Unit-6

Input/Output Device Management

Principles of I/O Hardware

Different people look at I/O hardware in different ways. Electrical engineers look at it in terms of chips, wires, power supplies, motors, and all the other physical components that make up the hardware. Programmers look at the interface presented to the software the commands the hardware accepts, the functions it carries out, and the errors that can be reported back. Nevertheless, the programming of many I/O devices is often intimately connected with their internal operation.

I/O devices

I/O devices can be roughly divided into two categories: block devices and character devices. A block device is one that stores information in fixed-size blocks, each one with its own address. Common block sizes range from 512 bytes to 32,768 bytes. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. Disks are the most common block devices.

If you look closely, the boundary between devices that are block addressable and those that are not is not well defined. Everyone agrees that a disk is a block addressable device because no matter where the arm currently is, it is always possible to seek to another cylinder and then wait for the required block to rotate under the head. Now consider a tape drive used for making disk backups. Tapes contain a sequence of blocks. If the tape drive is given a command to read block N, it can always rewind the tape and go forward until it comes to block N. This operation is analogous to a disk doing a seek, except that it takes much longer. Also, it may or may not be possible to rewrite one block in the middle of a tape. Even if it were possible to use tapes as random access block devices, that is stretching the point somewhat: they are not normally used that way.

The other type of I/O device is the character device. A character device delivers or accepts a stream of characters, without regard to any block structure. It is not addressable and does not have any seek operation. Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices that are not disk-like can be seen as character devices.

This classification scheme is not perfect. Some devices just do not fit in. Clocks, for example, are not block addressable. Nor do they generate or accept character streams. All they do is cause interrupts at well-defined intervals. Still, the model of block and character devices is general enough that it can be used as a basis for making some of the operating system software dealing with I/O device independent. The file system, for example, deals only with abstract block devices and leaves the device-dependent part to lower-level software called device drivers.

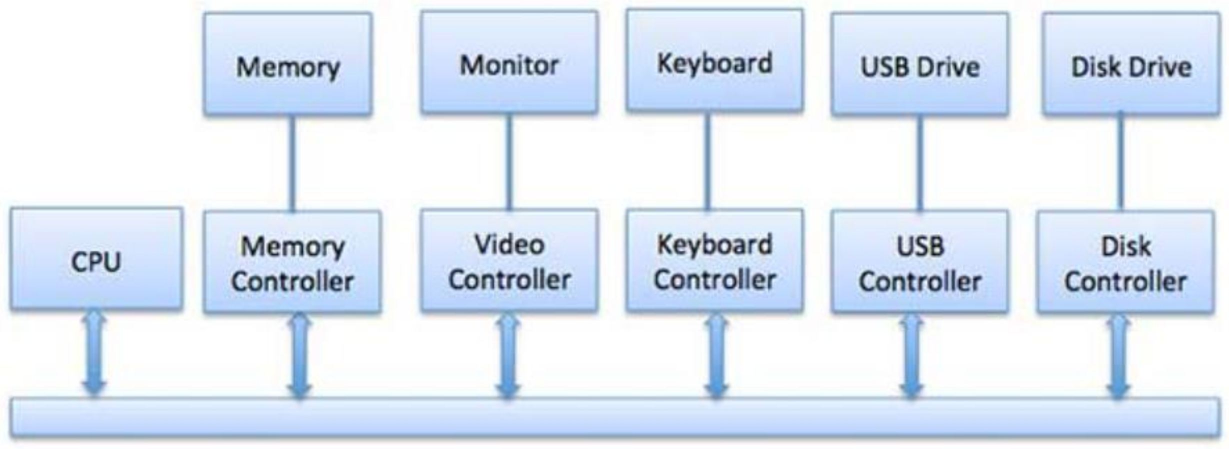
Device controllers

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.



Interrupts

Interrupts are signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system.

There are three types of interrupts:

1. **Hardware Interrupts** are generated by hardware devices to signal that they need some attention from the OS. They may have just received some data (e.g., keystrokes on the keyboard or an data on the ethernet card); or they have just completed a task which the operating system previous requested, such as transferring data between the hard drive and memory.

2. **Software Interrupts** are generated by programs when they want to request a system call to be performed by the operating system.
3. **Traps** are generated by the CPU itself to indicate that some error or condition occurred for which assistance from the operating system is needed.

Interrupts are important because they give the user better control over the computer. Without interrupts, a user may have to wait for a given application to have a higher priority over the CPU to be ran. This ensures that the CPU will deal with the process immediately.

Working mechanism of Interrupts

When I/O device has finished the work given to it, it causes an interrupt. It does this by asserting a signal on a bus, that it has been assigned. The signal detected by the interrupt controller then decides what to do? If no other interrupts are pending, the interrupt controller processes the interrupts immediately, if another is in progress, then it is ignored for a moment.

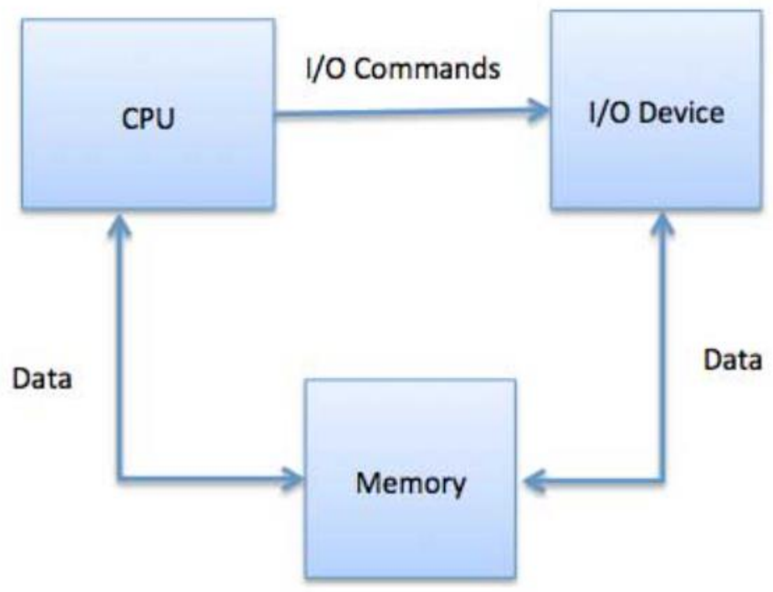
To handle the interrupt, the controller puts a number of address line specifying which device wants attention and asserts a signal that interrupt the CPU. The number of address lines is used as index to a table called interrupt vector to fetch a new program counter, this program counter points to the start of corresponding service procedure.

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.



Goals of IO Software

There are many goals of I/O software. Some of major principles of I/O software are listed below:

a. **Device independence**
   A key concept in the design of I/O software is known as device independence. It means that I/O devices should be accessible to programs without specifying the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.
b. **Uniform naming**
   Uniform Naming, simply be a string or an integer and not depend on the device in any way. In UNIX, all disks can be integrated in the file-system hierarchy in arbitrary ways so the user need not be aware of which name corresponds to which device. Here the name of file or device should be some specific string or number. It must not depend upon device in any way. All files and devices are addressed the same way: by a path name.
c. **Error handling**
   Generally, errors should be handled as close as possible to the computer hardware. It should try to correct the error itself if it can in case if the controller discovers a read error. And in case if it can’t then the device driver should handle it. If the controller discovers a read error, it should try to correct the error itself if it can. If it cannot, then the device driver should handle it, perhaps by just trying to read the block again. In many cases, error recovery can be done transparently at a low level without the upper levels even knowing about the error.

d. **Synchronous vs. Asynchronous transfers**
   Most physical input/output is asynchronous however some very high performance applications need to control all the details of the I/O, so some operating systems make asynchronous I/O available to them. The central processing unit starts the transfer and goes off to do something other until the interrupt arrives. In case if input/output operations are blocking the user programs are much easier to write. After a read system call the program is automatically suspended until the data are available in buffer. Basically, it is up to the OS to make the operation that are really asynchronous look blocking to the user programs.
e. **Buffering**
   Sometime data that come off a device can't be stored directly in its final destination. Buffering sometime has a major impact on the systems input/output performance because it involves considerable copying.
   Data comes in main memory cannot be stored directly. For example, data packets come from the network cannot be directly stored in physical memory. Packet to be put into output buffer for examining them. Some devices have several real-time constraints, so data must be put into output buffer in advance to decouple the rate at which buffer is filled and the rate at which it is emptied, in order to avoid buffer under runs.
f. **Sharable and Dedicated devices**
   Some I/O devices, such as disks, can be used by many users at the same time. No problems are caused by multiple users having open files on the same disk at the same time. Other devices, such as printers, have to be dedicated to a single user until that user is finished. Then another user can have the printer. Introducing dedicated (unshared) devices also introduces a variety of problems, such as deadlocks. Again, the operating system must be able to handle both shared and dedicated devices in a way that avoids problems.

Programmed IO

Programmable I/O is one of the I/O technique other than the interrupt-driven I/O and direct memory access (DMA). The programmed I/O was the simplest type of I/O technique for the exchanges of data or any types of communication between the processor and the external devices. With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time. The overall operation of the programmed I/O can be summaries as follow:

1. The processor is executing a program and encounters an instruction relating to I/O operation.
2. The processor then executes that instruction by issuing a command to the appropriate I/O module.

3. The I/O module will perform the requested action based on the I/O command issued by the processor (READ/WRITE) and set the appropriate bits in the I/O status register.
4. The processor will periodically check the status of the I/O module until it find that the operation is complete.

Advantages

- Simple to implement
- Very little hardware support

Disadvantages

- Busy waiting
- Ties up CPU for long period with no useful work

Interrupt driven IO

This mode uses an interrupt facility and special commands to inform the interface to issue the interrupt command when data becomes available and interface is ready for the data transfer. In the meantime, CPU keeps on executing other tasks and need not check for the flag. When the flag is set, the interface is informed and an interrupt is initiated. This interrupt causes the CPU to deviate from what it is doing to respond to the I/O transfer. The CPU responds to the signal by storing the return address from the program counter (PC) into the memory stack and then branches to service that processes the I/O request. After the transfer is complete, CPU returns to the previous task it was executing. The branch address of the service can be chosen in two ways known as vectored and non-vectored interrupt. In vectored interrupt, the source that interrupts, supplies the branch information to the CPU while in case of non-vectored interrupt the branch address is assigned to a fixed location in memory.

Difference between Programmed IO and Interrupt driven IO

Programmed IO	**Interrupt driven IO**
1. Data transfer is initiated by the means of instructions stored in the computer program. Whenever there is a request for I/O transfer the instructions are executed from the program.	1. The I/O transfer is initiated by the interrupt command issued to the CPU.
2. The CPU stays in the loop to know if the device is ready for transfer and has to continuously monitor the peripheral device.	2. There is no need for the CPU to stay in the loop as the interrupt command interrupts the CPU when the device is ready for data transfer.

3. This leads to the wastage of CPU cycles as CPU remains busy needlessly and thus the efficiency of system gets reduced.	3. The CPU cycles are not wasted as CPU and hence this method is more efficient.
4. CPU cannot do any work until the transfer is complete as it has to stay in the loop to continuously monitor the peripheral device.	4. CPU can do any other work until it is interrupted by the command indicating the readiness of device for data transfer.
5. Its module is treated as a slow module.	5. Its module is faster than programmed I/O module.
6. It is quite easy to program and understand.	6. It can be tricky and complicated to understand if one uses low level language.

DMA (Direct Memory Access)

DMA is a mechanism that provides a device controller the ability to transfer data directly to or from the memory without involving the processor. When large volumes of data are to be moved, a more efficient technique is required: direct memory access (DMA).

DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices. In order for devices to use direct memory access, they must be assigned to a DMA channel. Each type of port on a computer has a set of DMA channels that can be assigned to each connected device. For example, a PCI controller and a hard drive controller each have their own set of DMA channels.

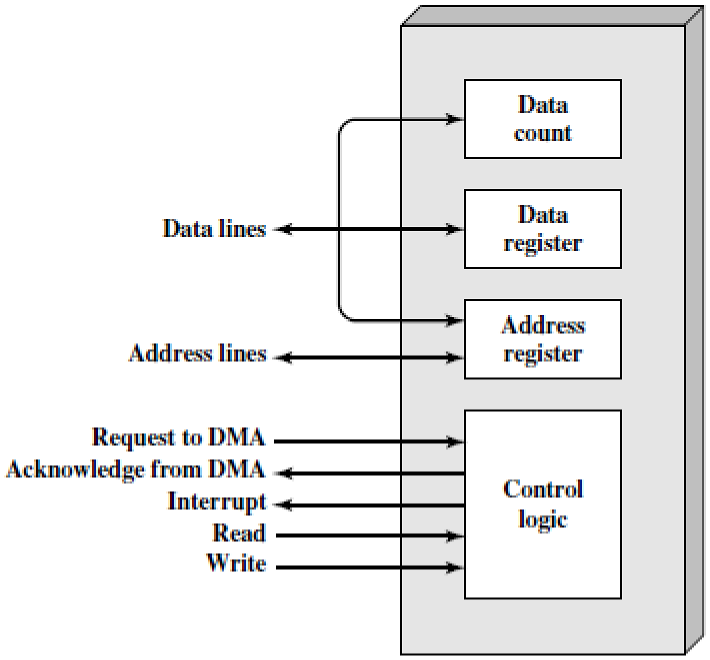


Fig: Block diagram of DMA

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.

Working procedure of DMA

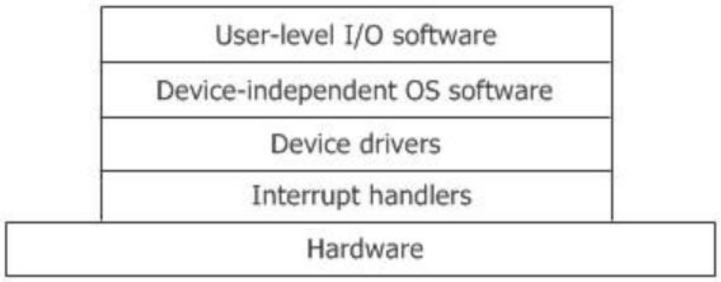
- The CPU programs the DMA controller by its registers so it knows what to transfer where. It also issues a command to disk controller telling it to read data from disk to its internal buffer and verify the checksum. When valid data are in disk's controller buffer, DMA can begin.
- The DMA controller initiates the transfer by issuing a read request over the bus to disk controller.
- Data transferred from disk controller to memory.
- When transferred completed, the disk controller sends an acknowledgement signal to DMA controller. The DMA controller then increments the memory address to use and decrement the byte count. This continues until the byte count greater than 0.
- When transfer completed the DMA controller interrupt the CPU.

I/O software layers

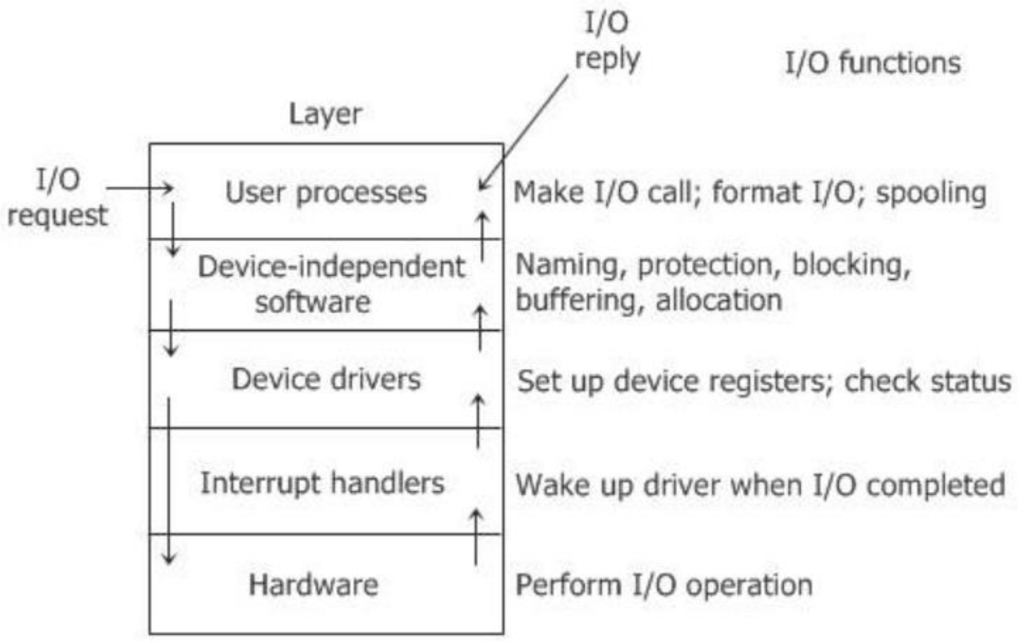
Basically, input/output software organized in the following four layers:

- Interrupt handlers
- Device drivers
- Device-independent input/output software
- User-space input/output software

In every input/output software, each of the above given four layer has a well-defined function to perform and a well-defined interface to the adjacent layers. The figure given below shows all the layers along with hardware of the input/output software system.



Here is another figure shows all the layers of the input/output software system along with their principal functions.


I/O reply
I/O functions
Layer
I/O request
User processes
Make I/O call; format I/O; spooling
Device-independent software
Naming, protection, blocking, buffering, allocation
Device drivers
Set up device registers; check status
Interrupt handlers
Wake up driver when I/O completed
Hardware
Perform I/O operation

Interrupt Handlers

Whenever the interrupt occurs, then the interrupt procedure does whatever it has to in order to handle the interrupt.

Device Drivers

Basically, device driver is a device-specific code just for controlling the input/output device that are attached to the computer system.

Device-Independent Input/Output Software

In some of the input/output software is device specific, and other parts of that input/output software are device-independent.

The exact boundary between the device-independent software and drivers is device dependent, just because of that some functions that could be done in a device-independent way sometime be done in the drivers, for efficiency or any other reasons.

Here are the list of some functions that are done in the device-independent software:

- Uniform interfacing for device drivers
- Buffering
- Error reporting
- Allocating and releasing dedicated devices
- Providing a device-independent block size

User-Space Input/Output Software

Generally most of the input/output software is within the operating system (OS), and some small part of that input/output software consists of libraries that are linked with the user programs and even whole programs running outside the kernel.

Disk structure

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

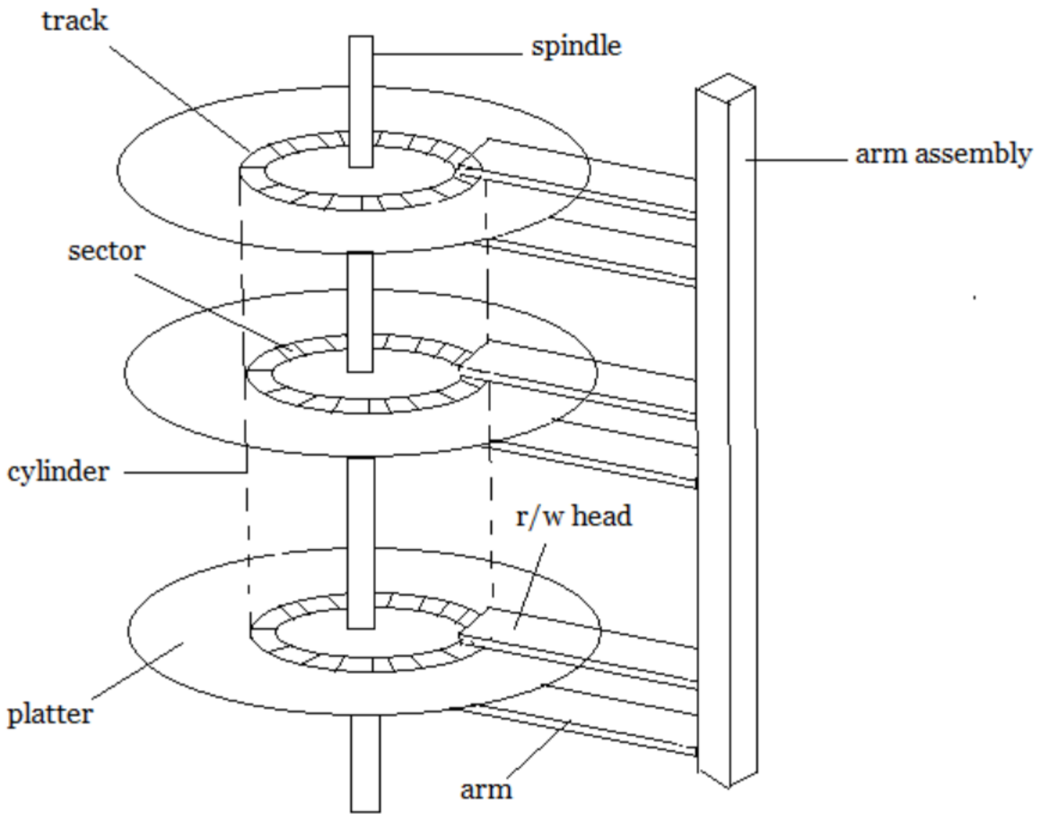


Fig: structure of a magnetic disk

Disk structure key terms

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:

  Disk Access Time = Seek Time + Rotational Latency + Transfer Time
- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all

requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

- **Transmission Time:** To access a particular record, first the arm assembly must be moved to the appropriate cylinder, and then rotate the disk until it is immediately under the read write head. The time taken to access the whole record is called transmission time.

Disk Scheduling

Various disk scheduling algorithms are:

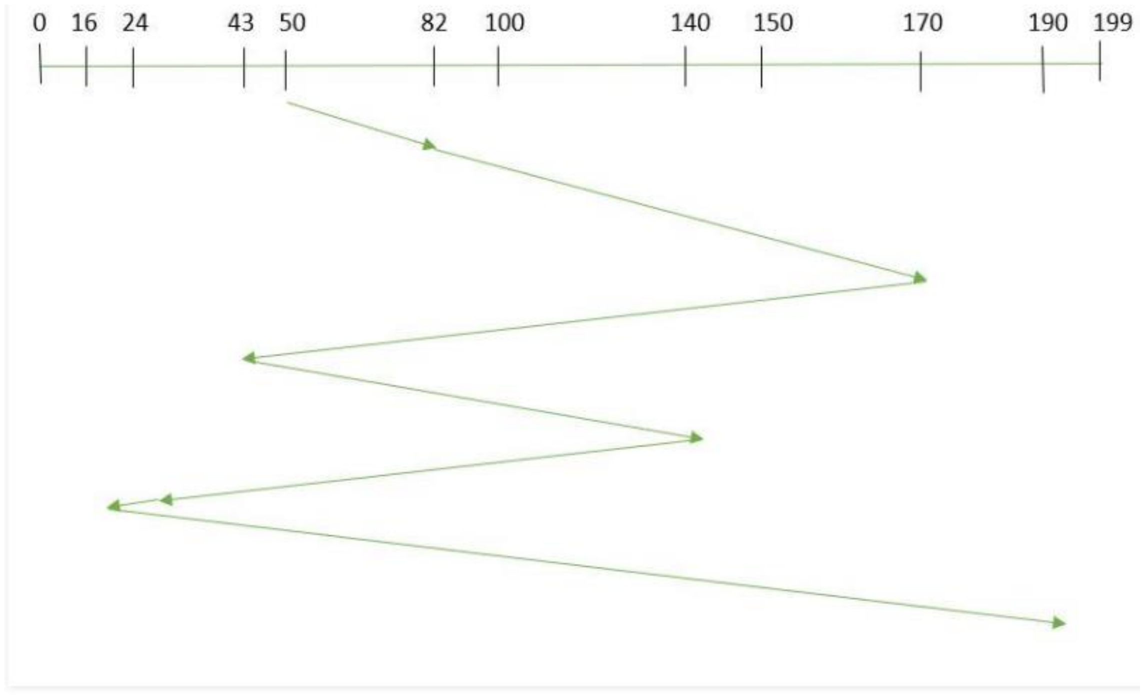
1. First Come-First Serve (FCFS)
2. Shortest Seek Time First (SSTF)
3. Elevator (SCAN)
4. Circular SCAN (C-SCAN)
5. LOOK
6. C-LOOK

1. First Come-First Serve (FCFS)

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50



So, total seek time:

=(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)

=642

Advantages:

- Every request gets a fair chance
- No indefinite postponement

Disadvantages:

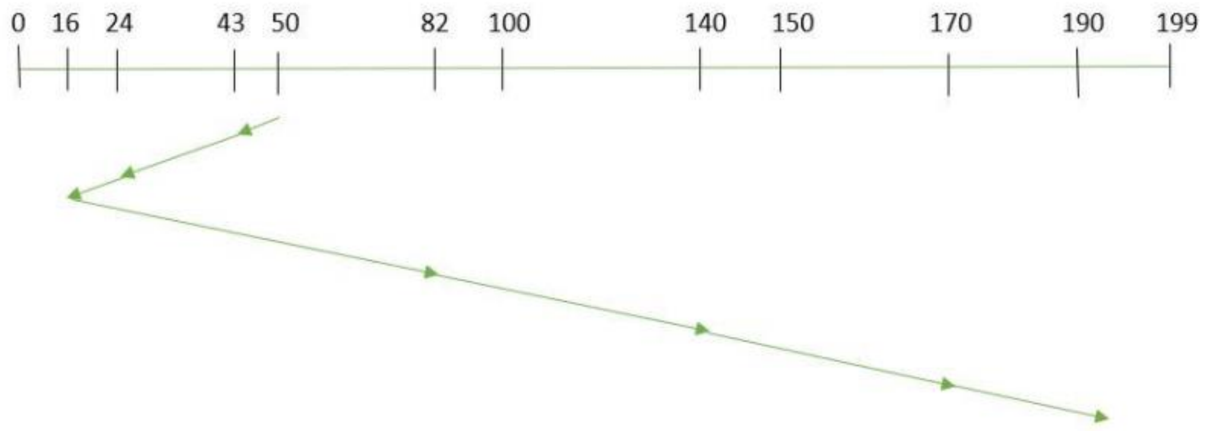
- Does not try to optimize seek time
- May not provide the best possible service

2. Shortest Seek Time First (SSTF)

In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system. Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50



So, total seek time:

=(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-40)+(190-170)
=208

Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

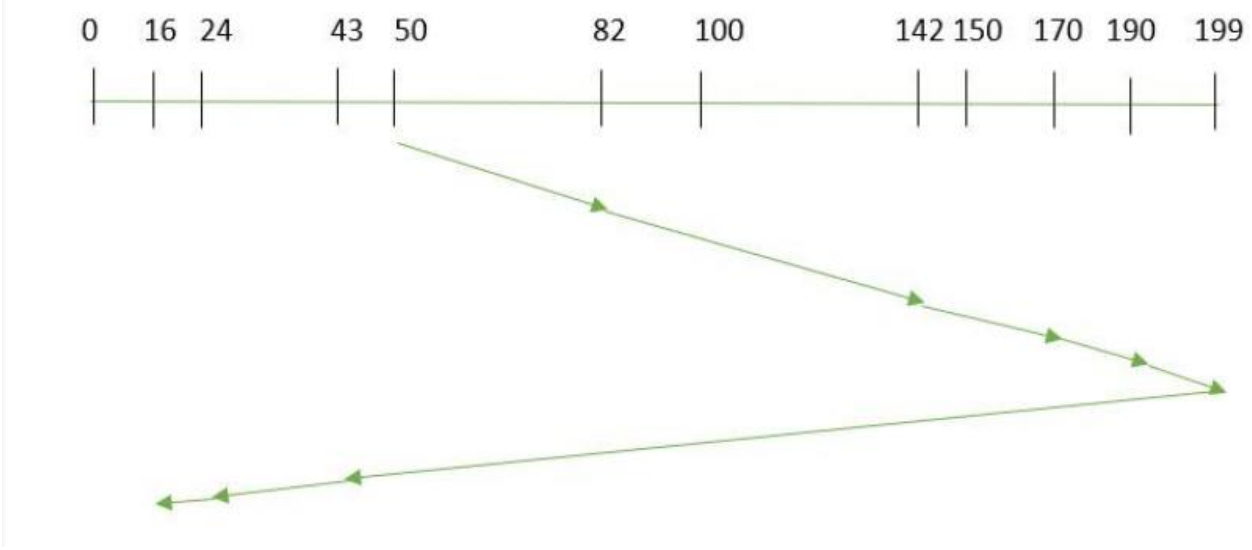
- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

3. Elevator (SCAN)

In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



Therefore, the seek time is calculated as:
=(199-50)+(199-16)

=332

Advantages:

- High throughput
- Low variance of response time
- Average response time

Disadvantages:

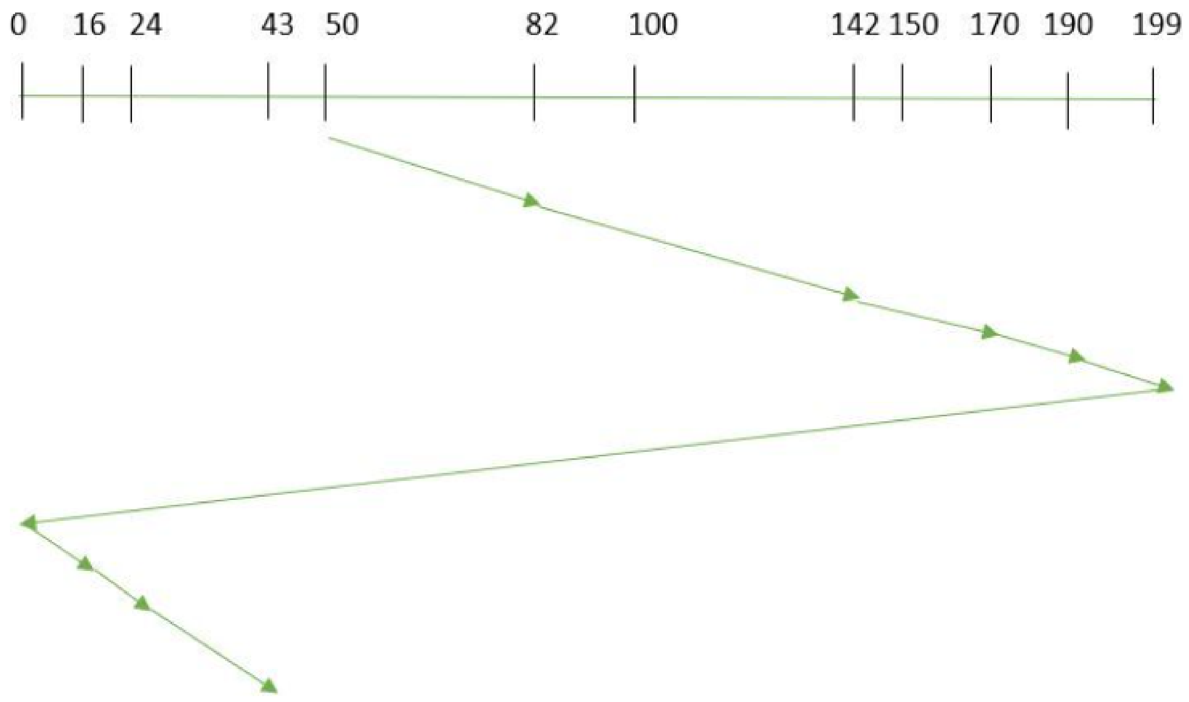
- Long waiting time for requests for locations just visited by disk arm

4. Circular SCAN (C-SCAN)

In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



Seek time is calculated as:

=(199-50)+(199-0)+(43-0)

=391

Advantages:

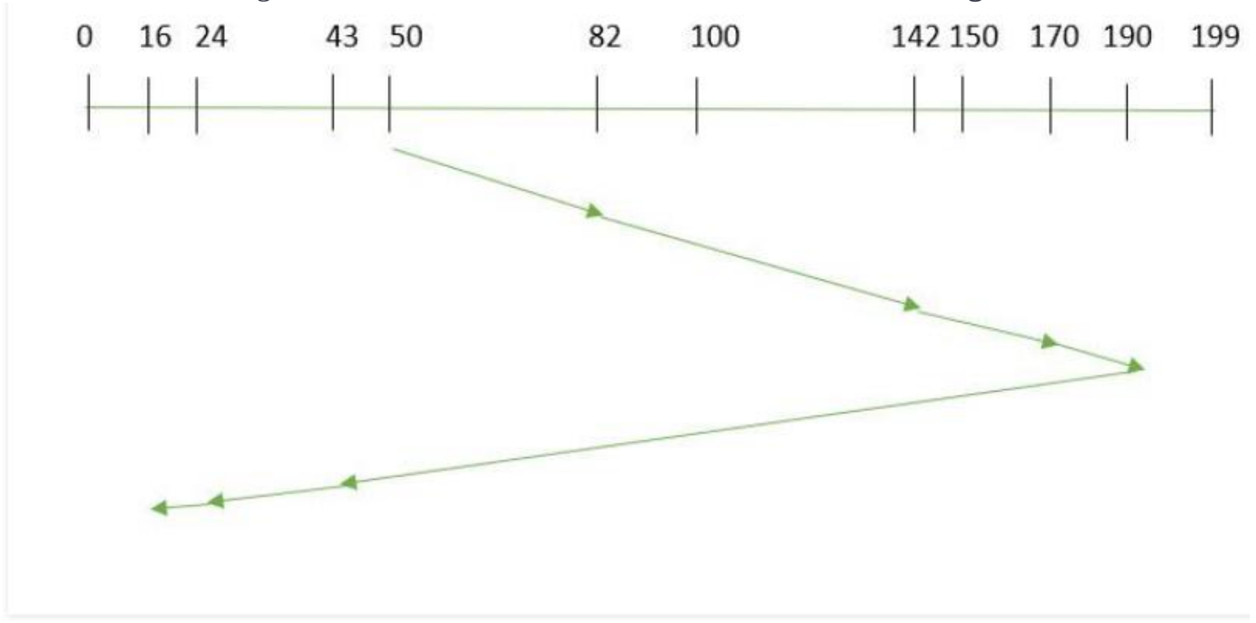
- Provides more uniform wait time compared to SCAN

5. LOOK

It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



So, the seek time is calculated as:

=(190-50)+(190-16)

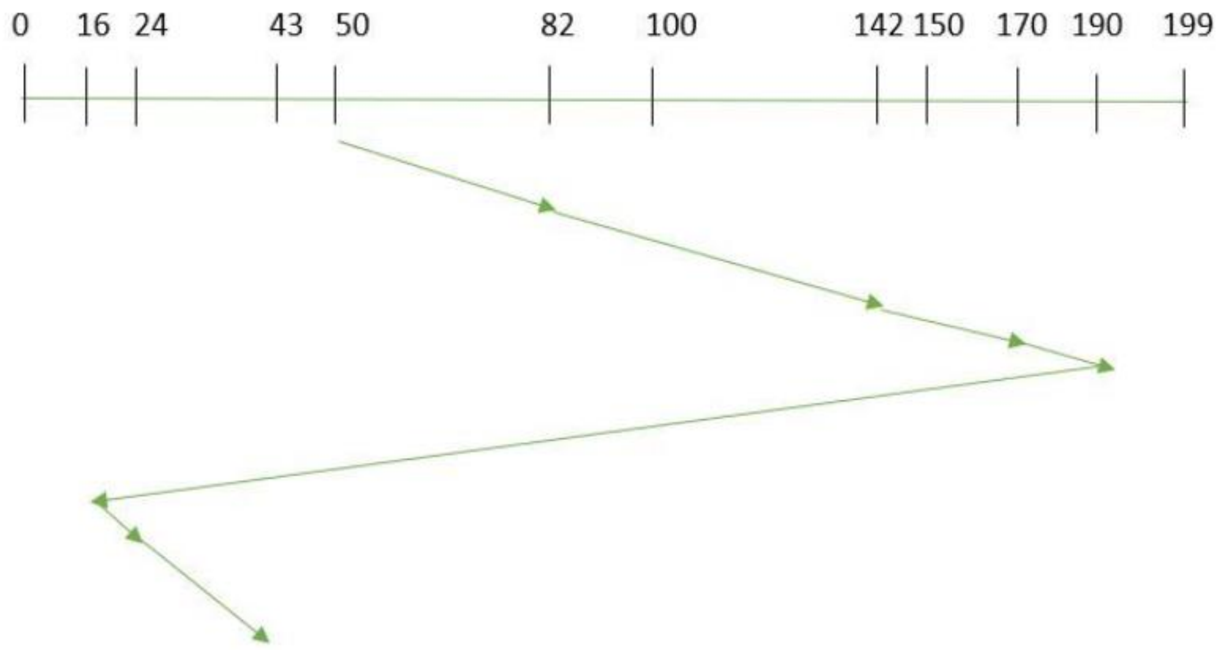
=314

6. C-LOOK

As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value"**



So, the seek time is calculated as:

=(190-50)+(190-16)+(43-16)
=341