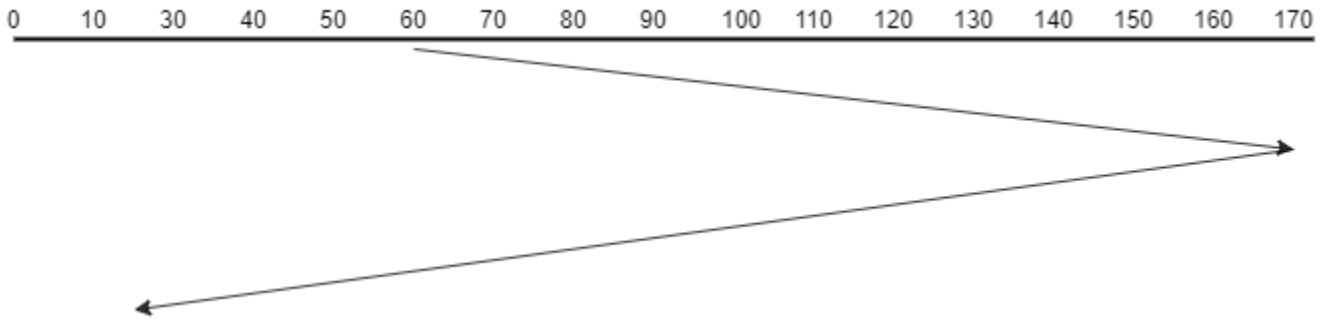



Seek Time = Distance Moved by the disk arm = (60-50)+(50-30)+(30-25)+(70-25)+(125-70)+(140-125)+(160-125)=270

### 3. SCAN

In this algorithm, the disk arm moves in a particular direction till the end and serves all the requests in its path, then it returns to the opposite direction and moves till the last request is found in that direction and serves all of them.

Eg. Suppose the order of requests are 70, 140, 50, 125, 30, 25, 160 and the initial position of the Read-Write head is 60. And it is given that the disk arm should move towards the larger value.

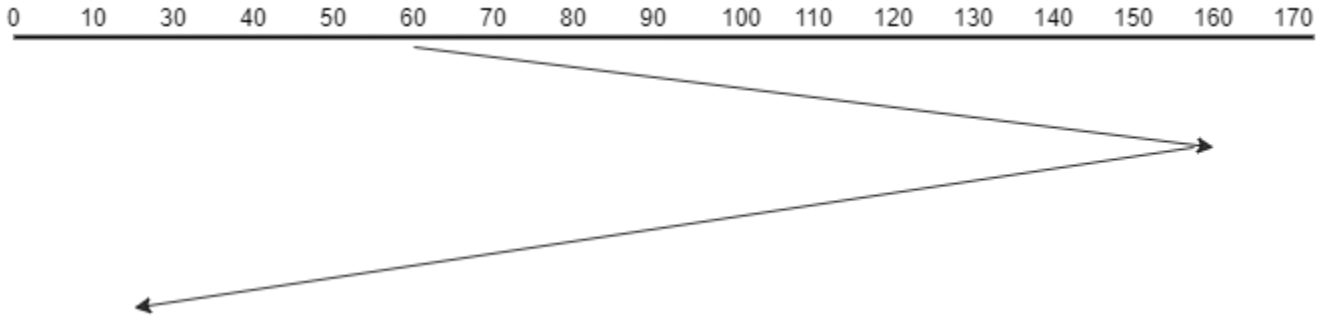



Seek Time = Distance Moved by the disk arm = (170-60)+(170-25)=255

### 4. LOOK

In this algorithm, the disk arm moves in a particular direction till the last request is found in that direction and serves all of them found in the path, and then reverses its direction and serves the requests found in the path again up to the last request found. The only difference between SCAN and LOOK is, it doesn't go to the end it only moves up to which the request is found.

Eg. Suppose the order of requests are 70, 140, 50, 125, 30, 25, 160 and the initial position of the Read-Write head is 60. And it is given that the disk arm should move towards the larger value.

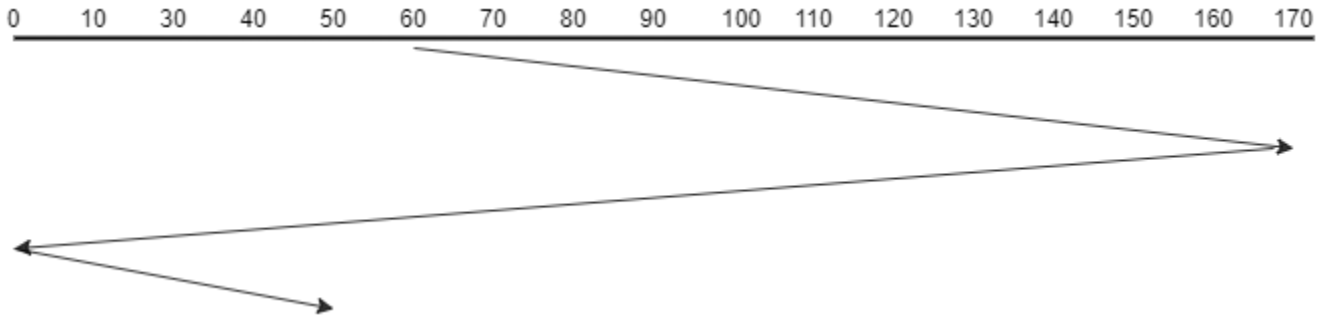



Seek Time = Distance Moved by the disk arm = (170-60)+(170-25)=235

## 5. C-SCAN

This algorithm is the same as the SCAN algorithm. The only difference between SCAN and C-SCAN is, it moves in a particular direction till the last and serves the requests in its path. Then, it returns in the opposite direction till the end and doesn't serve the request while returning. Then, again reverses the direction and serves the requests found in the path. It moves circularly.

Eg. Suppose the order of requests are 70, 140, 50, 125, 30, 25, 160 and the initial position of the Read-Write head is 60. And it is given that the disk arm should move towards the larger value.

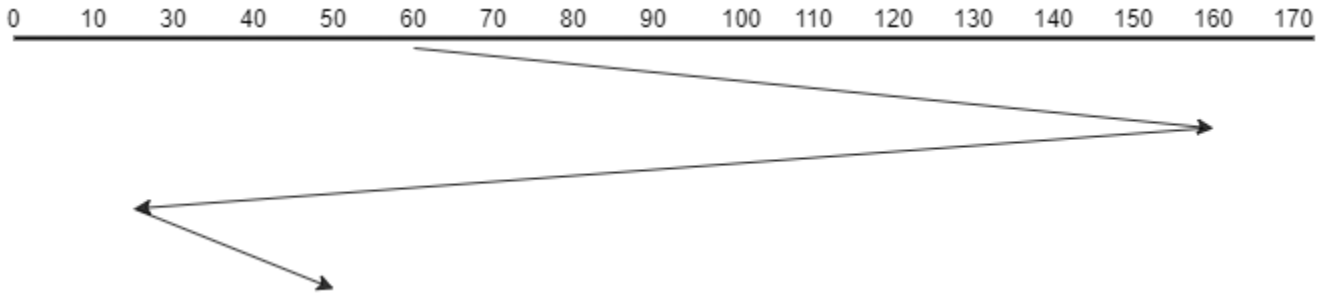



Seek Time = Distance Moved by the disk arm = (170-60)+(170-0)+(50-0)=330

## 6. C-LOOK

This algorithm is also the same as the LOOK algorithm. The only difference between LOOK and C-LOOK is, it moves in a particular direction till the last request is found and serves the requests in its path. Then, it returns in the opposite direction till the last request is found in that direction and doesn't serve the request while returning. Then, again reverses the direction and serves the requests found in the path. It also moves circularly.

Eg. Suppose the order of requests are 70, 140, 50, 125, 30, 25, 160 and the initial position of the Read-Write head is 60. And it is given that the disk arm should move towards the larger value.




Seek Time = Distance Moved by the disk arm = (160-60)+(160-25)+(50-25)=260

## RAID:

RAID is a technique that makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy, or both. The term was coined by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987.

# Why Data Redundancy?

Data redundancy, although taking up extra space, adds to disk reliability. This means, in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation. On the other hand, if the data is spread across just multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

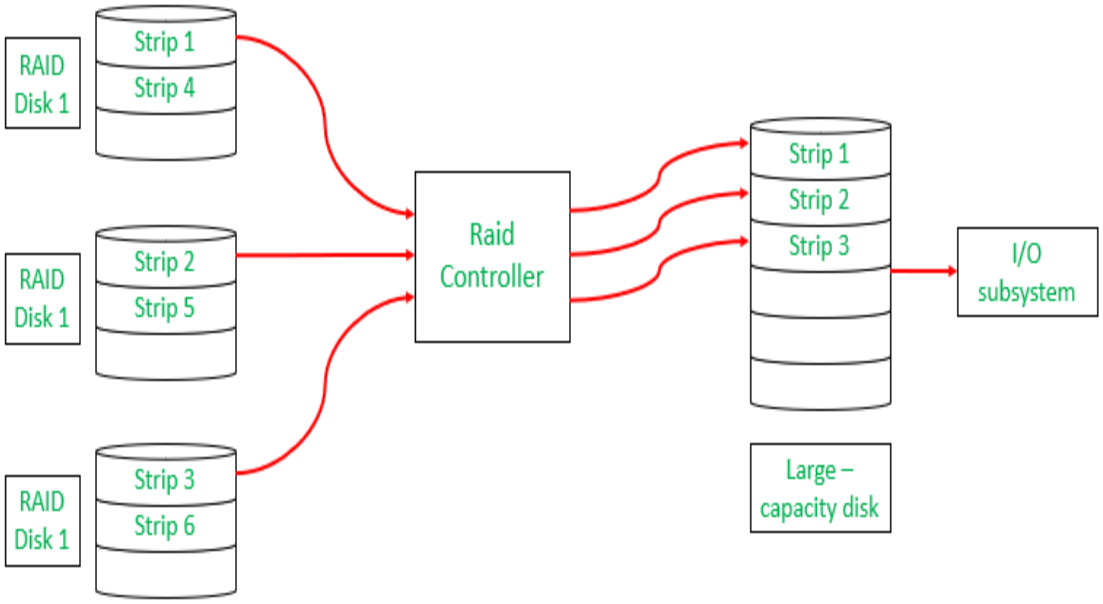
# Key Evaluation Points for a RAID System

- **Reliability:** How many disk faults can the system tolerate?
- **Availability:** What fraction of the total session time is a system in uptime mode, i.e. how available is the system for actual use?
- **Performance:** How good is the response time? How high is the throughput (rate of processing work)? Note that performance contains a lot of parameters and not just the two.
- **Capacity:** Given a set of N disks each with B blocks, how much useful capacity is available to the user?

RAID is very transparent to the underlying system. This means, to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes to the existing code.

# Different RAID Levels

1. RAID-0 (Stripping)
2. RAID-1 (Mirroring)
3. RAID-2 (Bit-Level Stripping with Dedicated Parity)
4. RAID-3 (Byte-Level Stripping with Dedicated Parity)
5. RAID-4 (Block-Level Stripping with Dedicated Parity)
6. RAID-5 (Block-Level Stripping with Distributed Parity)
7. RAID-6 (Block-Level Stripping with two Parity Bits)




Raid Controller

# What are File Access Methods in OS?

A file is a collection of bits/bytes or lines which is stored on secondary storage devices like a hard drive (magnetic disks).

File access methods in OS are nothing but techniques to read data from the system's memory. There are various ways in which we can access the files from the memory like:

- Sequential Access
- Direct/Relative Access, and
- Indexed Sequential Access.

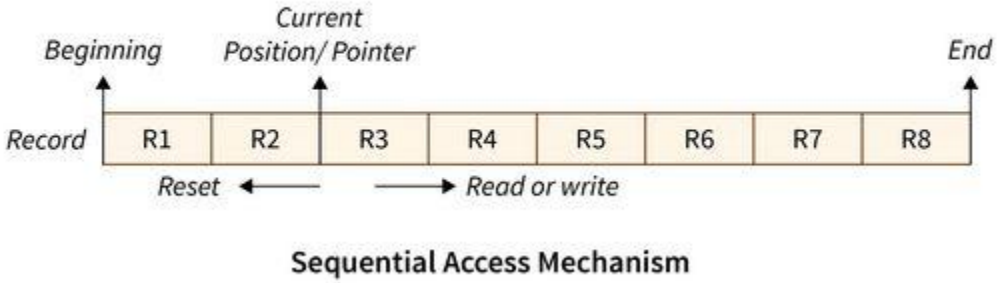
These methods by which the records in a file can be accessed are referred to as the file access mechanism. Each file access mechanism has its own set of benefits and drawbacks, which are discussed further in this article.

# Types of File Access Methods in the Operating System

## 1. Sequential Access

The operating system reads the file word by word in sequential access method of file accessing. A pointer is made, which first links to the file's base address. If the user wishes to read the first word of the file, the pointer gives it to them and raises its value to the next word. This procedure continues till the file is finished. It is the most basic way of file access. The data in the file is evaluated in the order that it appears in the file and that is why it is easy and simple to access a

file's data using sequential access mechanism. For example, editors and compilers frequently use this method to check the validity of the code.




Sequential Access Mechanism

**Advantages of Sequential Access:**

- The sequential access mechanism is very easy to implement.
- It uses lexicographic order to enable quick access to the next entry.

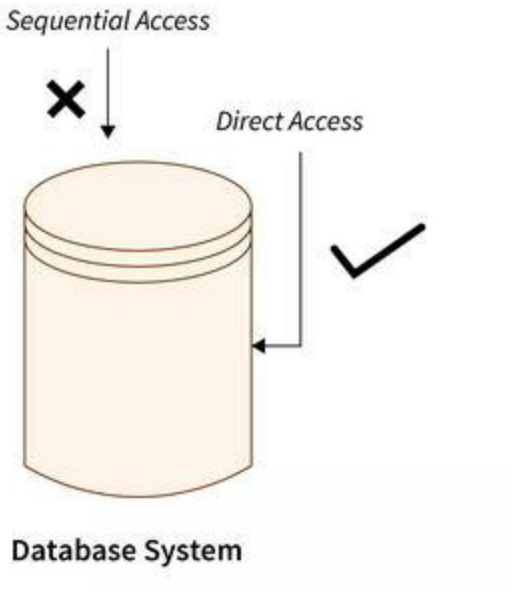
**Disadvantages of Sequential Access:**

- Sequential access will become slow if the next file record to be retrieved is not present next to the currently pointed record.
- Adding a new record may need relocating a significant number of records of the file.

## 2. Direct (or Relative) Access

A Direct/Relative file access mechanism is mostly required with the database systems. In the majority of the circumstances, we require filtered/specific data from the database, and in such circumstances, sequential access might be highly inefficient. Assume that each block of storage holds four records and that the record we want to access is stored in the tenth block. In such a situation, sequential access will not be used since it will have to traverse all of the blocks to get to the required record, while direct access will allow us to access the required record instantly.

The direct access mechanism requires the OS to perform some additional tasks but eventually leads to much faster retrieval of records as compared to the sequential access.




**Advantages of Direct/Relative Access:**

- The files can be retrieved right away with direct access mechanism, reducing the average access time of a file.
- There is no need to traverse all of the blocks that come before the required block to access the record.

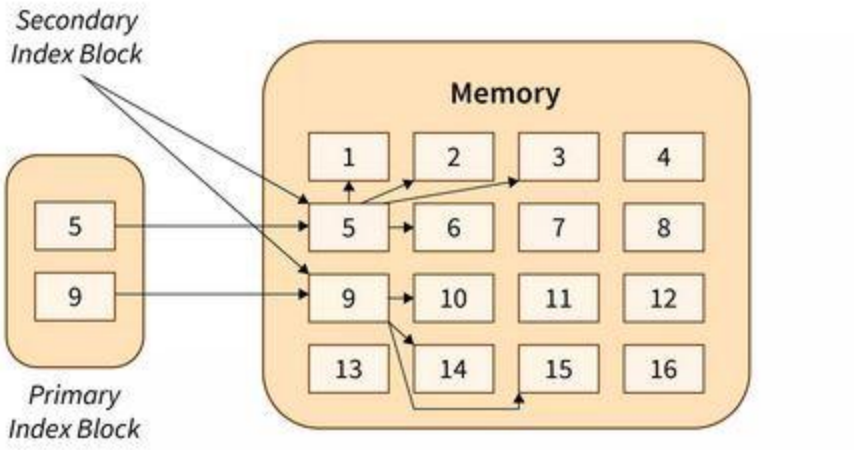
**Disadvantages of Direct/Relative Access:**

- The direct access mechanism is typically difficult to implement due to its complexity.
- Organizations can face security issues as a result of direct access as the users may access/modify the sensitive information. As a result, additional security processes must be put in place.

## 3. Indexed Sequential Access

It's the other approach to access a file that's constructed on top of the sequential access mechanism. This method is practically similar to the pointer to pointer concept in which we store an address of a pointer variable containing address of some other variable/record in another pointer variable. The indexes, similar to a book's index (pointers), contain a link to various blocks present in the memory. To locate a record in the file, we first search the indexes and then use the pointer to pointer concept to navigate to the required file.

Primary index blocks contain the links of the secondary inner blocks which contains links to the data in the memory.



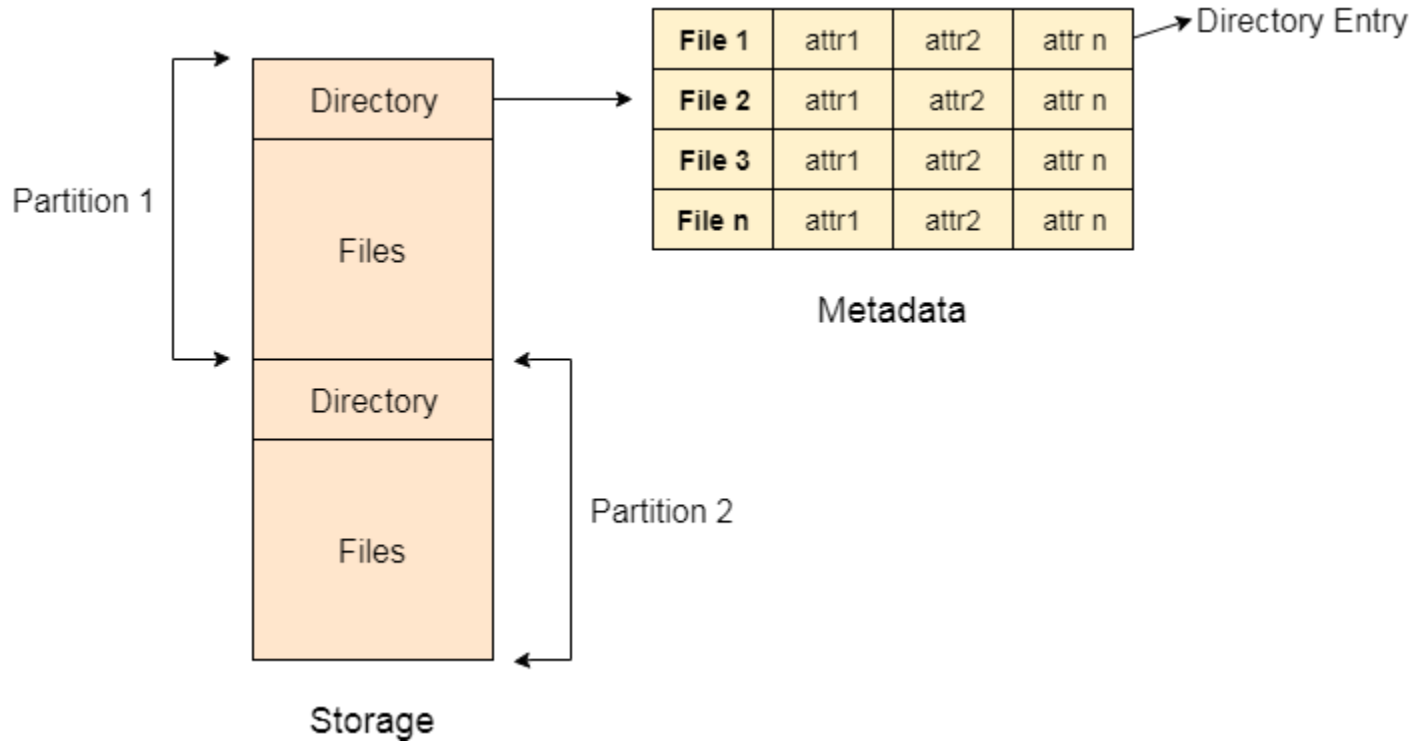

Indexed Sequential Access Of File

# What is a directory?

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.

To get the benefit of different file systems on the different operating systems, A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.

Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.




A directory can be viewed as a file which contains the Meta data of the bunch of files.

Every Directory supports a number of common operations on the file:

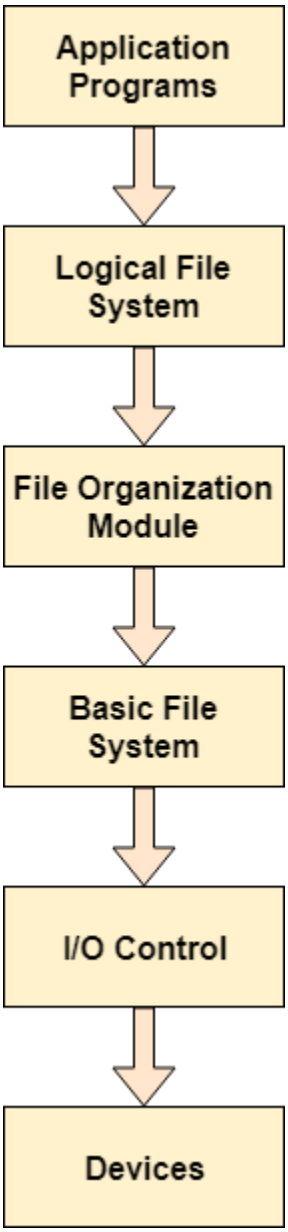
1. File Creation
2. Search for the file
3. File deletion
4. Renaming the file
5. Traversing Files
6. Listing of files

## File System Structure

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.




- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.

- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

# Definition of file system implementation

File system implementation is the process of designing, developing, and implementing the software components that manage the organization, allocation, and access to files on a storage device in an operating system.

# Importance of file system implementation

The file system implementation plays a critical role in ensuring the reliability, performance, and security of an operating system's file storage capabilities. Without an effective file system implementation, an operating system cannot efficiently manage the storage of data on a storage device, resulting in data loss, corruption, and inefficiency.

# File System Structure

- Disk layout and partitioning
- File system organization
- File allocation methods
- Directory structure

## Disk layout and partitioning

The disk layout and partitioning refers to how a physical disk is divided into logical partitions, which can be accessed by the operating system as separate entities. The disk is divided into one or more partitions which can be formatted with a file system. Disk partitioning involves creating partitions on the disk, while disk formatting involves creating a file system on the partition. The partitioning process is typically done when the disk is first installed, and the formatting process is typically done when a partition is created.

## File system organization

The file system organization refers to how files and directories are stored on the disk. A file system is responsible for managing files and directories, and providing a way for users and applications to access and modify them. Different file systems may organize files and directories in different ways, and may use different methods for storing and accessing them. For example, the FAT file system used by Windows organizes files in a simple directory hierarchy, while the HFS+ file system used by macOS organizes files in a more complex tree structure.

## File allocation methods

The file allocation method refers to how file data is stored on the disk. There are several different file allocation methods, including contiguous allocation, linked allocation, and indexed allocation. Contiguous allocation stores files in contiguous blocks on the disk, while linked allocation uses pointers to link blocks of data together. Indexed allocation uses an index to keep track of where each file block is stored on the disk.

## Directory structure

The directory structure refers to how directories are organized and managed on the disk. Directories are used to organize files and other directories into a hierarchy, which can be navigated by users and applications. Different file systems may use different directory structures, including single-level directories, two-level directories, and tree-structured directories. Directories can also have various attributes such as permissions and ownership, which can control who can access and modify files within them.

# Directory Implementation

There is the number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.

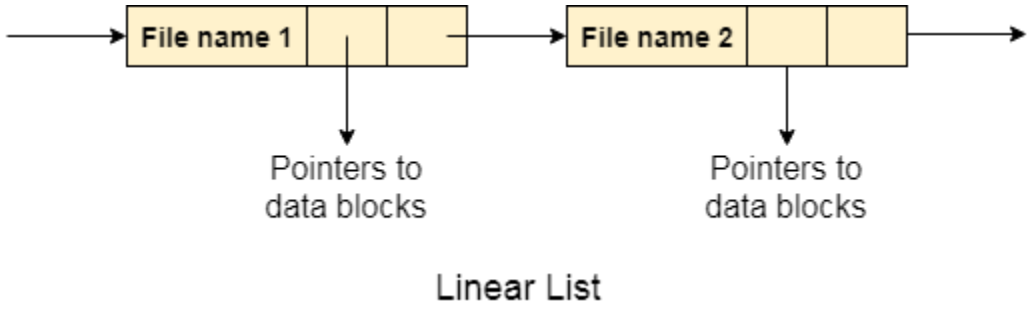
The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are used in these days.

## 1. Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

### Characteristics

1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.



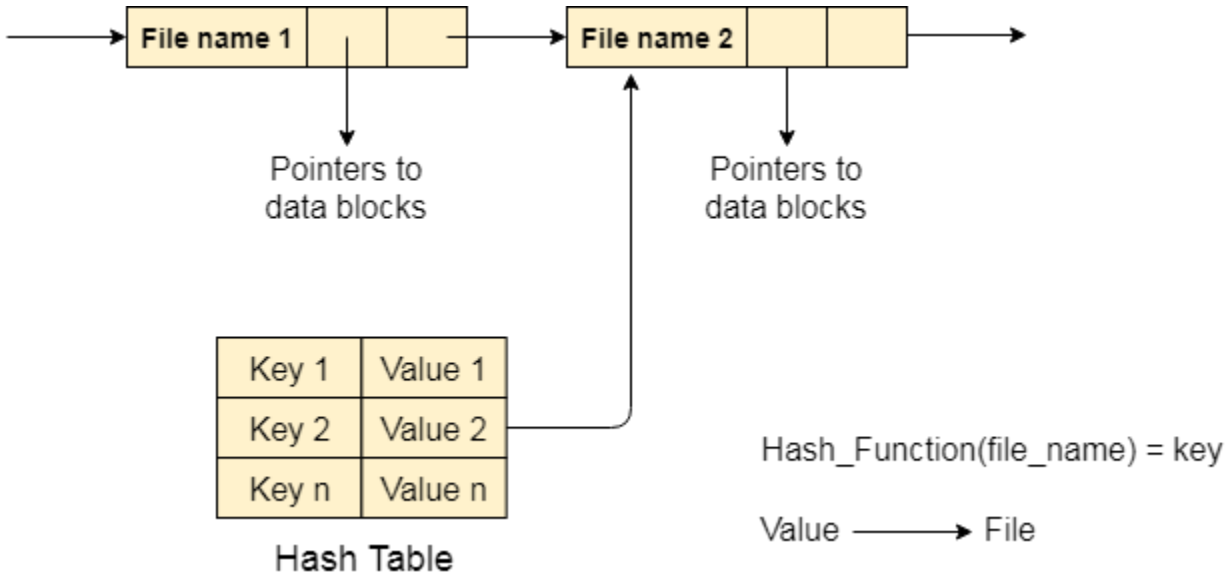

Linear List

## 2. Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.




Hash Table

## Operating System - I/O Hardware:

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

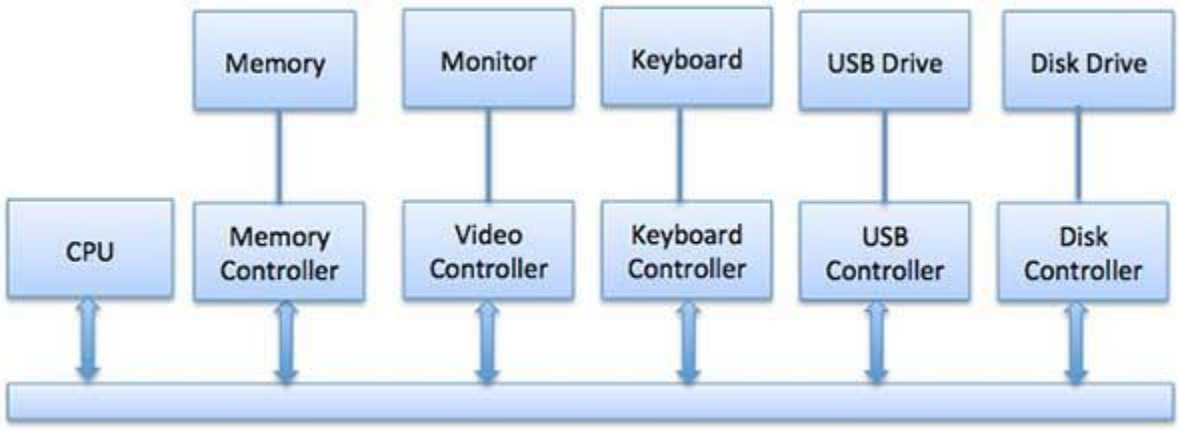
# Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.




# Synchronous vs asynchronous I/O

- **Synchronous I/O** − In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** − I/O proceeds concurrently with CPU execution

# Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

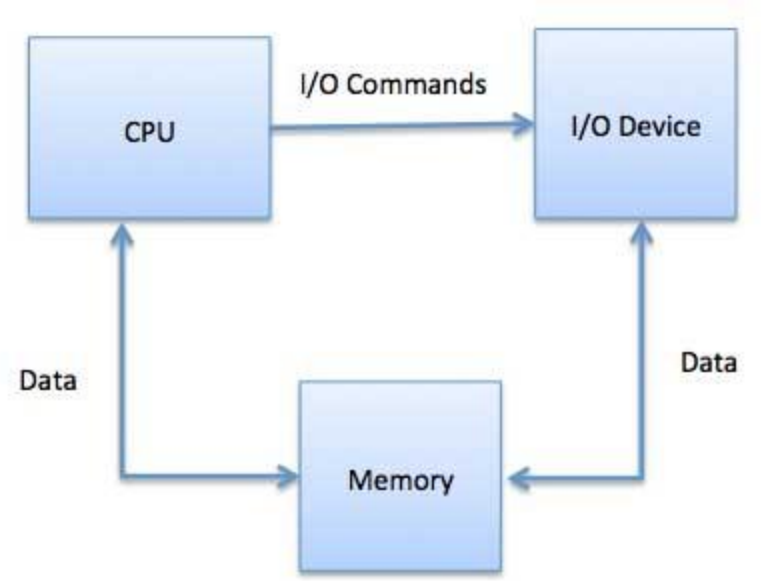
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

## Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

## Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.




While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

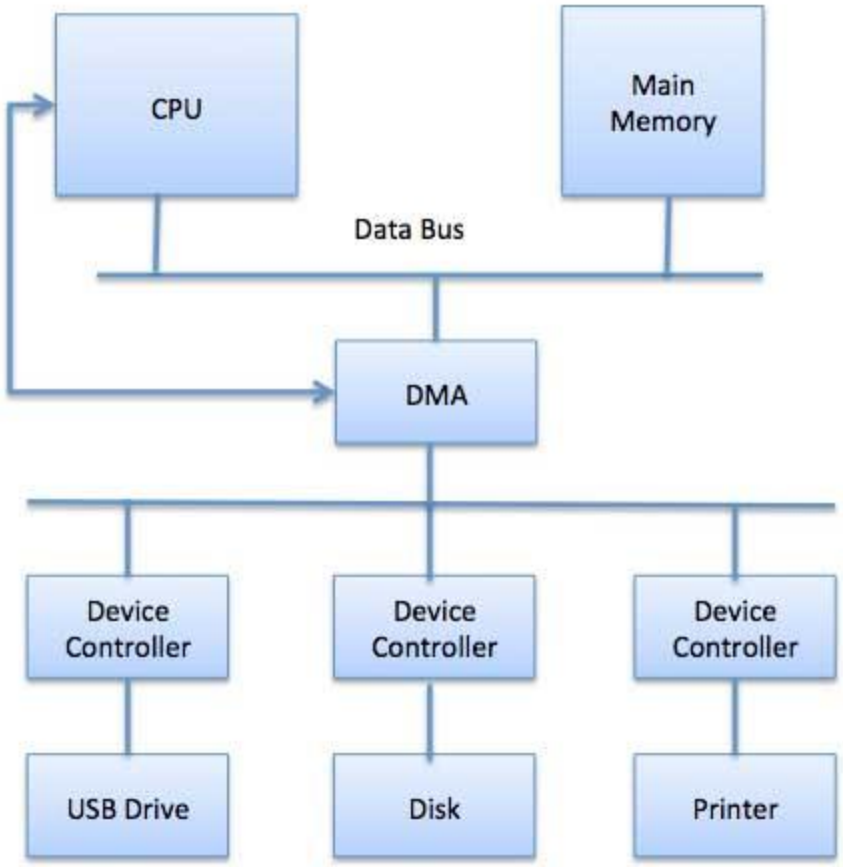
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

# Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.




The operating system uses the DMA hardware as follows −

| Step | Description |
|---|---|
| 1 | Device driver is instructed to transfer disk data to a buffer address X. |
| 2 | Device driver then instruct disk controller to transfer data to buffer. |
| 3 | Disk controller starts DMA transfer. |
| 4 | Disk controller sends each byte to DMA controller. |
| 5 | DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero. |
| 6 | When C becomes zero, DMA interrupts CPU to signal transfer completion. |

# Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

## Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

## Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU’s attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.