UNIT-5

# Input/Output Management

## Overview of Mass Storage Structures

### Magnetic Disks

- Traditional magnetic disks have the following basic structure:
  - One or more *platters* in the form of disks covered with magnetic media. *Hard disk* platters are made of rigid metal, while "*floppy*" disks are made of more flexible plastic.
  - Each platter has two working *surfaces*. Older hard disk drives would sometimes not use the very top or bottom surface of a stack of platters, as these surfaces were more susceptible to potential damage.
  - Each working surface is divided into a number of concentric rings called *tracks*. The collection of all tracks that are the same distance from the edge of the platter, ( i.e. all tracks immediately above one another in the following diagram ) is called a *cylinder*.
  - Each track is further divided into *sectors*, traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes. ( Sectors also include a header and a trailer, including checksum information among other things. Larger sector sizes reduce the fraction of the disk consumed by headers and trailers, but increase internal fragmentation and the amount of disk that must be marked bad in the case of errors. )
  - The data on a hard drive is read by read-write *heads*. The standard configuration ( shown below ) uses one head per surface, each on a separate *arm*, and controlled by a common *arm assembly* which moves all heads simultaneously from one cylinder to another. ( Other configurations, including independent read-write heads, may speed up disk access, but involve serious technical difficulties. )
  - The storage capacity of a traditional disk drive is equal to the number of heads ( i.e. the number of working surfaces ), times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector. A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located.

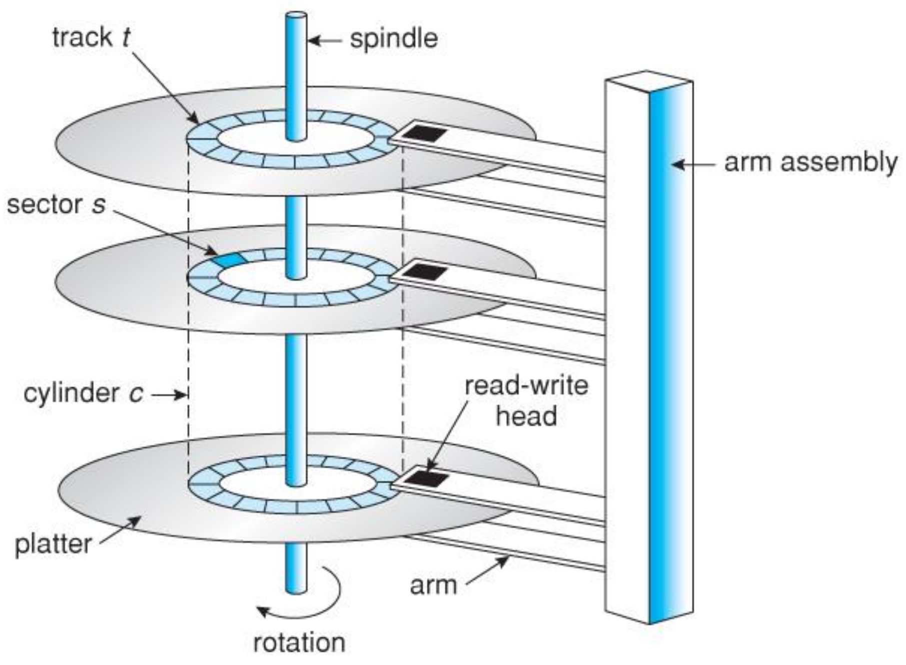


Figure - Moving-head disk mechanism.

- In operation the disk rotates at high speed, such as 7200 rpm ( 120 revolutions per second. ) The rate at which data can be transferred from the disk to the computer is composed of several steps:
    - The *positioning time*, a.k.a. the *seek time* or *random access time* is the time required to move the heads from one cylinder to another, and for the heads to settle down after the move. This is typically the slowest step in the process and the predominant bottleneck to overall transfer rates.
    - The *rotational latency* is the amount of time required for the desired sector to rotate around and come under the read-write head.This can range anywhere from zero to one full revolution, and on the average will equal one-half revolution. This is another physical step and is usually the second slowest step behind seek time. ( For a disk rotating at 7200 rpm, the average rotational latency would be 1/2 revolution / 120 revolutions per second, or just over 4 milliseconds, a long time by computer standards.
    - The *transfer rate*, which is the time required to move the data electronically from the disk to the computer. ( Some authors may also use the term transfer rate to refer to the overall transfer rate, including seek time and rotational latency as well as the electronic data transfer rate. )
- Disk heads "fly" over the surface on a very thin cushion of air. If they should accidentally contact the disk, then a *head crash* occurs, which may or may not permanently damage the disk or even destroy it completely. For this reason it is normal to *park* the disk heads when turning a computer off, which means to move the heads off the disk or to an area of the disk where there is no data stored.
- Floppy disks are normally *removable*. Hard drives can also be removable, and some are even *hot-swappable*, meaning they can be removed while the computer is running, and a new hard drive inserted in their place.
- Disk drives are connected to the computer via a cable known as the *I/O Bus*. Some of the common interface formats include Enhanced Integrated Drive Electronics, EIDE; Advanced

Technology Attachment, ATA; Serial ATA, SATA, Universal Serial Bus, USB; Fiber Channel, FC, and Small Computer Systems Interface, SCSI.

- The *host controller* is at the computer end of the I/O bus, and the *disk controller* is built into the disk itself. The CPU issues commands to the host controller via I/O ports. Data is transferred between the magnetic surface and onboard *cache* by the disk controller, and then the data is transferred from that cache to the host controller and the motherboard memory at electronic speeds.

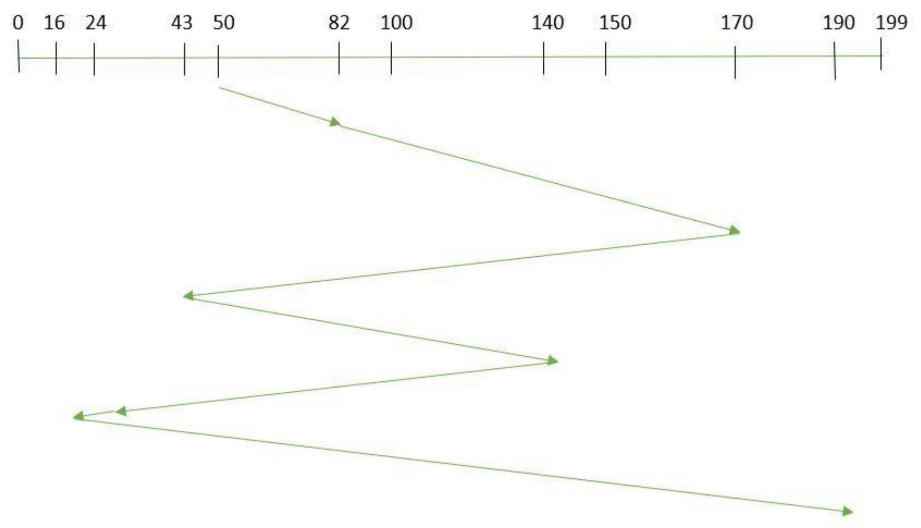
**Disk Scheduling**

- As mentioned earlier, disk transfer speeds are limited primarily by *seek times* and *rotational latency*. When multiple requests are to be processed there is also some inherent delay in waiting for other requests to be processed.
- *Bandwidth* is measured by the amount of data transferred divided by the total amount of time from the first request being made to the last transfer being completed, ( for a series of disk requests. )
- Both bandwidth and access time can be improved by processing requests in a good order.
- Disk requests include the disk address, memory address, number of sectors to transfer, and whether the request is for reading or writing.

**FCFS (First Come First Serve)**

FCFS is the simplest of all 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.

Q.The order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is: 50.Find total distance covered by the disk arm.



*First Come First Serve*

**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 overhead movement (total distance covered by the disk arm) =

(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

**Advantages of FCFS**

Here are some of the advantages of First Come First Serve.

- Every request gets a fair chance
- No indefinite postponement

**Disadvantages of FCFS**

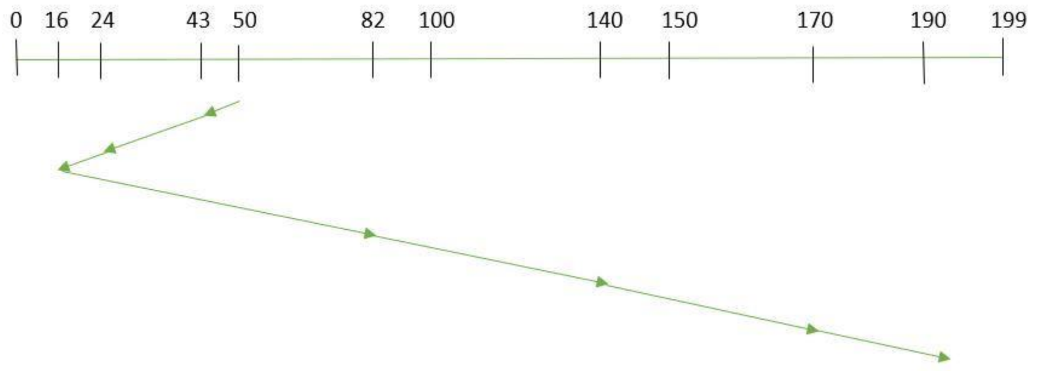
Here are some of the disadvantages of First Come First Serve.

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

**SSTF (Shortest Seek Time First)**

In SSTF (Shortest Seek Time First), requests having the 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 the system. Let us understand this with the help of an example.

**Example:**



*Shortest Seek Time First*

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

So,

total overhead movement (total distance covered by the disk arm) =

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

**Advantages of Shortest Seek Time First**

Here are some of the advantages of Shortest Seek Time First.

- The average Response Time decreases
- Throughput increases

**Disadvantages of Shortest Seek Time First**

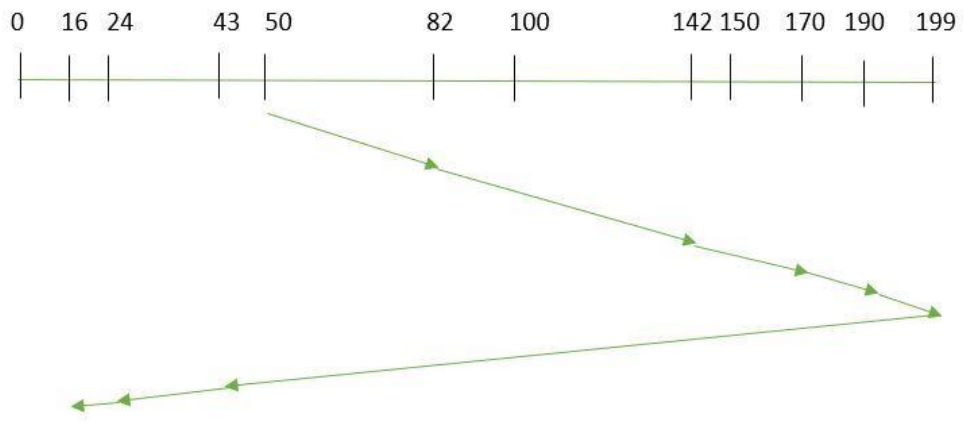
Here are some of the disadvantages of Shortest Seek Time First.

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

**SCAN**

In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an **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:**



*SCAN Algorithm*

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 total overhead movement (total distance covered by the disk arm) is calculated as

= (199-50) + (199-16) = 332

**Advantages of SCAN Algorithm**

Here are some of the advantages of the SCAN Algorithm.

- High throughput

- Low variance of response time
- Average response time

**Disadvantages of SCAN Algorithm**

Here are some of the disadvantages of the SCAN Algorithm.

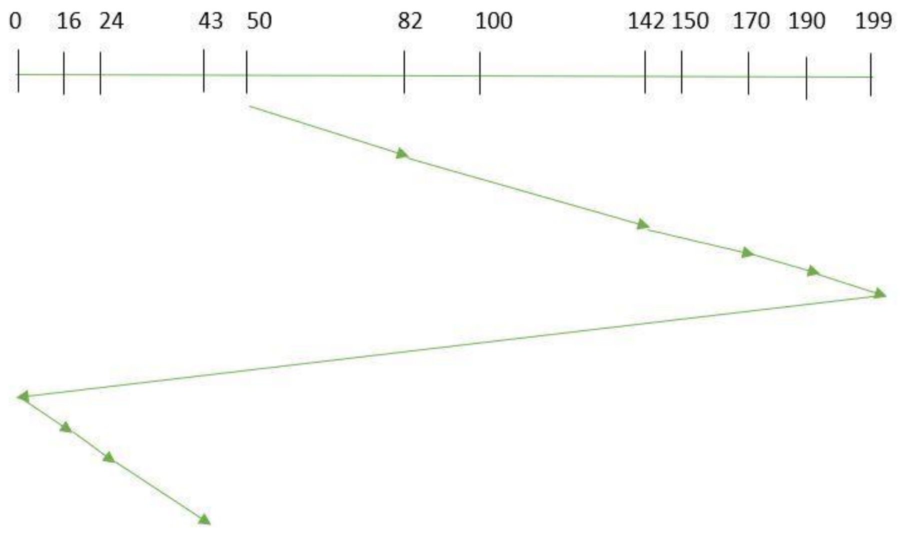
- Long waiting time for requests for locations just visited by disk arm

**C-SCAN**

In the 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 the *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 the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).

**Example:**



*Circular SCAN*

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 total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50) + (199-0) + (43-0) = 391

**Advantages of C-SCAN Algorithm**

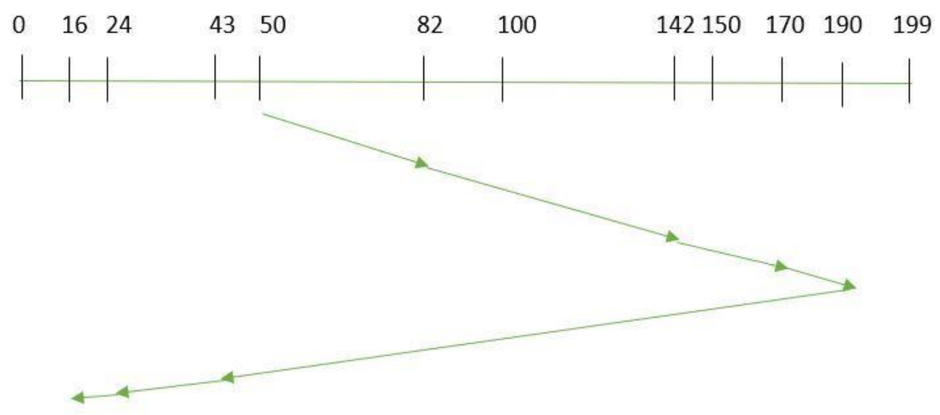
Here are some of the advantages of C-SCAN.

- Provides more uniform wait time compared to SCAN.

**LOOK**

LOOK Algorithm 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:**



*LOOK Algorithm*

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 total overhead movement (total distance covered by the disk arm) is calculated as:

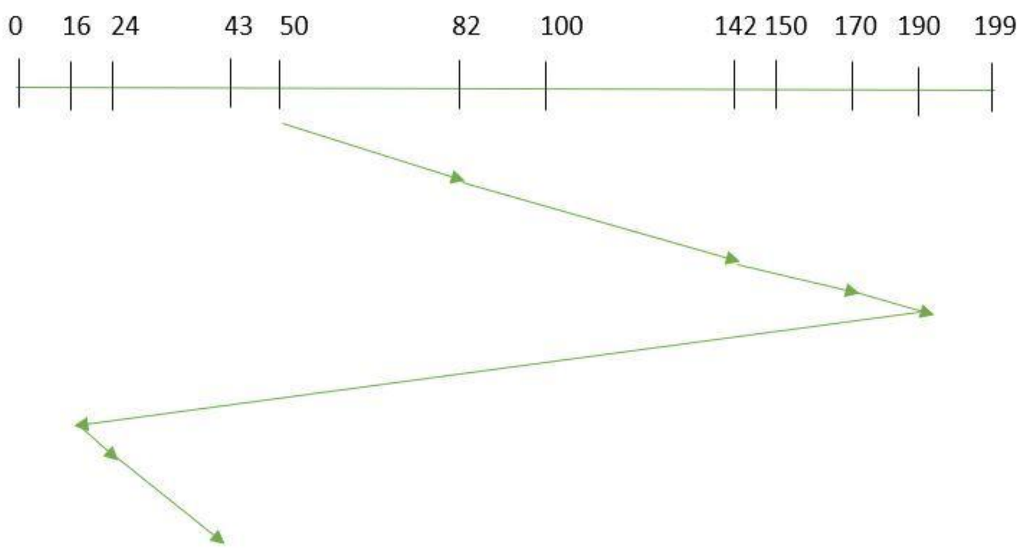
= (190-50) + (190-16) = 314

**C-LOOK**

As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the 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:**

1. 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"**



*C-LOOK*

So, the total overhead movement (total distance covered by the disk arm) is calculated as

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

**6 Selection of a Disk-Scheduling Algorithm**

- With very low loads all algorithms are equal, since there will normally only be one request to process at a time.
- For slightly larger loads, SSTF offers better performance than FCFS, but may lead to starvation when loads become heavy enough.
- For busier systems, SCAN and LOOK algorithms eliminate starvation problems.
- The actual optimal algorithm may be something even more complex than those discussed here, but the incremental improvements are generally not worth the additional overhead.
- Some improvement to overall filesystem access times can be made by intelligent placement of directory and/or inode information. If those structures are placed in the middle of the disk instead of at the beginning of the disk, then the maximum distance from those structures to data blocks is reduced to only one-half of the disk size. If those structures can be further distributed and furthermore have their data blocks stored as close as possible to the corresponding directory structures, then that reduces still further the overall time to find the disk block numbers and then access the corresponding data blocks.
- On modern disks the rotational latency can be almost as significant as the seek time, however it is not within the OSes control to account for that, because modern disks do not reveal their internal sector mapping schemes, ( particularly when bad blocks have been remapped to spare sectors. )
    - Some disk manufacturers provide for disk scheduling algorithms directly on their disk controllers, ( which do know the actual geometry of the disk as well as any remapping),

so that if a series of requests are sent from the computer to the controller then those requests can be processed in an optimal order.

- Unfortunately there are some considerations that the OS must take into account that are beyond the abilities of the on-board disk-scheduling algorithms, such as priorities of some requests over others, or the need to process certain requests in a particular order. For this reason OSes may elect to spoon-feed requests to the disk controller one at a time in certain situations.

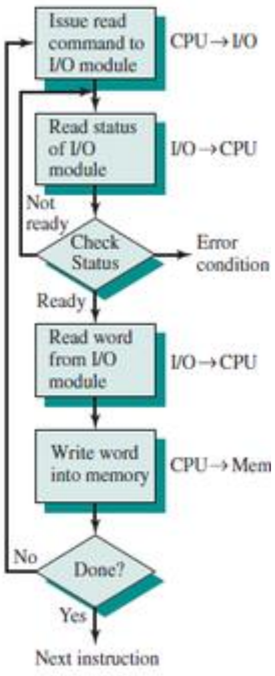
The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

**Mode of Transfer:**

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes. Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access( DMA).

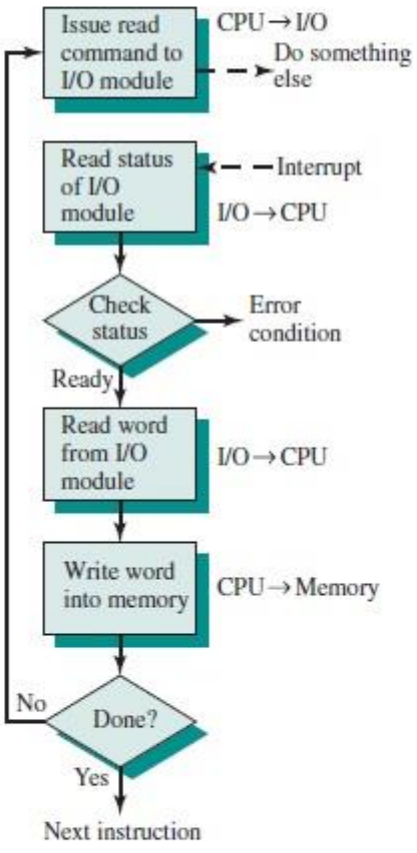
1. **Programmed I/O**: It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.
   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.



2. **Interrupt-** initiated I/O: Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

    - The I/O transfer rate is limited by the speed with which the processor can test and service a device.
    - The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.
    - Terms:
        - Hardware Interrupts: Interrupts present in the hardware pins.
        - Software Interrupts: These are the instructions used in the program whenever the required functionality is needed.
        - Vectored interrupts: These interrupts are associated with the static vector address.
        - Non-vectored interrupts: These interrupts are associated with the dynamic vector address.
        - Maskable Interrupts: These interrupts can be enabled or disabled explicitly.
        - Non-maskable interrupts: These are always in the enabled state. we cannot disable them.
        - External interrupts: Generated by external devices such as I/O.

- Internal interrupts: These devices are generated by the  internal components of the processor such as power failure, error instruction, temperature sensor, etc.
- Synchronous interrupts: These interrupts are controlled by the fixed time interval. All the interval interrupts are called as synchronous interrupts.
- Asynchronous interrupts: These are initiated based on the feedback of previous instructions. All the external interrupts are called as asynchronous interrupts.

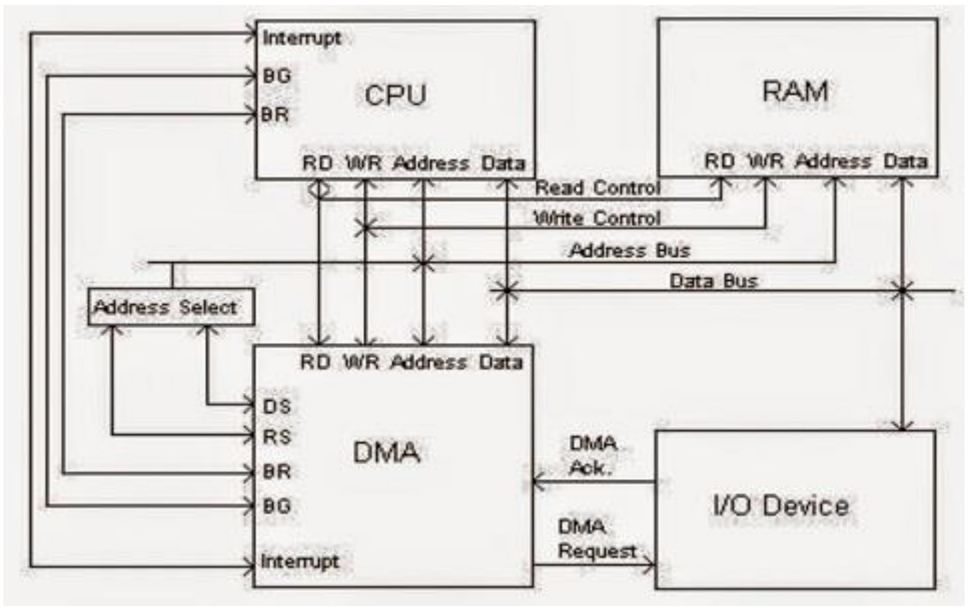


3. **Direct Memory Access**:

**Working of DMA Controller**

DMA controller has to share the bus with the processor to make the data transfer. The device that holds the bus at a given time is called bus master. When a transfer from I/O device to the memory or vice versa has to be made, the processor stops the execution of the current program, increments the program counter, moves data over stack then sends a DMA select signal to DMA controller over the address bus.

If the DMA controller is free, it requests the control of bus from the processor by raising the bus request signal. Processor grants the bus to the controller by raising the bus grant signal, now DMA controller is the bus master. The processor initiates the DMA controller by sending the memory addresses, number of blocks of data to be transferred and direction of data transfer. After assigning the data transfer task to the DMA controller, instead of waiting ideally till completion of data transfer, the processor resumes the execution of the program after retrieving instructions from the stack.



**Difference Between Blocking and Non-Blocking I/O**

| **Blocking I/O** | **Non-Blocking I/O** |
|---|---|
| Blocking I/O is when a process is waiting for an I/O operation to complete. | Non-blocking I/O means that when user call **GetData().** |
| For example, if users are trying to write data from your application into a file on disk, then if another process wants access too at that same time then they would have their own read or write operations started at the same time as yours. This can cause problems because there's no guarantee that they'll finish before the user does. | In our example above, it returns immediately without waiting until all the requested data has been transferred over (and returned). The result here could be different depending on how busy those other processes were when calling GetData(), So this doesn't really matter much here; just know that what we're doing now won't block any other programs from getting their jobs done either since everything happens on its own schedule – even though it might seem like nothing else was happening while waiting around forever. |

**What is Buffering in OS?**

In operating systems, **buffering** is a technique which is used to enhance the performance of I/O operations of the system. Basically, buffering in operating system is a method of storing data in a buffer or cache temporarily, this buffered data then can be accessed more quickly as compared to the original source of the data.

In a computer system, data is stored on several devices like hard discs, magnetic tapes, optical discs and network devices. In the case, when a process requires to read or write data from one of these storage devices, it has to wait while the device retrieves or stores the data. This waiting time could be very high, especially for those devices which are slow or have a high latency.

This problem can be addressed by buffering. Buffering provides a temporary storage area, called **buffer.** Buffer can store data before it is sent to or retrieved from the storage device. When the buffer is fully occupied, then data is sent to the storage device in a batch, this will reduce the number of access operations required and hence improves the performance of the system.

**Reasons of Buffering**

The following three are the major reasons for buffering in operating systems −

- Buffering creates a synchronization between two devices having different processing speed. For example, if a hard disc (supplier of data) has high speed and a printer (accepter of data) has low speed, then buffering is required.
- Buffering is also required in cases where two devices have different data block sizes.
- Buffering is also required to support copy semantics for application I/O operations.

**Types of Buffering**

In operating systems, the following three types of buffering techniques are there −

- Single Buffering
- Double Buffering
- Circular Buffering

Let us discuss each type of buffering technique in detail.

Single Buffering

It is the simplest buffering that operating system can support. In the case of single buffering, when a process issues an I/O request, the operating system assigns a buffer (or cache) in the system potion of the main memory to the operation. Then, the input transfers are made to the buffer and are moved to the user space when needed.

Double Buffering

Double buffering is an extended variant of single buffering. In this type buffering, a process can transfer data to or from one buffer while the operating system removes or fills the other. Therefore, double buffering has two system buffers instead of one.

Circular Buffering

When more than two buffers are used, then it is called circular buffering. It is used to solve the issues associated with the double buffering technique. Sometimes, the double buffering becomes insufficient, when the process performs rapid bursts of I/O. In the circular buffer, each individual buffer acts a unit.

**Advantages of Buffering**

Some of important advantages of buffering are listed below −

- Buffering reduces the number of I/O operations required to access data.
- Buffering reduces the amount of time for that processes have to wait for the data.
- Buffering improves the performance of I/O operations as it allows data to be read or written in large blocks instead of 1 byte or 1 character at a time.
- Buffering can improve the overall performance of the system by reducing the number of system calls and context switches required for I/O operations.

**Limitations of Buffering**

Buffering also has some limitations -

- Buffers of large sizes consume significant amount of memory that can degrade the system performance.
- Buffering may cause a delay between the time data is read or written and the time it is processed by the application.
- Buffering may also impact the real-time system performance and hence, can cause synchronization issues.

he kernel provides many services related to I/O. Several services such as scheduling, caching, spooling, device reservation, and error handling – are provided by the kernel's I/O subsystem built on the hardware and device-driver infrastructure. The I/O subsystem is also responsible for protecting itself from errant processes and malicious users.

1. **I/O Scheduling** –
   To schedule a set of I/O requests means to determine a good order in which to execute them. The order in which the application issues the system call is the best choice. Scheduling can improve the overall performance of the system, can share device access permission fairly to all the processes, and reduce the average waiting time, response time, and turnaround time for I/O to complete.

OS developers implement schedules by maintaining a wait queue of the request for each device. When an application issues a blocking I/O system call, The request is placed in the queue for that device. The I/O scheduler rearranges the order to improve the efficiency of the system.

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

1. The first is to cope with a speed mismatch between the producer and consumer of a data stream.
2. The second use of buffering is to provide adaptation for data that have different data-transfer sizes.
3. The third use of buffering is to support copy semantics for the application I/O, “copy semantic ” means, suppose that an application wants to write data on a disk that is stored in its buffer. it calls the **write()** system’s call, providing a pointer to the buffer and the integer specifying the number of bytes to write.

3. **Caching** –
   A *cache* is a region of fast memory that holds a copy of data. Access to the cached copy is much easier than the original file. For instance, the instruction of the currently running process is stored on the disk, cached in physical memory, and copied again in the CPU’s secondary and primary cache.

The main difference between a buffer and a cache is that a buffer may hold only the existing copy of a data item, while a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

4. **Spooling and Device Reservation** –
   A *spool* is a buffer that holds the output of a device, such as a printer that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixes together.

The OS solves this problem by preventing all output from continuing to the printer. The output of all applications is spooled in a separate disk file. When an application finishes printing then the spooling system queues the corresponding spool file for output to the printer.

5. **Error Handling** –
   An Os that uses protected memory can guard against many kinds of hardware and application errors so that a complete system failure is not the usual result of each minor mechanical glitch, Devices, and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded or for permanent reasons, as when a disk controller becomes defective.
6. **I/O Protection** –
   Errors and the issue of protection are closely related. A user process may attempt to issue illegal I/O instructions to disrupt the normal function of a system. We can use the various mechanisms to ensure that such disruption cannot take place in the system.

To prevent illegal I/O access, we define all I/O instructions to be privileged instructions. The user cannot issue I/O instruction directly.

**The Kernel I/O Subsystem in Operating System**

An Operating System (OS) is a complex software program that manages the hardware and software resources of a computer system. One of the critical components of an OS is the Kernel I/O Subsystem, which provides an interface between the operating system and input/output (I/O) devices. The Kernel I/O Subsystem manages the I/O requests made by the user applications and translates them into hardware commands that the devices can understand. In this article, we will discuss the importance of the Kernel I/O Subsystem and its advantages and disadvantages.

**Importance of Kernel I/O Subsystem**

The Kernel I/O Subsystem is an essential part of any modern Operating System. It provides a unified and consistent interface to the I/O devices, which enables the user applications to access them without knowing the details of the underlying hardware. The Kernel I/O Subsystem also manages the concurrency and synchronization issues that arise when multiple applications try to access the same device simultaneously.

**Advantages of Kernel I/O Subsystem**

- **Device Independence:** The Kernel I/O Subsystem provides device independence to the user applications. It abstracts the hardware details and provides a unified interface to the devices. This means that the application developers can write code that is independent of the hardware platform, and the Kernel I/O Subsystem takes care of the hardware-specific details.
- **Efficient Resource Management:** The Kernel I/O Subsystem provides efficient resource management for the I/O devices. It manages the I/O requests and schedules them in a way that optimizes the usage of the available resources. This ensures that the I/O devices are not overutilized, and the system remains responsive.
- **Concurrency Management:** The Kernel I/O Subsystem manages the concurrency issues that arise when multiple applications try to access the same device simultaneously. It ensures that the applications get exclusive access to the device when needed and allows multiple applications to share the device when appropriate.

**Disadvantages of Kernel I/O Subsystem**

- **Complex Implementation:** The Kernel I/O Subsystem is a complex software component that requires a lot of resources to implement and maintain. Any issues with the Kernel I/O Subsystem can affect the performance and stability of the entire system.
- **Security Risks:** The Kernel I/O Subsystem can pose security risks to the system if not implemented correctly. Attackers can exploit vulnerabilities in the Kernel I/O Subsystem to gain unauthorized access to the system or cause a denial-of-service attack.

**Transforming I/O requests to hardware operations**

To understand concept let us consider example which is as follows.

**Example** –
We are reading file from disk. The application we request for will refers to data by file name. Within disk, file system maps from file name through file-system directories to obtain space allocation for file. In MS-DOS, name of file maps to number that indicates as entry in file-access table, and that entry to table tells us that which disk blocks are allocated to file. In UNIX, name maps to inode number, and inode number contains information about space-allocation. But here question arises that how connection is made from file name to disk controller?
The method that is used by MS-DOS, is relatively simple operating system. The first part of MS-DOS file name, is preceding with colon, is string that identifies that there is specific hardware device.

The UNIX uses different method from MS-DOS. It represents device names in regular file-system name space. Unlike MS-DOS file name, which has colon separator, but UNIX path name has no clear separation of device portion. In fact, no part of path name is name of device. Unix has mount table that associates with prefixes of path names with specific hardware device names.

Modern operating systems gain significant flexibility from multiple stages of lookup tables in path between request and physical device stages controller. There is general mechanisms is which is used to pass request between application and drivers. Thus, without recompiling kernel, we can introduce new devices and drivers into computer. In fact, some operating system have the ability to load device drivers on demand. At the time of booting, system firstly probes hardware buses to determine what devices are present. It is then loaded to necessary drivers, accordingly I/O request.

The typical life cycle of blocking read request, is shown in the following figure. From figure, we can suggest that I/O operation requires many steps that together consume large number of CPU cycles.

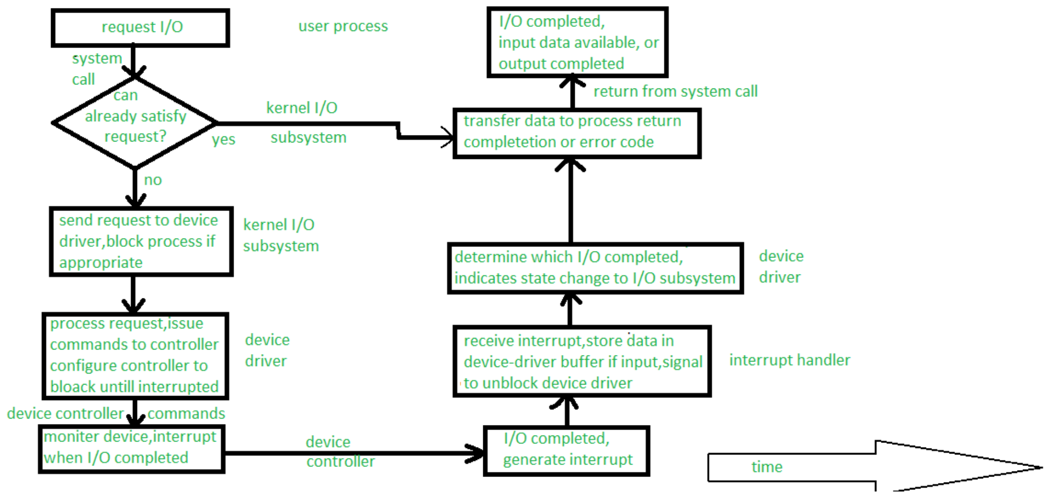

request I/O
user process
I/O completed, input data available, or output completed
system call
can already satisfy request?
return from system call
kernel I/O subsystem
yes
transfer data to process return completetion or error code
no
send request to device driver,block process if appropriate
kernel I/O subsystem
determine which I/O completed, indicates state change to I/O subsystem
device driver
process request,issue commands to controller configure controller to bloack untill interrupted
device driver
receive interrupt,store data in device-driver buffer if input,signal to unblock device driver
interrupt handler
device controller
commands
moniter device,interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

**Figure –** The life cycle of I/O request

1. Systemcall –
Whenever, any I/O request comes, process issues blocking read() system call to previously opened file descriptor of file. Basically, role of system-call code is to check parameters for correctness in kernel. If data we put in form of input is already available in buffer cache, data is going to returned to process, and in that case I/O request is completed.

2. **Alternative approach if input is not available** –
If the data is not available in buffer cache then physical I/O must be performed. The process is removes from run queue and is placed on wait queue for device, and I/O request is scheduled. After scheduling, I/O subsystem sends request to device driver via subroutine call or in-kernel message but it depends upon operating system by which mode request will send.

3. **Role of Device driver** –
After receiving the request, device driver have to receive data and it will receive data by allocating kernel buffer space and after receiving data it will schedules I/O. After all this, command will be given to device controller by writing into device-control registers.

4. **Role of Device Controller** –
Now, device controller operates device hardware. Actually, data transfer is done by device hardware.
5. **Role of DMA controller** –
After data transfer, driver may poll for status and data, or it may have set up DMA transfer into kernel memory. The transfer is managed by DMA controller. At last when transfers complete, it will generates interrupt.

6. **Role of interrupt handler** –
The interrupt is send to correct interrupt handler through interrupt-vector table. It store any necessary data, signals device driver, and returns from interrupt.

7. **Completion of I/O request –**
   When, device driver receives signal. This signal determines that I/O request has completed and also determines request’s status, signals kernel I/O subsystem that request has been completed. After transferring data or return codes to address space kernel moves process from wait queue back to ready queue.

8. **Completion of System call –**
   When process moves to ready queue it means process is unblocked. When the process is assigned to CPU, it means process resumes execution at completion of system call.

**Protection and security issues**

**Introduction:**

**System protection** in an operating system refers to the mechanisms implemented by the operating system to ensure the security and integrity of the system. System protection involves various techniques to prevent unauthorized access, misuse, or modification of the operating system and its resources.

There are several ways in which an operating system can provide system protection:

**User authentication:** The operating system requires users to authenticate themselves before accessing the system. Usernames and passwords are commonly used for this purpose.

**Access control:** The operating system uses access control lists (ACLs) to determine which users or processes have permission to access specific resources or perform specific actions.

**Encryption:** The operating system can use encryption to protect sensitive data and prevent unauthorized access.

**Firewall:** A firewall is a software program that monitors and controls incoming and outgoing network traffic based on predefined security rules.

**Antivirus software:** Antivirus software is used to protect the system from viruses, malware, and other malicious software.

**System updates and patches:** The operating system must be kept up-to-date with the latest security patches and updates to prevent known vulnerabilities from being exploited.

By implementing these protection mechanisms, the operating system can prevent unauthorized access to the system, protect sensitive data, and ensure the overall security and integrity of the system.

**What is Protection?**

**Protection** refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

**Need for Protection:**

- To prevent the access of unauthorized users

- To ensure that each active programs or processes in the system uses resources only as the stated policy
- To improve reliability by detecting latent errors

**Role of Protection:**

The role of protection is to provide a mechanism that implement policies which defines the uses of resources in the computer system. Some policies are defined at the time of design of the system, some are designed by management of the system and some are defined by the users of the system to protect their own files and programs. Every application has different policies for use of the resources and they may change over time so protection of the system is not only concern of the designer of the operating system. Application programmer should also design the protection mechanism to protect their system against misuse. Policy is different from mechanism. Mechanisms determine how something will be done and policies determine what will be done. Policies are changed over time and place to place. Separation of mechanism and policy is important for the flexibility of the system.

**Advantages of system protection in an operating system:**

1. Ensures the security and integrity of the system
2. Prevents unauthorized access, misuse, or modification of the operating system and its resources
3. Protects sensitive data
4. Provides a secure environment for users and applications
5. Prevents malware and other security threats from infecting the system
6. Allows for safe sharing of resources and data among users and applications
7. Helps maintain compliance with security regulations and standards

**Disadvantages of system protection in an operating system:**

1. Can be complex and difficult to implement and manage
2. May slow down system performance due to increased security measures
3. Can cause compatibility issues with some applications or hardware
4. Can create a false sense of security if users are not properly educated on safe computing practices
5. Can create additional costs for implementing and maintaining security measures.

## Virtualisation

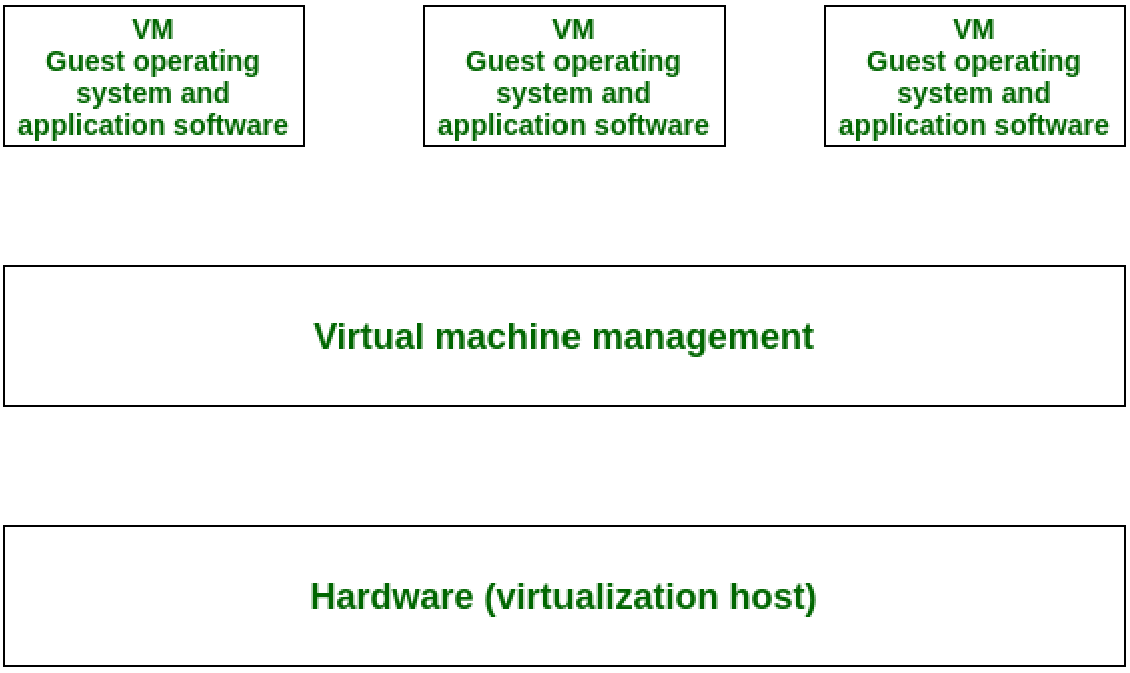
### Hypervisor

A hypervisor is a form of virtualization software used in Cloud hosting to divide and allocate the resources on various pieces of hardware. The program which provides partitioning, isolation, or abstraction is called a virtualization hypervisor. The hypervisor is a hardware virtualization technique that allows multiple guest operating systems (OS) to run on a single host system at the same time. A hypervisor is sometimes also called a virtual machine manager(VMM).

Operating system-based Virtualization refers to an operating system feature in which the kernel enables the existence of various isolated user-space instances. The installation of virtualization software also refers to Operating system-based virtualization. It is installed over a pre-existing operating system and that operating system is called the host operating system.

In virtualization, a user installs the virtualization software in the operating system of his system like any other program and utilizes this application to operate and generate various virtual machines. Here, the virtualization software allows direct access to any of the created virtual machines to the user. As the host OS can provide hardware devices with the mandatory support, operating system virtualization may affect compatibility issues of hardware even when the hardware driver is not allocated to the virtualization software.

Virtualization software is able to convert hardware IT resources that require unique software for operation into virtualized IT resources. As the host OS is a complete operating system in itself, many OS-based services are available as organizational management and administration tools can be utilized for the virtualization host management.


VM
Guest operating system and application software
VM
Guest operating system and application software
VM
Guest operating system and application software
Virtual machine management
Hardware (virtualization host)

Some major operating system-based services are mentioned below:

1. Backup and Recovery.
2. Security Management.
3. Integration to Directory Services.

Various major operations of Operating System Based Virtualization are described below:

1. Hardware capabilities can be employed, such as the network connection and CPU.
2. Connected peripherals with which it can interact, such as a webcam, printer, keyboard, or Scanners.
3. Data that can be read or written, such as files, folders, and network shares.

The Operating system may have the capability to allow or deny access to such resources based on which the program requests them and the user account in the context of which it runs. OS may also hide these resources, which leads that when a computer program computes them, they do not appear in the enumeration results. Nevertheless, from a programming perspective, the computer program has interacted with those resources and the operating system has managed an act of interaction.

With operating-system-virtualization or containerization, it is probable to run programs within containers, to which only parts of these resources are allocated. A program that is expected to perceive the whole computer, once run inside a container, can only see the allocated resources and believes them to be all that is available. Several containers can be formed on each operating system, to each of which a subset of the computer's resources is allocated. Each container may include many computer programs. These programs may run parallel or distinctly, even interrelate with each other.

**features of operating system-based virtualization are:**

- **Resource isolation:** Operating system-based virtualization provides a high level of resource isolation, which allows each container to have its own set of resources, including CPU, memory, and I/O bandwidth.
- **Lightweight:** Containers are lightweight compared to traditional virtual machines as they share the same host operating system, resulting in faster startup and lower resource usage.
- **Portability:** Containers are highly portable, making it easy to move them from one environment to another without needing to modify the underlying application.
- **Scalability:** Containers can be easily scaled up or down based on the application requirements, allowing applications to be highly responsive to changes in demand.
- **Security:** Containers provide a high level of security by isolating the containerized application from the host operating system and other containers running on the same system.
- **Reduced Overhead:** Containers incur less overhead than traditional virtual machines, as they do not need to emulate a full hardware environment.
- **Easy Management:** Containers are easy to manage, as they can be started, stopped, and monitored using simple commands.

Operating system-based virtualization can raise demands and problems related to performance overhead, such as:

1. The host operating system employs CPU, memory, and other hardware IT resources.
2. Hardware-related calls from guest operating systems need to navigate numerous layers to and from the hardware, which shrinkage overall performance.
3. Licenses are frequently essential for host operating systems, in addition to individual licenses for each of their guest operating systems.

**Advantages of Operating System-Based Virtualization:**

- **Resource Efficiency:** Operating system-based virtualization allows for greater resource efficiency as containers do not need to emulate a complete hardware environment, which reduces resource overhead.
- **High Scalability:** Containers can be quickly and easily scaled up or down depending on the demand, which makes it easy to respond to changes in the workload.**Easy Management:** Containers are easy to manage as they can be managed through simple commands, which makes it easy to deploy and maintain large numbers of containers.

**Reduced Costs:** Operating system-based virtualization can significantly reduce costs, as it requires fewer resources and infrastructure than traditional virtual machines.

- **Faster Deployment:** Containers can be deployed quickly, reducing the time required to launch new applications or update existing ones.
- **Portability:** Containers are highly portable, making it easy to move them from one environment to another without requiring changes to the underlying application.

**Disadvantages of Operating System-Based Virtualization:**

- **Security:** Operating system-based virtualization may pose security risks as containers share the same host operating system, which means that a security breach in one container could potentially affect all other containers running on the same system.
- **Limited Isolation:** Containers may not provide complete isolation between applications, which can lead to performance degradation or resource contention.
- **Complexity:** Operating system-based virtualization can be complex to set up and manage, requiring specialized skills and knowledge.
- **Dependency Issues:** Containers may have dependency issues with other containers or the host operating system, which can lead to compatibility issues and hinder deployment.
- **Limited Hardware Access:** Containers may have limited access to hardware resources, which can limit their ability to perform certain tasks or applications that require direct hardware access.