Unit 6

Device Management

The role of the Operating System in computer I/O is to manage and control I/O operation and I/O device. As input output device vary so widely in their function and speed, varied method are needed to control them. To encapsulate the details and oddities of different devices, the kernel of the OS is structure to use device-driver module. The device driver presents a uniform device access interface to the I/O subsystem much as a system call provides an interface between the application and operating system. Operating system takes helps from the device driver to handle all I/O devices.

Device management is the process of managing the implementation, operation and maintenance of physical or virtual device. Device controller works like an interface between a device and device driver. There is always a device controller and a device driver for each device to communicate with the operating system. Any device connected to the computer is connected by a plug and socket and the socket is connected to a device controller. Following activities are involved in device management:

- Keeps tracks of all devices. Program responsible for this task is known as I/O controller.
- Decides which process gets the device when and for how much time
- Allocated the device in the efficient way
- De-allocates devices.

Input/ Output Hardware:

Computer operates many kinds of device like storage device, transmission device and human interface device. A device communicates with a computer system by sending signals over a cable or even thorough the air. The device communicate with the machine via a connection point known as port or through the set of connected wire known as bus. A bus is a set of wire and a rigidly defined protocol that specifies a set of message that can be sent on wire. When a device A has cable connected to the device B and device to device C and device C plugs into a port on the computer then this arrangement is called a daisy chain and is usually operates on bus.

Buses are used widely in a computer architecture and vary in their signaling methods, speed, and throughput and connection method. A typical PC bus structure is shown in figure below.

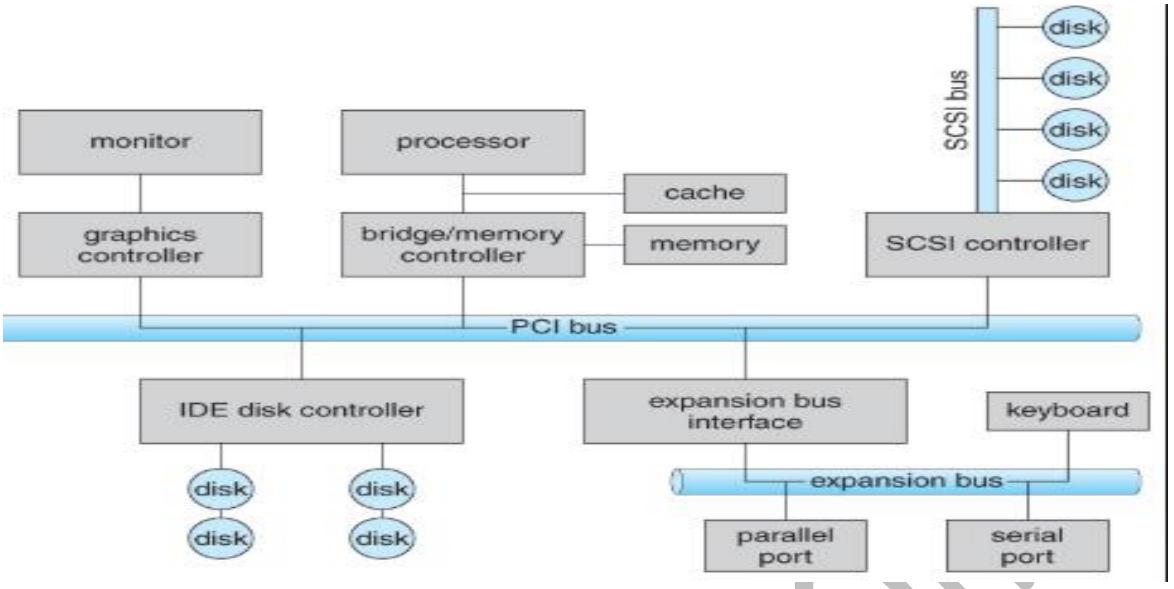


Figure: A typical bus structure

The above bus structure includes:

- A PCI bus that connects the processor memory subsystem to fast devices
- Expansion bus connects relatively slow devices like key board, USB ports.
- Four disk are connected together on a Small Computer System Interface (SCSI) bus plugged into a SCSI controller. SCSI controller consists of processor, micro code and some private memory to enable it to process the SCSI protocol message.
- A controller is a collection of electronics that can operate a port, a bus or device. A serial-port controller is a simple chip in the computer that controls the signal on the wires of a serial port.

An I/O port typically consists of four register:

- The data-in register which is read by the host to get input
- The data-out register which is written by the host to send output
- The status register that contain bits that can be read by the host. These bits indicates states such as whether the current command has completed or not, whether the byte is available to read from data-in register and whether a device error has occurred or not.
- The control register which can be written by the host to start a command or to change the mode of a device.

Methods for Detecting the Arrival of Any Type of Input:

A computer must have a way to detect the arrival of any type of input. The following are the ways to detect the arrival of input:

1. **Poling:**
   Poling is the process by where the computer or controlling device waits for an external device to check for its readiness or state. It is a simplest way to communicate with processor. Polling is the process of periodically checking the status of the device to see if it is time for the next I/O operation. The I/O device simply puts the information in a status register and the processor must come and get the information.
   Working procedure:
   The device controller indicates its state through the busy bit in the status register. The controller sets the busy bit (set bit to 1) to indicate that it is busy working and clears the busy bit (set bit to 0) when it is ready to accept next command. The hosts signals its command via the command-ready bit in the command register. The host set its command-bit when a command is available for the controller to execute.
   The two operation can be performed which are described below:
   - **I/O with Poling – Read:**
     Step 1: the host repeatedly reads the busy bit until that bit becomes clear.
     Step 2: After busy bit becomes clear, the host sets the read bit in the command register.
     Step 3: When the controller notice that the command-ready bit is set it in turns sets its busy bit to indicate that controller is busy working.
     Step 4: The controller reads the command register and sees the read operation.
     Step 5: Then controller allows a driver to copy the content of the controller's data register into the main memory user's process space.
     Step 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.

     Following figure shows the I/O poling- Read operation:

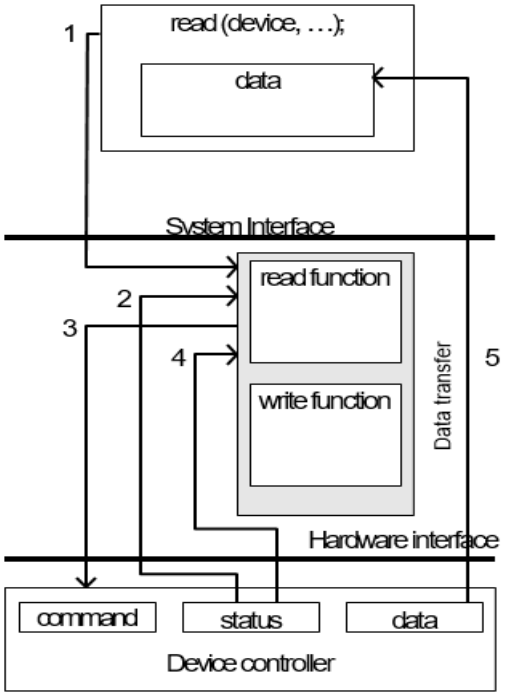


Figure: I/O poling- read operation

- **I/O with Polling – Write:**

  Step 1: the host repeatedly reads the busy bit until that bit becomes clear.

  Step 2: After busy bit becomes clear, the host sets the write bit in the command register.

  Step 3: When the controller notice that the command-ready bit is set it in turns sets its busy bit to indicate that controller is busy working.

  Step 4: The controller reads the command register and sees the write operation.

  Step 5: Then controller allows a driver to copy the content from user space memory to the controller's data register.

  Step 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.

  Following figure shows the I/O polling – write operation

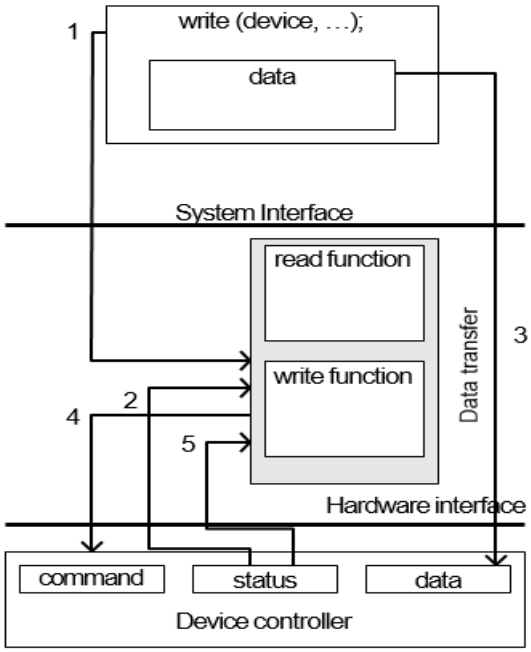


Figure: I/O poling- write operation

2. Interrupt driven I/O:

An alternative way for dealing with I/O is the interrupt driven method. Interrupt is a signal to the microprocessor from a device that requires attention. Interrupt is the hardware mechanism that enables a device to notify the CPU about I/O request. This process eliminates the needs for device driver to constantly poll the system register. Instead of polling, the device controller automatically notifies the device driver when the operation has completed.

Working mechanism:

Step1: The CPU hardware has a wire called the interrupt-request line that the CPU senses after executing every instruction.

Step 2: The device controller raises an interrupt by asserting a signal on the interrupt request line.

Step 3: When the CPU detect that the controller has asserted a signal on the interrupt request line then the CPU saves its current state (saves what it is currently doing) and jumps to the interrupt's handler routine at a fixed address in memory.

Step 4: The interrupt handler determines the cause of interrupt and, perform the necessary data processing.

Step 5: After completion of data processing, interrupt handler performs the state restore and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.

Step 6: After state restore operation, CPU returns to previous state (state that it was executing before interrupt had occurred) and resumes processing of interrupted task.

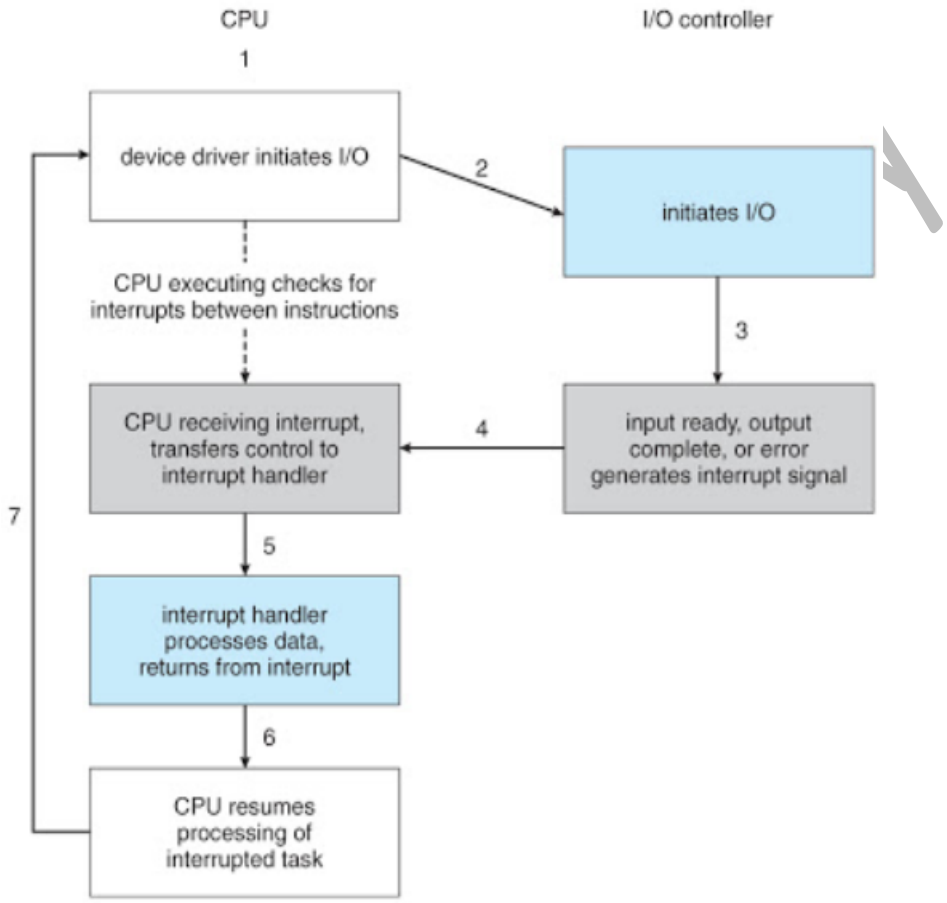


Figure: Interrupt Driven I/O cycle

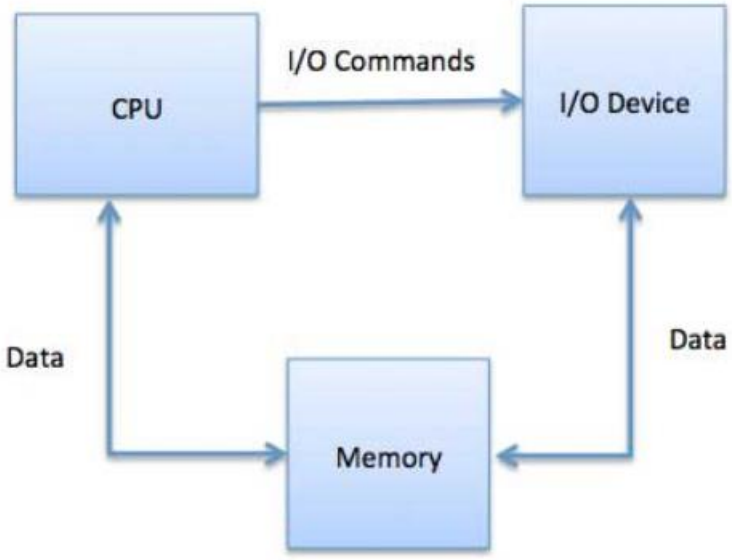
Communication to I/O Device:

The CPU must have a way to pass information to and from an I/O device. The controller has one or more register for data and control signal. Following are the approaches available to communicate with CPU and devices;

1) **Special Instruction I/O:**
   It specifies the transfer of byte or word to an I/O port address and uses the CPU instruction that are specifically made for controlling I/O devices. The I/O instruction triggers bus line to select the proper device and to move bits into or out of a device register.

2) **Memory mapped I/O:**

In this case, the device control register are mapped into the address space of the processor. The CPU executes the I/O request using the standard data transfer instruction to read and write the device control registers at their mapped locations in physical memory. The I/O device is also connected directly to certain main memory location so that I/O device can transfer block of data to/ from memory without going through CPU.



In this scheme, 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 and interrupts CPU when finished.

Advantages:

- Every instruction which can access memory can be used to manipulate I/O device.
- This method is used for most high speed I/O devices like disks, communication interfaces.

3) **Direct Memory Access (DMA):**

For a device that does large transfer such as a disk drive it seems wasteful to use general processor to watch status bit at a time. The operating system have to spend most of its time handling these interrupts. So, computer uses direct memory access (DMA) hardware to reduce this overhead.

DMA means CPU grants I/O authority to read from or write to memory without its involvement. That is DMA controller controls the exchange of data between main memory and the I/O device and CPU is only involved at the beginning and end of the transfer an interrupted only after entire block has been transferred.

Working Process:

Step1: To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains pointer to the source of the transfer, a pointer to the destination of the transfer and count of the number of bytes to be transferred.

Step 2: The CPU writes the address of this command block to the DMA controller then goes on with another work.

Step 3: Then the handshaking (connection) between the DMA controller and the device controller is initiated via a pair of wire called DMA-request and DMA-acknowledgement.

Step 4: The device controller places a signal on the DMA-request wire when a word of data is available to transfer

Step 5: This signal causes the DMA controller to seize the memory bus, places the desired address on the memory address wires and places a signal on the DMA acknowledgement wire without the involvement of CPU.

Step 6: When the device controller receives the DMA-acknowledgement signal, it transfer the word of data to memory and removes the DMA request signal.

Step 7: When the entire transfer is finished the DMA controller interrupts the CPU.

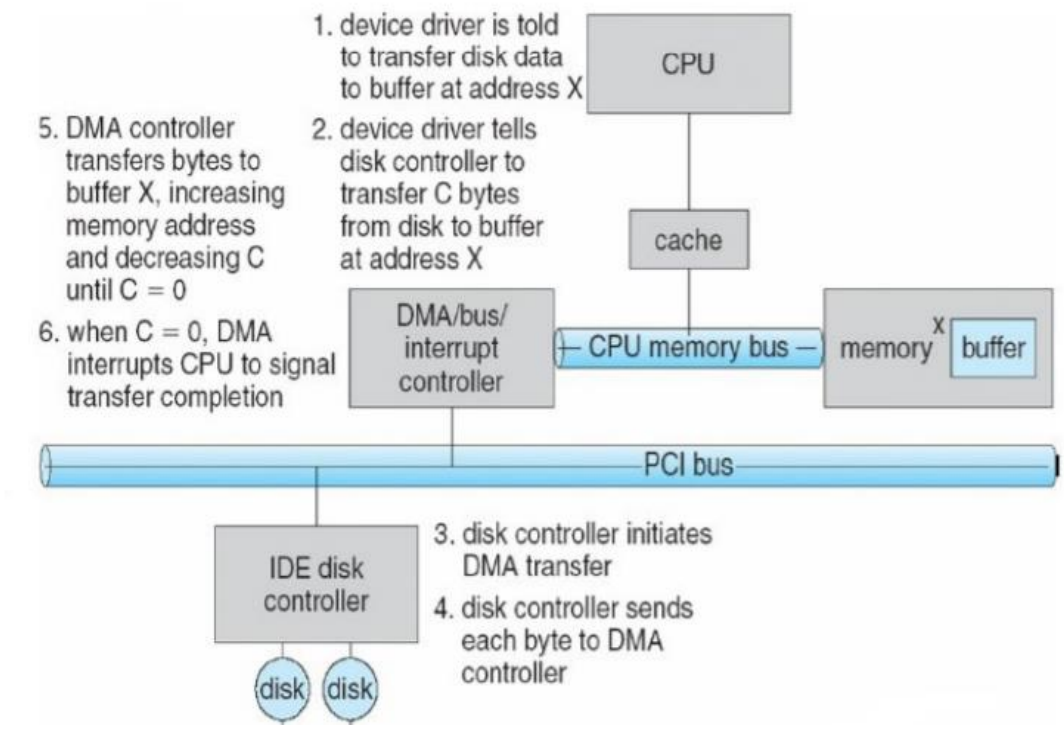


Figure: Steps in a DMA Transfer

Application I/O Interface:

The method that is used to transfer information between internal storage and external I/O devices is known as **I/O interface**. It enables the I/O device to be treated as standard, uniform way. The CPU is interfaced using special communication link by the peripheral connected to any computer system and such link are used to solve difference between CPU and peripheral. The approach here involves abstraction, encapsulation and software layering.

The detail difference in I/O device can be abstract (hide) by identifying a few general kinds. Each general kind of device is access through a standardized set of function known as interface. The difference in I/O device are **encapsulated** in kernel module called device driver that internally are custom-tailored (specifically maintained) according to specific device. The purpose of the device driver layer is to hide the difference among device controller from the I/O subsystem of the kernel. Making the I/O system independent of hardware simplifies the job of the OS developer and hardware manufacture. They either design new device to be compatible with existing controller interface or they write the device driver to interface new hardware.

Following figure illustrate how the I/O related portions of the kernel are structured in software layer:

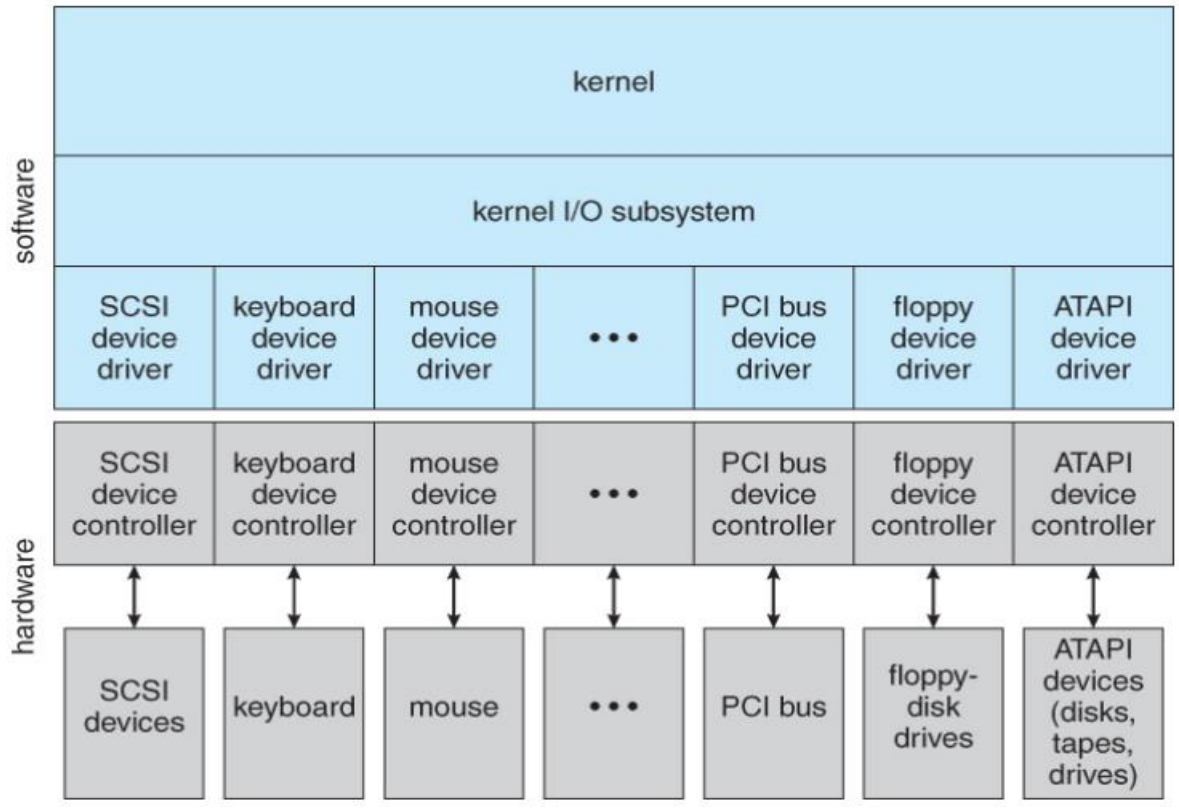


Figure: kernel I/O structure

Following are some of the ways of data transfer from the different devices:

- **Character stream or block:**
  A character stream device transfer bytes one by one and basic system calls in character interface enable an application to get () or put () one character at a time. A keyboard is an example of device that is accessed through a character stream interface. On top of this interface, libraries can be built that offer line at a time access with buffering and editing service such as if user type a backspace, the character is removed from the character input stream. Some example of character stream device are key board, mouse, modem etc.
  Block device transfer a block of byte as a unit. The block device interface captures all the aspects necessary for accessing disk drives and other block oriented device. The device is expected to understand command such as read () and write () and if the device is random access, it is also expected to have a seek () command to specify which block to transfer next.
- **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 location.
- **Synchronous or Asynchronous**
  Synchronous device performs data transfer with predictable response times in coordination with other aspects of the system. An asynchronous device exhibits irregular or unpredictable response times no coordinated with other computer event.
- **Sharable or Dedicated**:
  A sharable device can be used concurrently by several process or threads but dedicated device cannot.
- **Speed of Operation:**
  Device speed range from a few byte per second to a few gigabyte per second.
- **Read-write, read only or write only**:
  Some devices performs both input and output but other support one data transfer direction.

Input/ Output Organization:

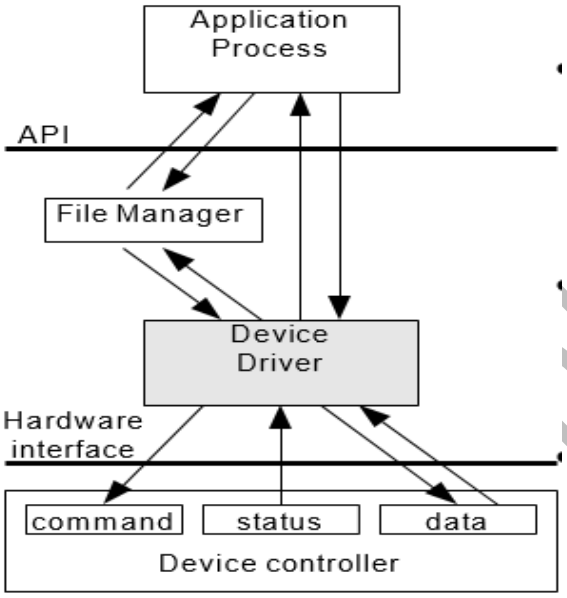


Figure: Organization of Input/output system

Above figure shows how the control flows from device to device driver when input arrives and when output is to be shown. First an application process uses an appropriate device to exchange data with device management (driver). Device driver implement a communication API (interface) that abstract the functionality of the device and provides specific operation to implement function defined by the API. Communication API should be similar across different device driver reducing the amount of information and application programmer needs to know to use the device.

Device driver communicates with the device controller to check what kind of command has arrived in controller (read, write etc.), to check whether the status of the controller is busy or idle and to transfer data to and from device controller data register as required for correct device operation.

Device drivers:

Device driver is a computer program that operates or controls a particular types of device attached to a computer. Drivers are hardware dependent and operating system specific. It enables one or more hardware device to communicate with operating system. If there is no any drivers then the computer would not be able to send and receive data correctly from or to hardware device like keyboard,

printer etc. Device driver act as a translator between the hardware device and the program or operating system that use it. Device manager is a features of OS that detects and lists hardware devices and their status information.

Designing the device manager involves invoking controller specific I/O operation while satisfying:

- An API that implements the I/O function available to the device still compliant with interfaces implemented by other drivers. This is called device driver interface
- Achieve correct coordination among the application processes, drivers and device controllers.
- Optimize the overall machine performance with correct driver strategies.

Each operating system has two major interfaces to the device manager:

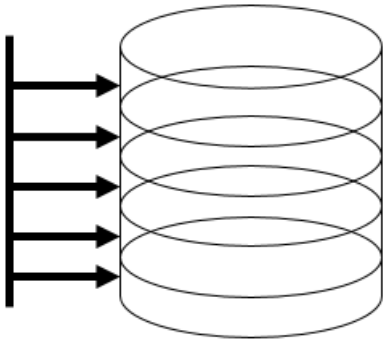
- The driver API
- The interface between a driver and the operating system kernel.

Some Device Management Scenarios:

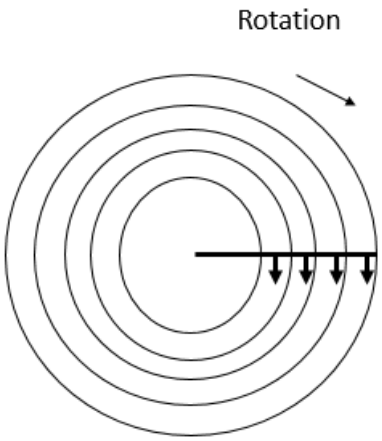
- **Sequential Access Storage Device:**
  It is a class of data storage device that read stored data in a sequence. Sequential access devices are usually a form of magnetic storage or optical storage. Sequential access is the reading of any file by searching from the storage device from the beginning. Length of the record can be of any length which is usually determined by the application program. In this method, record are identified by its position on the tape. For example: magnetic tape: to access a single record tape is mounted and fast forwarded from its beginning to the desired location.
- **Direct Access Storage Device (Random Access Storage Device):**
  Any device that can directly read and write to a specific place on a disk. It stores the data in discrete location with address. Example of this type of device are hard drive, optical drives and most magnetic storage device. This storage device allows a host computer to access the data directly from wherever it is stored within the storage device. This allows a computer to directly point to that location to get the data.

Two categories of DASD are:

- **DASD With Fixed Read/ Write Head:**
  This types of device have magnetically recordable drums which resembles like a giant coffee can and covered with magnetic film and formatted so the tracks run around it. Data is recorded serially on each track by the read/write head positioned over it. Fixed head drums were very fast but also very expensive and they did not hold as much data as other DASDs.



  Each fixed head disk looks like a phonograph album and covered with magnetic film that has been formatted usually on both sides into concentric circles. Each circle is known as track. Data is recorded serially on each track by the fixed read/write head positioned over it.



- **DASD With Movable Read/Write Heads:**
  This type of device have one read/write head that floats over each surface of disk. For example: hard drive. Disk can be a single platter or part of a disk pack (stack of disk platter) and each platter has two recording surfaces. Such surface is formatted with a specific

number concentric track where the data is recorded. The number of track varies from manufacturer to manufacturer.

Disk arm moves two read/write heads between each pair of surfaces i.e. one for the surface above it and one for the surface below. The arm moves all of the heads in unison and all are positioned on the same track but on their respective surfaces creating a virtual cylinder.

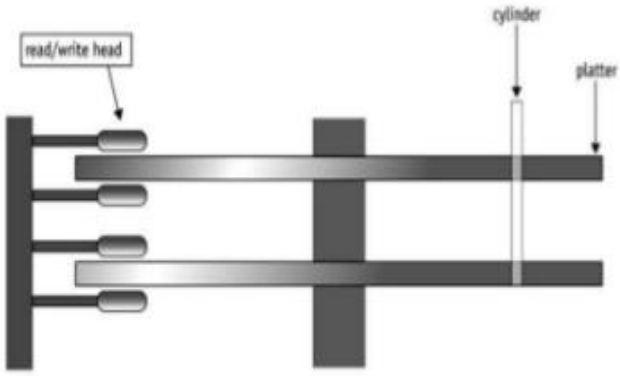


Figure: disk pack:

To access any given record, the system needs three things: its cylinder number so the arm can move the read/write heads to it, its surface number so the proper read/write head is activated, its sector number so the read/read head knows the instant when it should begin reading and writing.

- **Serial Communication:**

  It is a method of communicating data between devices in which the individual data bits being sent sequentially. Serial communication may be asynchronous where the data character includes start and stop bits to delimit the data or synchronous where such additional bits are omitted and delimiting of data depends purely on timing.

  For synchronous, transmitter and receiver needs to be synchronized and need extra hardware for clock synchronization having faster data transfer rate whereas in asynchronous, transmitter and receiver use different clock and no clock synchronization is required.

Secondary Storage Structure

Overview

Magnetic Disk:

Magnetic disk provide the bulk of secondary storage for modern computer system. This disk is conceptually simple as shown in figure below. Each disk platter has a flat circular shape like a Cd. The two surface of platter are covered with magnetic material and information are stored by recording it magnetically on the platter. The read write head lies just above each surface of platter and are attached to a disk arm that moves all the head as unit. The surface of a platter is logically divided into circular tracks which are subdivided into sector. The set of track that are at one arm position makes up a cylinder. There may be thousands of cylinder and each track may contains hundreds of sector.

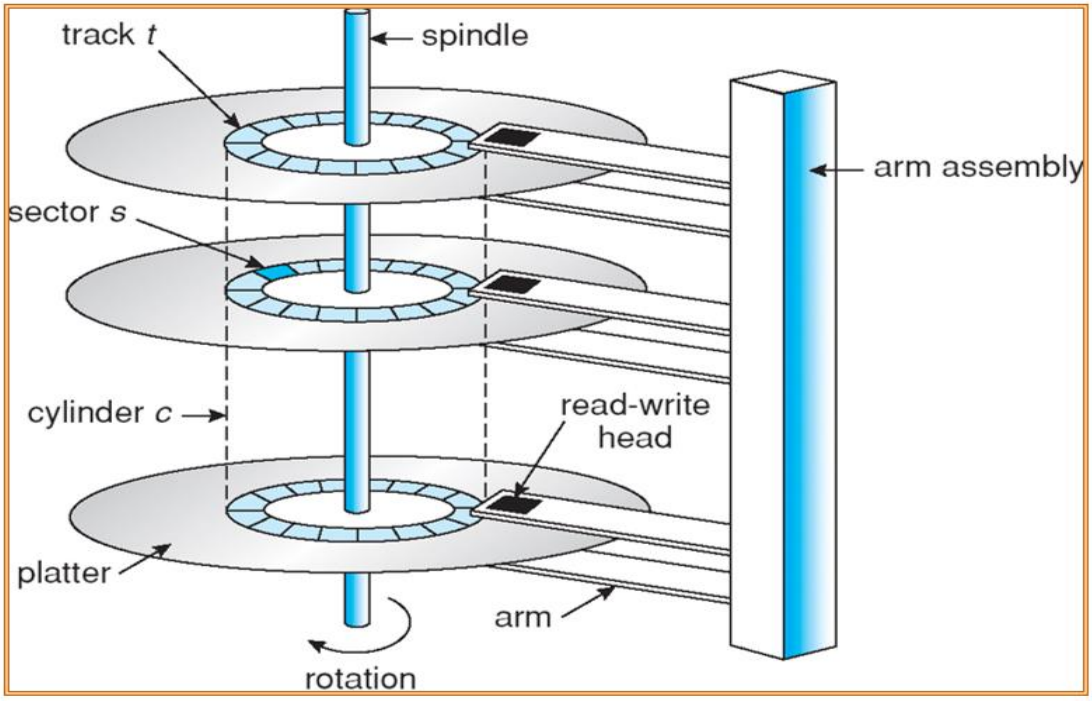


Figure: Moving head disk mechanism

When a disk is in use, a drive motor spin it at high speed like 60 to 250 times per second and are measured in term of rotation per minute (RPM). Disk speed has two part: the transfer rate is a rate at which data flow between the drive and computer. The positioning time or random-access time consist of two parts: the time necessary to move the disk arm to the desired cylinder called the seek time and the time necessary for the desired sector to rotate to the disk head called the rotational latency.

The disk platter are coated with a thin protective layer but the head will sometimes damage the magnetic surface. This accident is called a head crash. A disk drive is attach to computer by a set of wires called I/O bus. The data transfer on the bus are carried out by special processor called controller. The host controller is the controller at the computer end in which command is places for I/O operation. The host controller then sends message to disk controller which is built into each disk and operate to carry out the received command.

Disk Structure:

Modern magnetic disk drives are addressed as large one dimensional array of logical block where the logical block is the smallest unit of transfer. The one dimensional array of the logical block is mapped into a sector of the disk sequentially. Sector 0 is the first sector of the first track on the outer most cylinder. The mapping proceeds in order throughout the rest of the track in that cylinder and then throughout the rest of the cylinder from the outermost to innermost.

To map logical block into disk address some media uses **constant linear velocity** (CLV) in which density of bit per track is uniform. The farther the track is from center of disk the greater its length so the more sector can be hold. From the outer zone to inner zone the number of sector per track decreases. The drive increases its rotation speed from as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head. Alternately, the disk rotation speed can stay constant. In this case density of bits decreases from inner track to outer track to keep the data rate constant. This method is used in hard drive and is known as **constant angular velocity (CAV).**

Disk Scheduling:

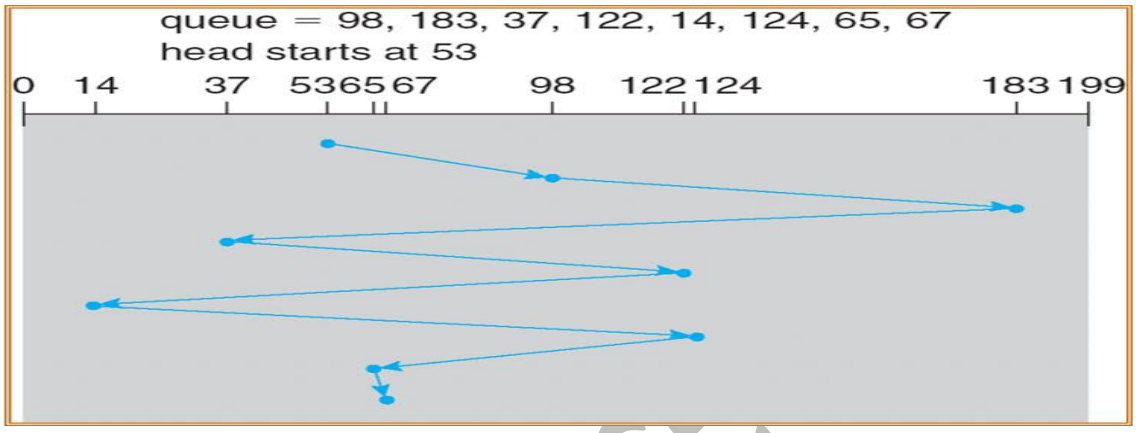
One of the responsibility of the operating system is to use the hardware efficiently. For the disk drive, meeting this responsibility means having faster access time and large disk bandwidth. The access time has three major component:

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

Both the access time and bandwidth can be improve by managing the order in which disk I/O requests are serviced.

There are number of algorithm exist to schedule the disk I/O requests:

1) **First Come First Serve (FCFS) Scheduling:**
   This algorithm is fair but generally does not provide fast service. This scheme selects the I/O block that have request first. For example:



   As head starts from 53, it will first move from 53 to 98 then to 183, 37, 122, 14, and 124, 65 and last to 67.

2) **Shortest Seek Time First (SSTF):**
   It selects the request with the least seek time from the current head position i.e. chooses the request closest to current head position. SSTF is essentially a form of shortest job first (SJF) scheduling so may cause starvation of some request.

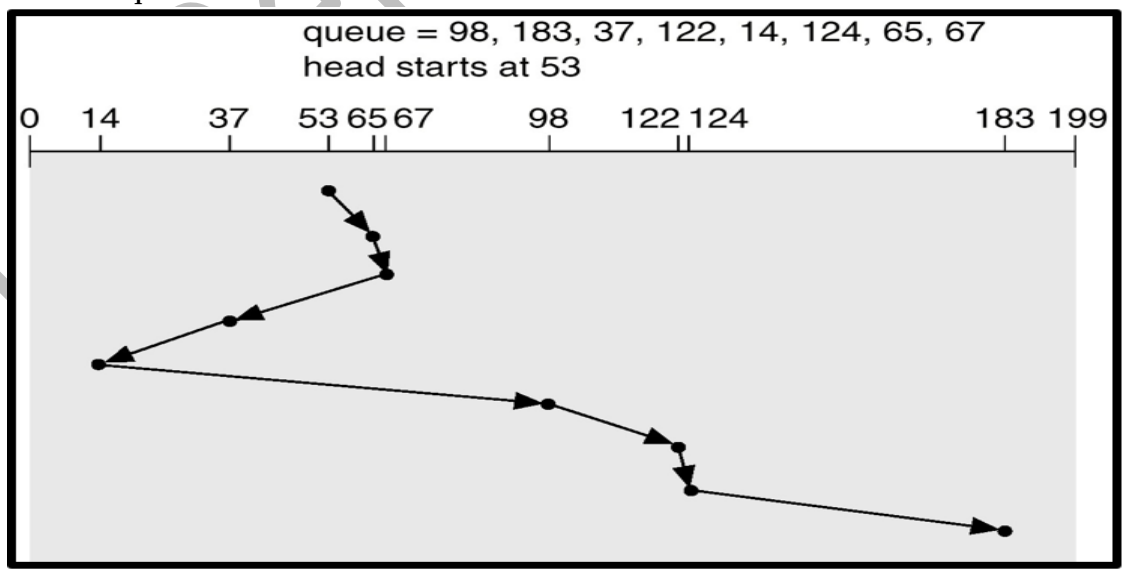
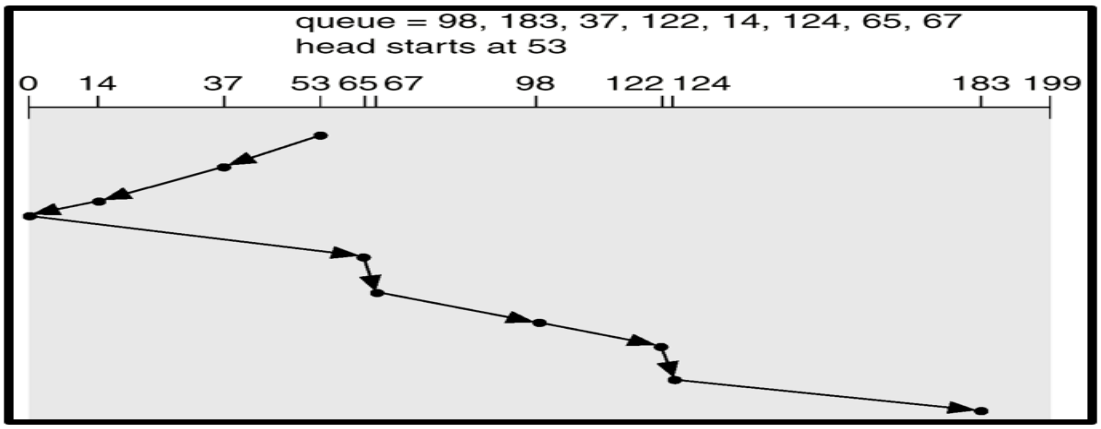


Figure: SSTF scheduling:

From the disk head 53, 65 is closed to 53 so it moves to 65. From 65, 67 is closer than other block so it moves to 67, from 67, 37 is closer than 98 because (67-37 = 30) and 98-67 = 31. Similarly process continues.

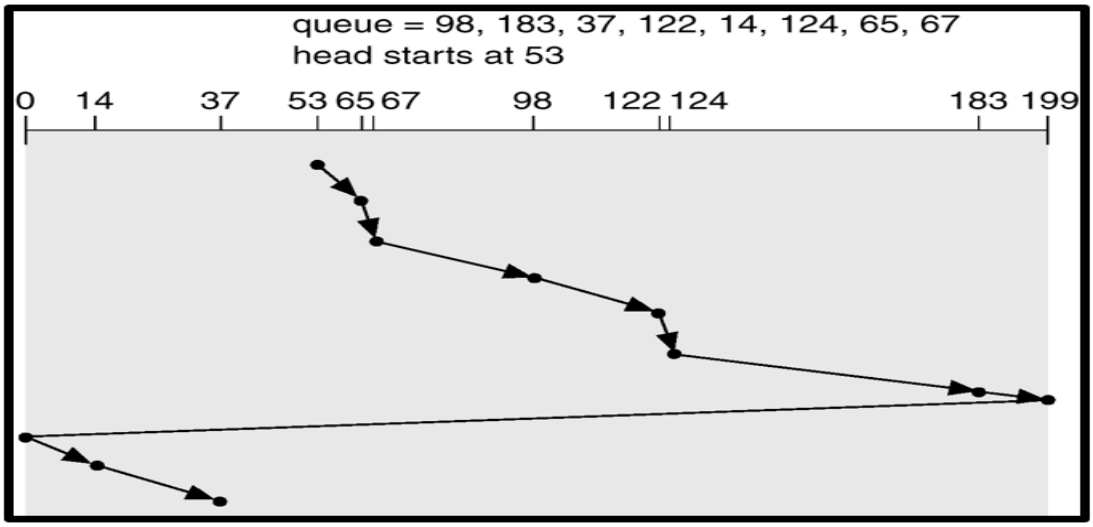
3) **SCAN:**

In this algorithm, the disk arm starts at one end of the disk and moves toward other end of the disk servicing request as it reaches each cylinder until it get to the other end of the disk. The head continuously scans back and forth across the disk. Sometimes called elevator algorithm.



4) **C-SCAN:**

Circular SCAN scheduling is a variant of SCAN in which head moves from one end of the disk to the other servicing request along a way. When the head reaches the other end however it immediately returns to the beginning of the disk without servicing any requests on the return trip. This algorithm treats the cylinder as a circular list that wraps around from the final cylinder to the first one.

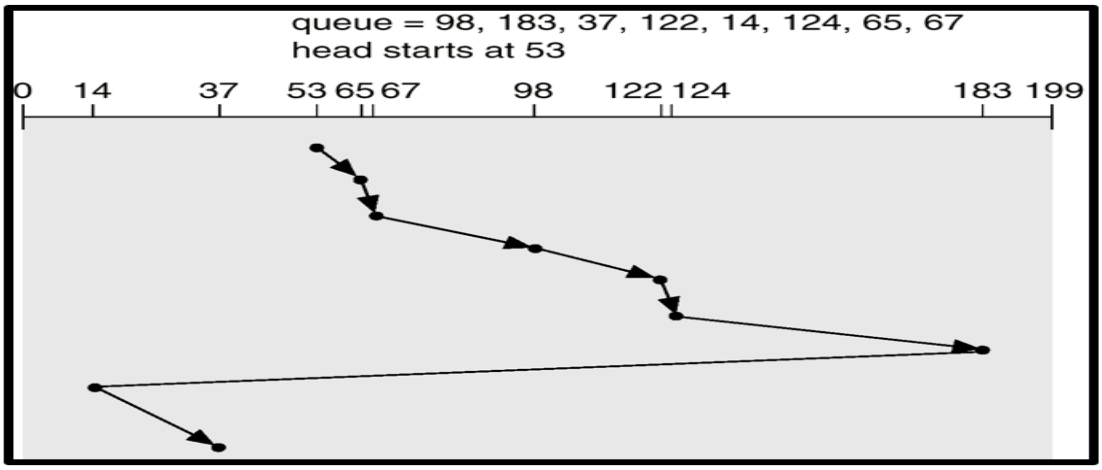


5) **LOOK:**

In this algorithm, the disk arm starts at one end of the disk and goes as far as the final request and reverse the direction from there i.e. will not move to the end of the disk.

6) **CLOOK:**

It is similar to C-SCAN, in which the disk arm in spite of going to the end goes to the last request to be services and then from there goes to the other end's last request.



Disk Management:

The operating system is responsible for several other aspects of disk management such as initializing a disk, booting from disk, bad block recovery etc.

1. **Disk Formatting:**

   A new magnetic disk is a blank state: it is just a platter of magnetic recording material. Before a disk can store data it must be divided into sector such that disk controller can read and write. This process is called **low level formatting or physical formatting.** This low level formatting fills the disk with a special data structure for each sector which consist of header, a data area and a trailer. Header and trailer contains information used by the disk controller such as a sector number and **Error Correcting Code (ECC).**

   When the controller writes a sector of data during normal I/O the ECC is updated with a value calculated form all the bytes in the data area. When the sector is read the ECC is recalculated and compared with the store value. If the calculated value does not match with stored value then this indicates the data area of sector have been corrupted and the disk sector may be bad.

   Before operating system can use a disk to hold file, OS needs to record its own data structure on the disk. It does in two step:

- The first step is to partition the disk into one or more group of cylinders. One partition can hold a copy of the OS's executable file while another can hold user's file.
- The second step is logical formatting or creation of file system in which OS stores the initial file system data structure onto the disk. These data structure may include maps of free and allocated space and an initial empty directory.

2. **Boot Block:**

When the computer is turned on, an initial program is run which initializes all aspect of the system from CPU program register to device controller and the content of main memory and then starts the operating system. Such initial program is known as **bootstrap program**. To do this job, bootstrap program finds the operating system kernel into memory and jumps to an initial address to begin the operating system execution.

For most computer bootstrap program is stored in ROM but a problem is that changing the bootstrap code requires changing the RAM chip. So, most system store a tiny bootstrap loader program in the boot ROM whose only job is to bring a full boot strap program from disk. The full bootstrap program is store in the boot block at a fixed location on the disk. A disk that has a boot partition is called **boot block or system disk.** The code in the boot ROM instructs the disk controller to read the boot block into memory and then starts executing code.

Example of boot process in Windows:

Windows allowed hard disk to be divided into partition and one partition identified as the boot partition contains the operating system and device driver. The windows system places its boot code in the first sector on the hard disk which is known as master boot record (MBR). Booting begins by running code that is resident in the system's ROM memory. This code direct the system to read the boot code from the MBR. MBR also contains the table listing the partition for hard disk and a flag indicating which partition the system is to be booted from.

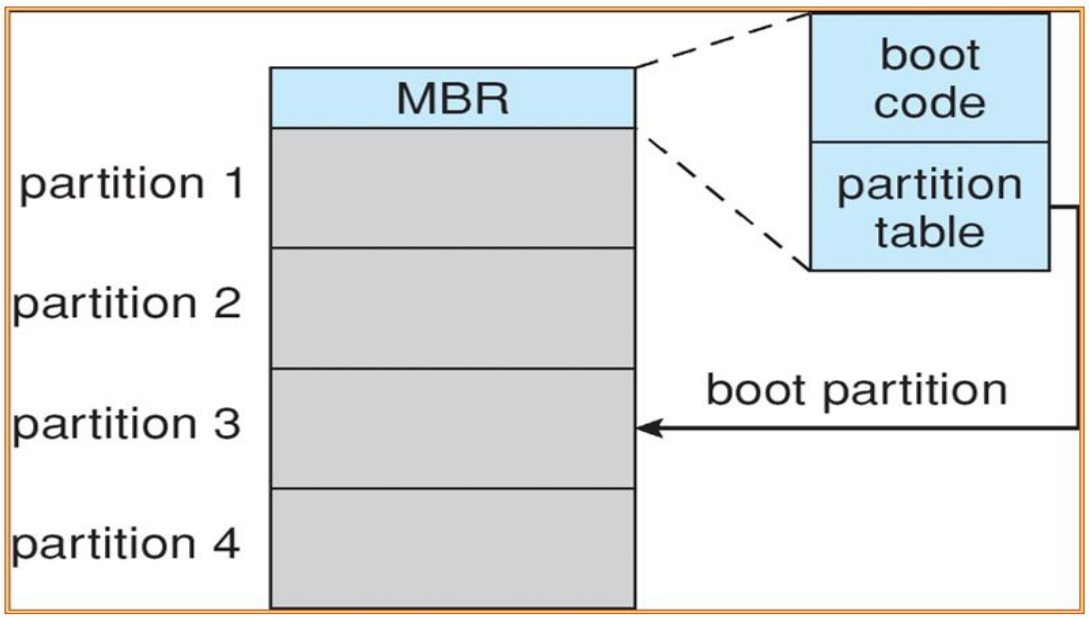


Figure: Booting from disk in windows

Once the system identifies the boot partition it reads the first sector from that partition which is called boot sector and continues with the remainder of the boot process which includes loading the various subsystem and system service.

3. **Bad Blocks:**

As disk have moving parts and small tolerance they are prone to failure. More frequently one or more sector becomes defective. Most disk even comes from the factory with the bad blocks. Depending on the disk and controller in use these blocks are handled in a variety of ways:

- On **simple disk** with IDE controllers, bad blocks are handled manually. One strategy is to scan a disk to find bad blocks while the disk is being formatted. Any bad block that are discovered are flagged as unusable so that the file does not allocate them.
- In **most sophisticated** disk, the controller maintains a list of bad blocks on the disk. The list is initialized during the low level formatting at the factory and is updated over the life of the disk. The controller can be told to replace each bad block logically with one of the spare sector. This scheme is known as sector sparing or forwarding.