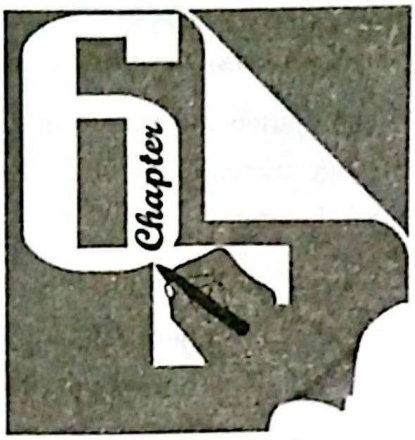


INPUT / OUTPUT DEVICE MANAGEMENT

CHAPTER OUTLINE

After comprehensive study of this chapter, you will be able to:

- Principle of I/O Hardware: I/O devices, Device Controllers, Memory Mapped I/O, Direct Memory Access
- Principle of I/O Software: Goals of I/O Software, Program I/O, Interrupt Driven I/O, I/O using DMA
- I/O Software Layers: Interrupt Handler, Device Drivers, Device Independent I/O Software, User space I/O Software
- Disk: Disk Hardware, Disk Scheduling: Seek Time, Rational Delay, Transfer Time;
- Disk Scheduling Algorithms: FCFS Scheduling, SSTF scheduling, SCAN Scheduling, C-SCAN Scheduling, Lock Scheduling.

INTRODUCTION

All computers have physical devices for acquiring input and producing output. OS is responsible to manage and control all the I/O operations and I/O devices. Device management generally performs the following:

- Installing device and component-level drivers and related software
- Configuring a device so it performs as expected using the bundled operating system, business/workflow software and with other hardware devices.
- Implementing security measures and processes.

Devices usually refer to physical devices such as computers, laptops, servers, mobile phones and more. They could also be virtual, however such as virtual machines or virtual switches. In Windows, device management is also an administrative module that is used for managing or configuring the physical devices, ports and interfaces of a computer or server.

One of the important jobs of an operating system is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, bit-mapped screen, LED, analog-to-digital converter, on/off switch, network connections, audio I/O, printers etc. An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application.

CLASSIFICATION OF I/O DEVICES

I/O devices can be divided into following three categories

- Machine readable or Block devices
- User readable or Character devices (Human)
- Communications devices

Block devices

A block device is one with which the driver communicates by sending entire blocks of data. Here information in fixed blocks with each block has its own address. Each block can be read from/written to independently. For example, Hard disks, USB cameras, Disk-On-Key, tapes, sensors etc.

A block device generally requires fewer pins and can thus be placed in a smaller package than a word-addressed device. In addition, a block device interface to slower forms of memory, such as rotating media and NAND flash, is much easier to build than a word-addressed interface, and these slower forms of memory generally have a much larger capacity dollar-for-dollar than NOR flash.

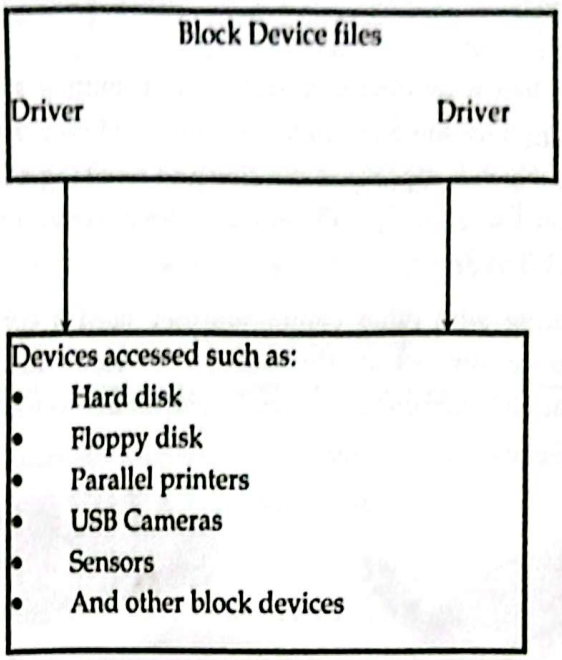


Fig 6.1: Block Device

Character Devices

A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). It accepts a stream of characters with no attention paid to block structure. It is not addressable and has no seek ability. For example, printers, graphical terminals, screen, keyboard, mouse, serial ports, parallel ports, sounds cards etc. Character devices are devices that do not have physically addressable storage media, such as tape drives or serial ports, where I/O is normally performed in a byte stream.

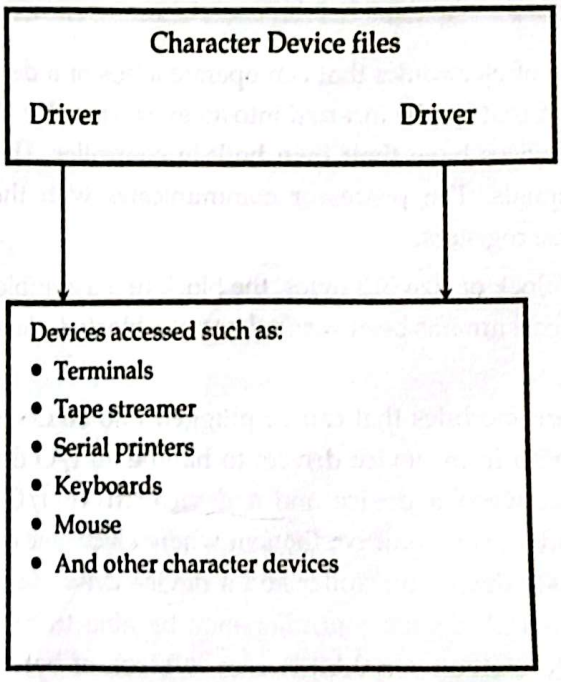


Fig 6.2: Character Device

Communication Devices

A communication device is a hardware device capable of transmitting an analog or digital signal over the telephone, other communication wire, or wirelessly. The best example of a communication device is a computer Modem, which is capable of sending and receiving a signal to allow computers to talk to other computers over the telephone. Other examples of communication devices include a NIC (network interface card), Wi-Fi devices, and access points.

For a computer to communicate with other computers they need a communication device. For example, for your computer to connect to the Internet to view this web page it needed a communication device. Without a communication device you'd have to use a net to transfer or share data between computers.

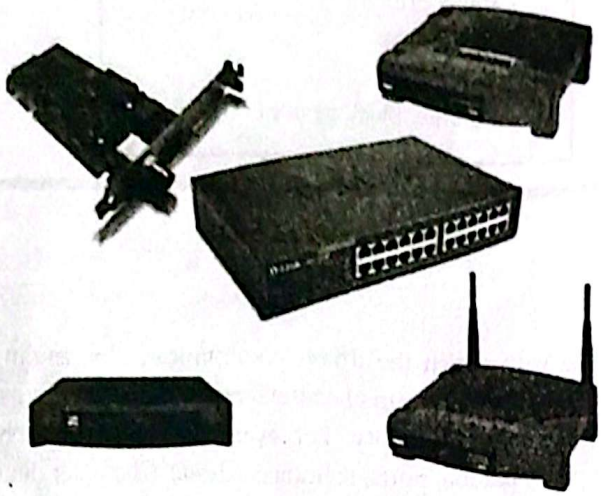


Fig 6.3: Communication devices

DEVICE CONTROLLERS

A controller is a collection of electronics that can operate a bus or a device. On PC, it often takes the form of printed circuit card that can be inserted into an expansion slot. A single controller can handle multiple devices; some devices have their own built-in controller. The controller has one or more registers for data and signals. The processor communicates with the controller by reading and writing bit patterns in these registers.

When transferring a disk block of size 512 bytes, the block first assembled bit by bit in a buffer inside the controller. After its checksum has been verified and the block declared to be error free, it can then be copied to main memory.

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 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. 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.

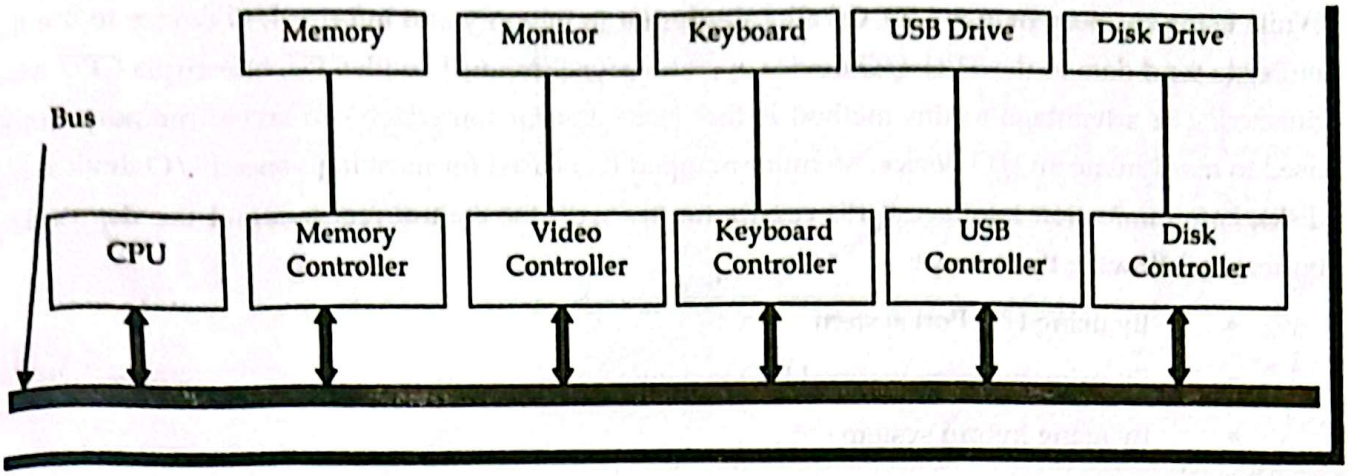


Fig 6.4: Device Controllers System

Difference between device driver and device controller

The main difference between device driver and device controller is that the device driver is software that works as the interface for the device controller to communicate with the operating system or an application program. Whereas the device controller is a hardware component that works as a bridge between the hardware device and the operating system or an application program. A device driver is specific to an operating system and it is hardware dependent. It provides interrupt handling required for necessary asynchronous time-dependent hardware interface. On the other hand, device controller is a circuit board between the device and the operating system.

COMMUNICATION OF CPU TO I/O DEVICES

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

1. Special Instruction I/O
2. Memory-mapped I/O
3. Direct memory access (DMA)

Special Instruction I/O

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

Memory Mapped I/O

Device controller has their own register and buffer for communicating with the CPU, by writing and reading these register OS perform the I/O operation. The device control registers are mapped into memory space, called memory-mapped I/O. Usually, the assigned addresses are at the top of the address space.

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.CPU communicates with the control registers and the device data buffers in following three ways:

- By using I/O Port system
- By using memory mapped I/O system
- By using hybrid system

Using I/O Port

It is also called isolated I/O system. In this approach each control register is assigned an I/O port number of an 8 or 16-bit integer. CPU reads and writes control registers by using special I/O instructions IN and OUT as shown in below. Access to them requires the use of assembly code since there is no way to execute an IN and OUT instruction in programming languages like C, C++ etc.

```
IN REG, PORT
OUT PORT, REG
```

The first instruction is used to read data from specified port or control registers and second instruction is used to write data to specified port or control register.

The address space for memory and I/O are different therefore separate instructions are needed to read/write memory and I/O control registers.

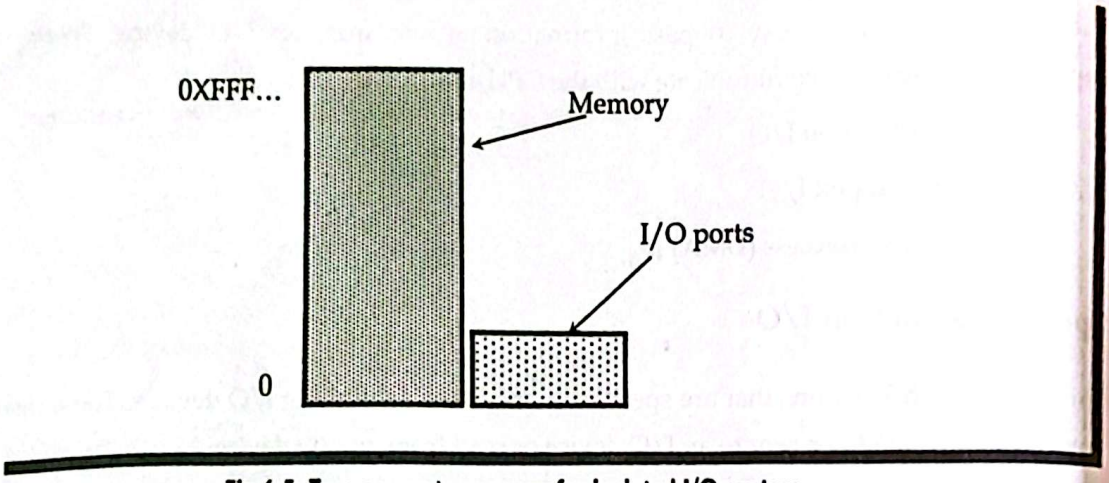


Fig 6.5: Two separate memory for isolated I/O system

Memory Mapped I/O

In this approach all control registers are mapped into the memory space. Each control register is assigned a unique address to which no memory is assigned. No special protection mechanism is needed to keep user processes from performing I/O. Here the address space for memory and I/O is same therefore instructions needed to read/write memory can be used to read/write I/O control registers.

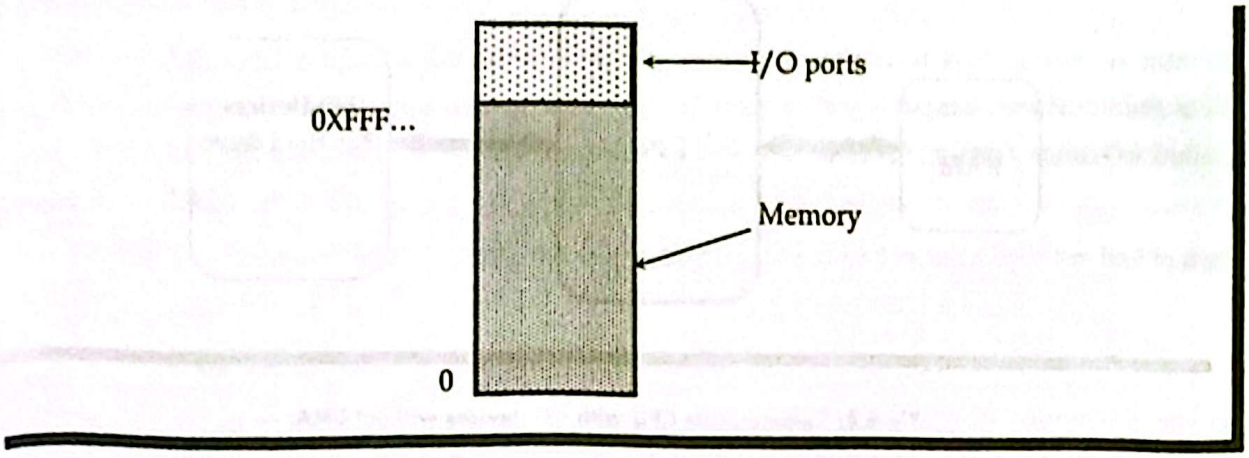


Fig 6.6: Memory mapped I/O system

By using Hybrid System

It is the combined form of both memory mapped I/O and I/O port system. It also uses two types of memory separately.

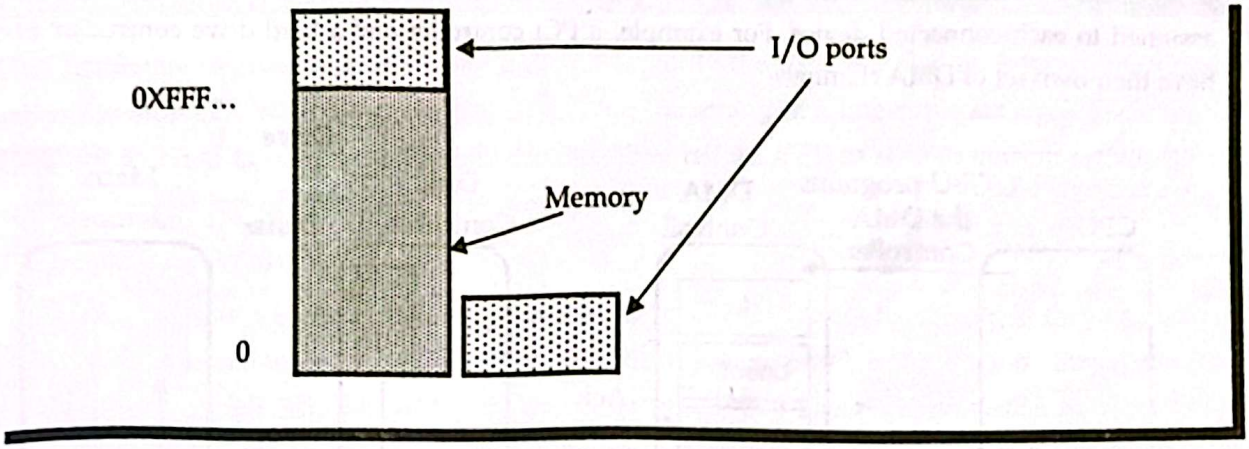


Fig 6.7: Hybrid I/O system

Advantages of memory mapped I/O

- Can be implemented in high-level languages such as C.
- No separate protection mechanism is needed.
- Every instruction that can reference the memory can also reference the control registers (in hybrid).

Disadvantages memory mapped I/O

- Adds extra complexity to both hardware and OS.
- All memory modules and all I/O devices must examine all memory references to see which one to respond to.

Direct Memory Access (DMA)

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

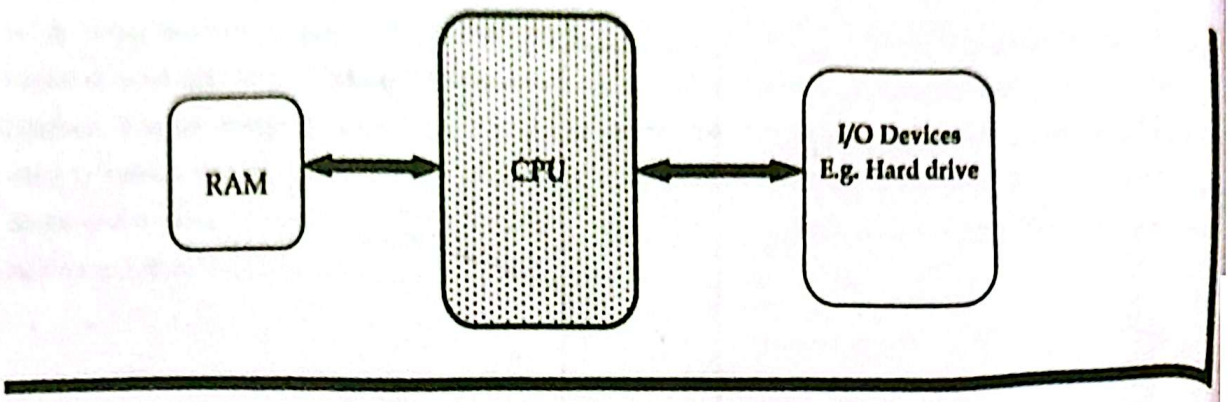


Fig 6.8: Communicate CPU with I/O devices without DMA

DMA is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU. While most data that is input or output from your computer is processed by the CPU, some data does not require processing, or can be processed by another device. In these situations, 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.

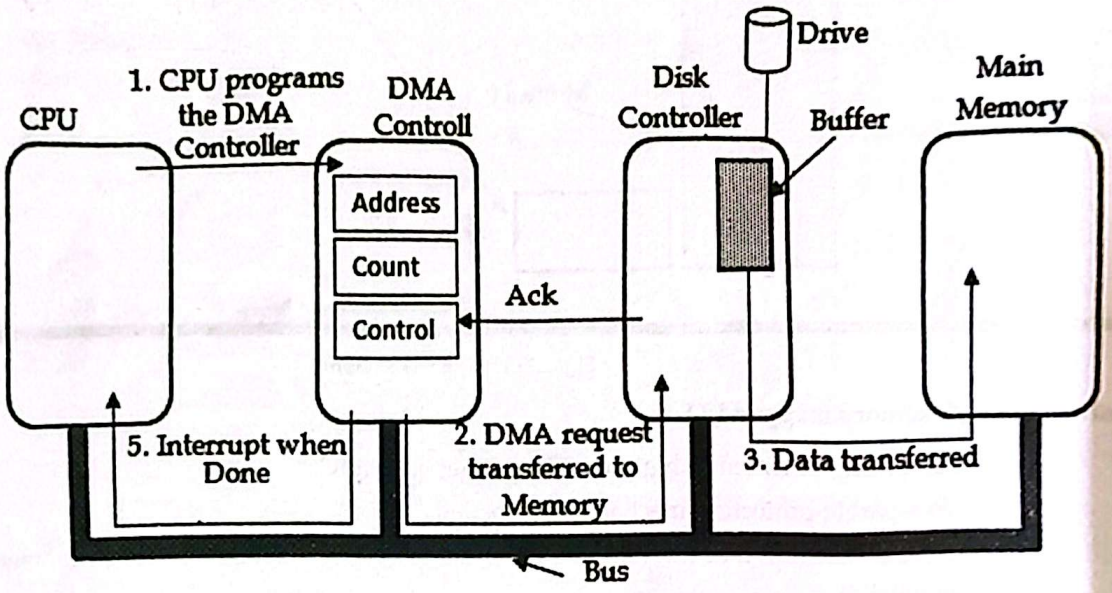


Fig 6.9: Operations of a DMA transfer

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 acknowledgment 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.

INTERRUPTS

The hardware mechanism that enables a device to notify the CPU is called an interrupt. Interrupt forced to stop CPU what it is doing and start doing something else. 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

- **Hardware Interrupts** are generated by hardware devices to signal that they need some attention from the OS. They may have just received some data or they have just completed a task which the operating system previous requested, such as transferring data between the hard drive and memory.
- **Software Interrupts** are generated by programs when they want to request a system call to be performed by the operating system.
- **Traps** are generated by the CPU itself to indicate that some error or condition occurred for which assistance from the operating system is needed.

An interrupt is a signal to the microprocessor from a device that requires attention. A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt; it saves its current state and invokes the appropriate interrupt handler using the interrupt vector. When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

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 run. 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 lines 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.

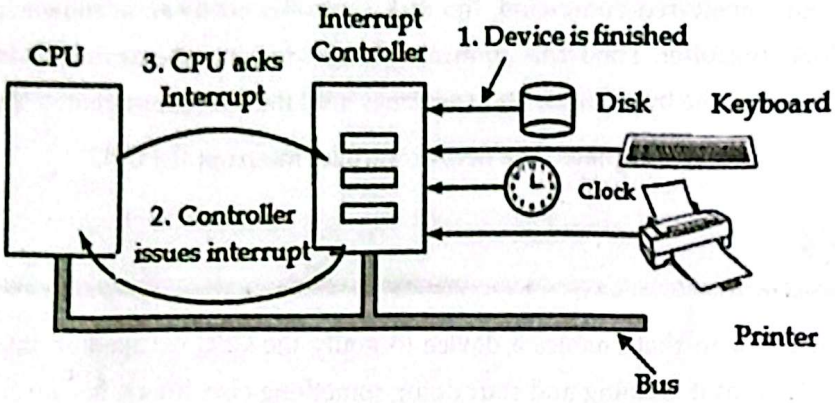


Fig 6.10: Interrupt handler

GOALS OF I/O SOFTWARE

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

a. Device independence
b. Uniform naming
c. Error Handling
d. Synchronous vs. asynchronous transfers
e. Buffering
f. Sharable and Dedicated devices

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 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 (blocking) and Asynchronous (interrupt-driven) 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.

HANDLING I/O

There are three fundamentally different ways of performing I/O operations which are listed below:

a. Programmed I/O

b. Interrupt Driven I/O and

c. I/O using DMA

1. **Programmed I/O**

This is one of the three fundamentally different ways that I/O can be performed. 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:

- The processor is executing a program and encounters an instruction relating to I/O operation.
- The processor then executes that instruction by issuing a command to the appropriate I/O module.
- The I/O module will perform the requested action based on the I/O command issued by the processor and set the appropriate bits in the I/O status register.
- The processor will periodically check the status of the I/O module until it finds that the operation is complete.

Programmed I/O Mode: Input Data Transfer

- Each input is read after first testing whether the device is ready with the input (a state reflected by a bit in a status register).
- The program waits for the ready status by repeatedly testing the status bit and till all targeted bytes is read from the input device.
- The program is in busy (non-waiting) state only after the device gets ready else in wait state.

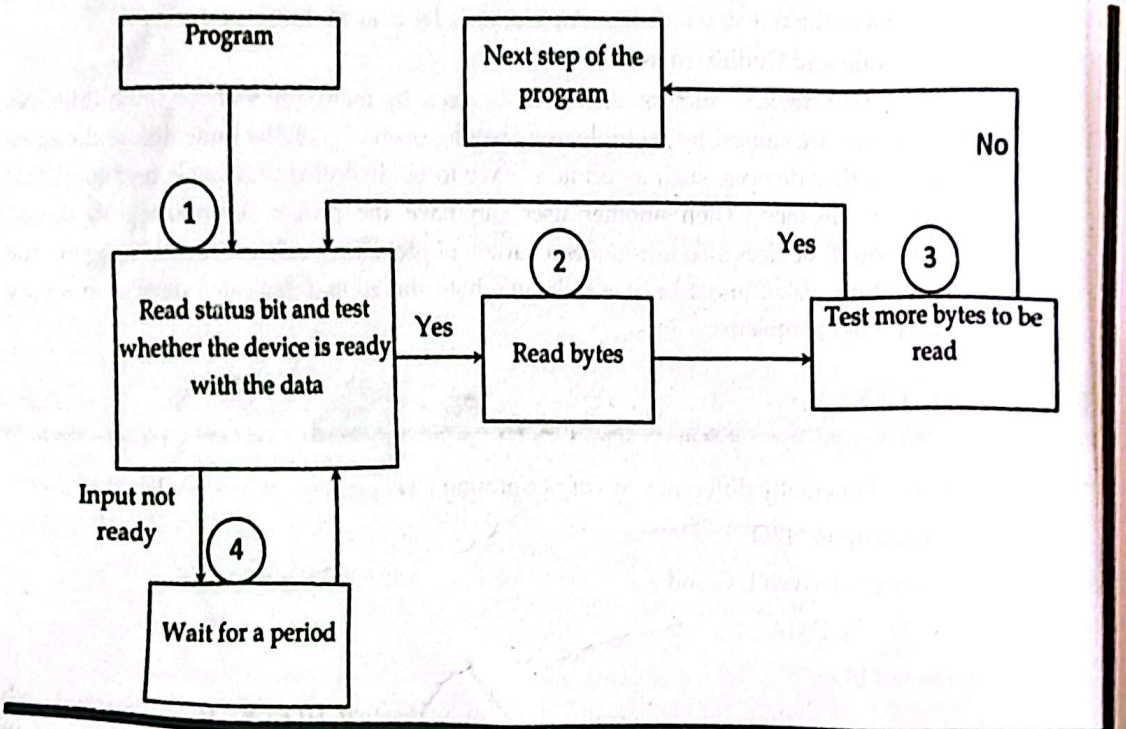


Fig 6.11: Input data transfer in programmed I/O mode

Programmed I/O Mode: Output Data Transfer

- Each output written after first testing whether the device is ready to accept the byte at its output register or output buffer is empty.
- The program waits for the ready status by repeatedly testing the status bit(s) and till all the targeted bytes are written to the device.
- The program in busy (non-waiting) state only after the device gets ready else waits state.

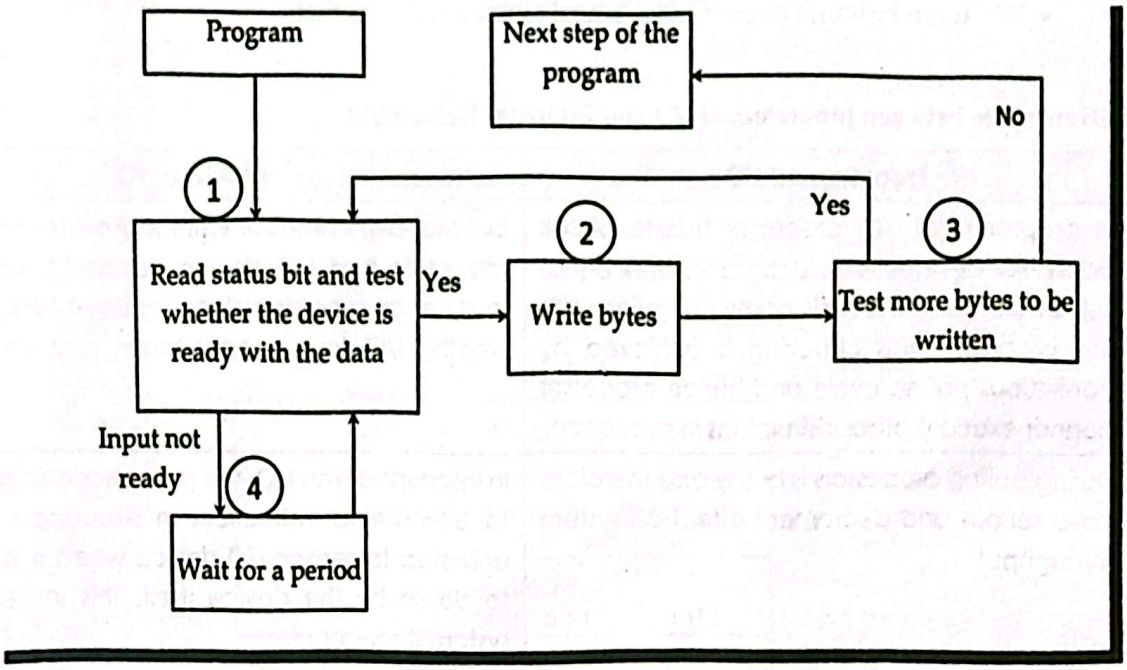


Fig 6.12: Output data transfer in programmed I/O mode

Advantages of Programmed I/O

- Simple to implement
- Very little hardware support

Disadvantages of Programmed I/O

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

b. Interrupt-Driven I/O

Interrupt driven I/O is an alternative scheme dealing with I/O. Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupts. Relative to the total interrupt system, the processors enter an interrupt service routine.

This strategy allows the CPU to carry on with its other operations until the module is ready to transfer data. When the CPU wants to communicate with a device, it issues an instruction to the appropriate I/O module, and then continues with other operations. When the device is ready, it will interrupt the CPU. The CPU can then carry out the data transfer. This also removes the need for the CPU to continually poll input devices to see if it must read any data. When an input device has data, then the appropriate I/O module can interrupt the CPU to request a data transfer.

Advantages of Interrupt-Driven I/O

- It is faster than Programmed I/O
- Efficient too

Disadvantages of Interrupt-Driven I/O

- It can be tricky to write if using a low level language.
- It can be tough to get various pieces to work well together.

Differentiate between programmed I/O and Interrupt Driven I/O

Programmed I/O	Interrupt Driven I/O
In programmed I/O, processor has to check each I/O devices in sequence and in effect ask each one if it needs communication with the processor. This checking is achieved by continuous polling cycle and hence processor cannot execute other instructions in sequence.	External asynchronous input is used to tell the processor that I/O device needs its service and hence processor does not have to check whether I/O device needs it service or not.
During polling processors is bushy and therefore have serious and decrement effect on system throughput.	In interrupt driven I/O, the processor is allowed to execute its instructions in sequence and only stop to service I/O device when it is told to do so by the device itself. This increases system throughput.
It is implemented without interrupt hardware support.	It is implemented using interrupt hardware support.
It does not depend on interrupt status.	Interrupt must be enabled to process interrupt driven I/O
It does not need initialization of stack.	It needs initialization of stack.
System throughput decreases as number of I/O devices connected in the system increases.	System throughput does not depend on the number of I/O devices connected in the system.

c. **I/O using DMA**

Although interrupt driven I/O is much more efficient than program controlled I/O, all data is still transferred through the CPU. This will be inefficient if large quantities of data are being transferred between the peripheral and memory. The transfer will be slower than necessary, and the CPU will be unable to perform any other actions while it is taking place. Many systems therefore use an additional strategy, known as direct memory access (DMA). DMA uses an additional piece of hardware- a DMA controller. The DMA controller can take over the system bus and transfer data between an I/O module and main memory without the intervention of the CPU. Whenever the CPU wants to transfer data, it tells the DMA controller the direction of the transfer, the I/O module involved, the location of the data in memory, and the size of the block of data to be transferred. It can then continue with other instructions and the DMA controller will interrupt it when the transfer is complete.

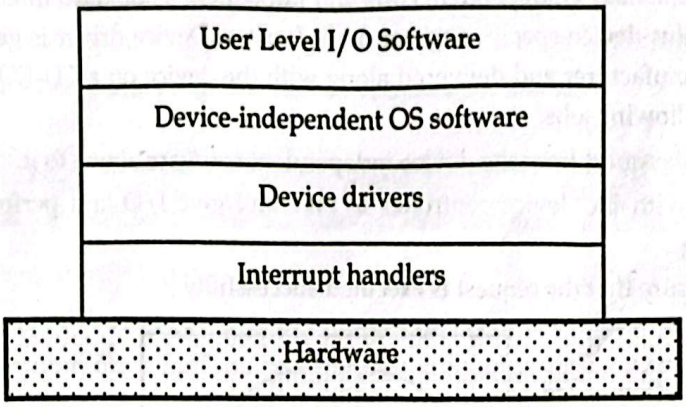
In brief, direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The process is managed by a chip known as a DMA controller (DMAC).

I/O Software Layers

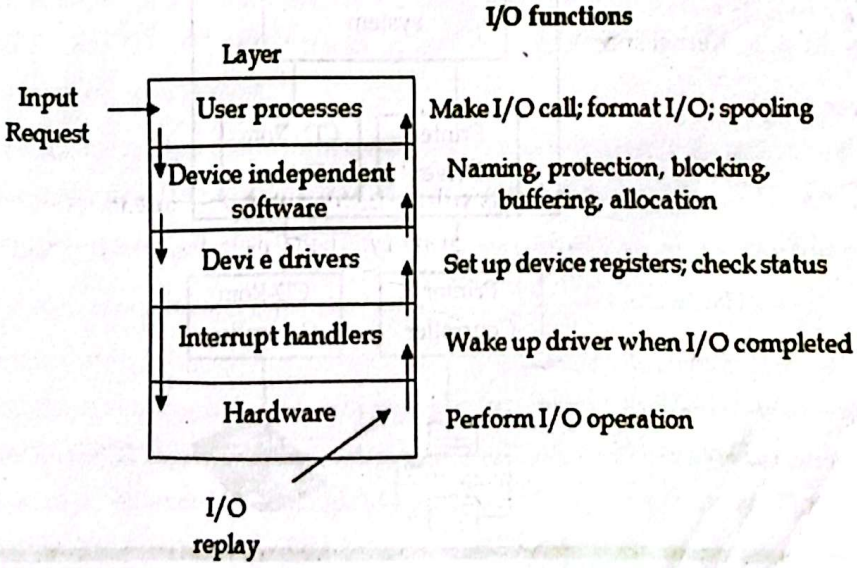
Basically, input/output software organized in the following four layers:

- 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.



Now let's describe briefly, all the four input/output software layers that are listed above.

a. **Interrupt Handlers**

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen. The interrupt mechanism accepts an address - a number that selects a specific interrupt handling function from a small set. In most architecture, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

b. **Device Drivers**

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

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. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver is generally written by the device's manufacturer and delivered along with the device on a CD-ROM:A device driver performs the following jobs:

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

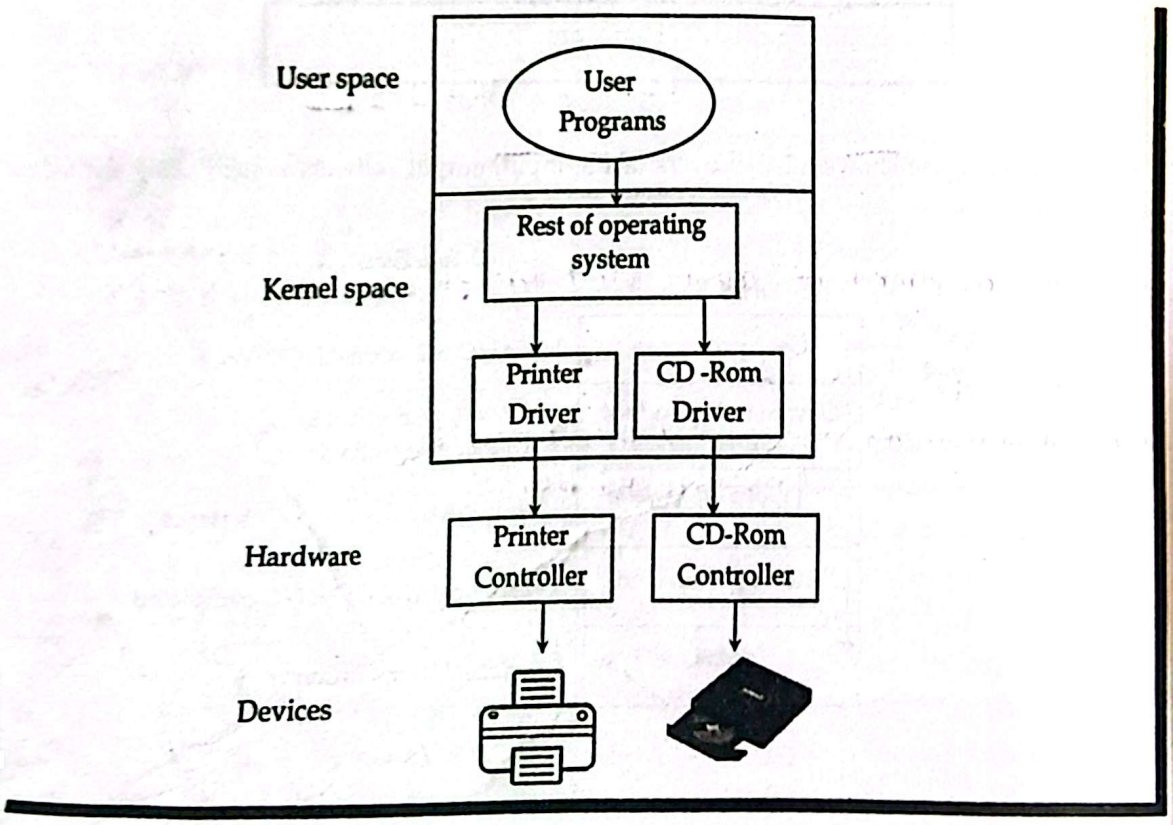


Fig 6.13: Device Drivers

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 lists 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. Each modern disk contains concentric tracks and each track is divided into multiple sectors. The disks are usually arranged as a one dimensional array of blocks, where blocks are the smallest storage unit. Blocks can also be called as sectors. For each surface of the disk, there is a read/write desk available. The same tracks on all the surfaces are known as a cylinder. Sector 0 is the first sector of the first track on the outermost cylinder. Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

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.

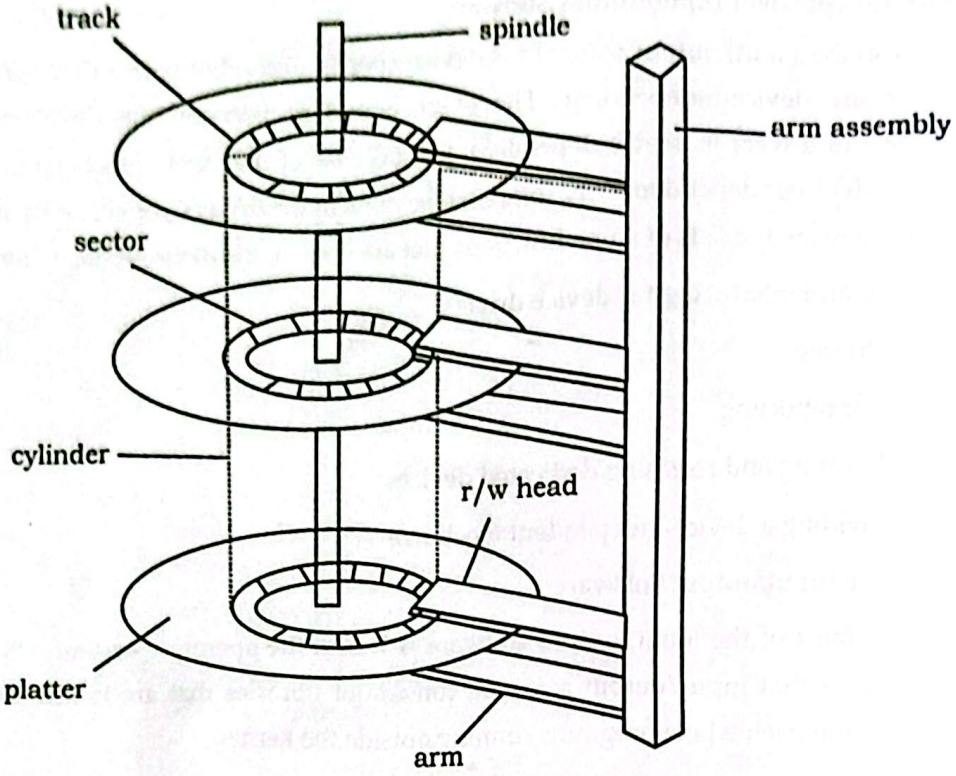


Fig 6.14: Disk structure

Disk structure key terms

- **Seek Time**

  Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

- **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 given as,

  Disk Access Time = Rotational Latency + Seek Time + 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.

Numerical Problem 1: Consider a hard disk with:

4 surfaces

64 tracks/surface

128 sectors/track

256 bytes/sector

What is the capacity of the hard disk?

a. Disk capacity = surfaces * tracks/surface * sectors/track * bytes/sector

Disk capacity = 4 * 64 * 128 * 256

Thus, Disk capacity = 8 MB

b. The disk is rotating at 3600 RPM, what is the data transfer rate?

60 sec → 3600 rotations

1 sec → 60 rotations

Data transfer rate = number of rotations per second * track capacity * number of surfaces (since 1 R-W head is used for each surface)

Data transfer rate = 60 * 128 * 256 * 4

Data transfer rate = 7.5 MB/sec

c. The disk is rotating at 3600 RPM, what is the average access time?

Since, seek time, controller time and the amount of data to be transferred is not given, we consider all the three terms as 0.

Therefore, Average Access time = Average rotational delay

Rotational latency = 60 sec =3600 rotations

1 sec →60 rotations

Rotational latency = (1/60) sec = 16.67 msec.

Average Rotational latency = (16.67)/2

= 8.33 msec.

Average Access time = 8.33 msec.

Numerical problem 2: disk has an average seek time of 5ms, a rotational speed of 15,000 rpm, and 500 sectors per track. What is the average access time to read a single sector? What is the expected time to read 500 contiguous sectors on the same track? What is the expected time to read 500 sectors scattered over the disk?

Answer: Given,

Avg. seek time, Ts = 5ms = 5×10^{-3} s

Rotational speed, r = 15000 rpm = $\frac{15000}{60}$ rps = 250 rps

Sectors per track = 500

Hence, average time to read a single sector $= Ts + \frac{1}{2r} + \frac{1}{r*500}$

$= 5x\ 10^{-3} + \frac{1}{2 \times 250} + \frac{1}{250 \times 500}$

$= 7.008 \times 10^{-3}$ s

= 7.008 ms

Hence expected time read 500 contiguous sectors on the same track = 7.008 ms + $\frac{1}{250x500} \times 499$ s

= 7.008 ms + 3.992 ms

= 11 ms

Hence, time to read 500 sectors scattered over the disk = 7.008 ms x 500

= 3.504 s Ans

Problem 3: Consider a disk with a mean seek time of 8ms, a rotational rate of 15,000 rpm, and 262,144 bytes per track. What are the access times for block sizes of 1 KB, 2 KB, and 4 KB, respectively?

Answer: Given,

Average seek time, Ts = 8ms = $8x10^{-3}$ s

Rotational rate, r = 15000 rpm = $\frac{15000}{60}$ rps = 250 rps

Number of bytes per track, N = 262144

Hence, access time for 1 KB block $= Ts + \frac{1}{2r} + \frac{1024}{r \times N}$

$= 8 \times 10^{-3} + \frac{1}{2 \times 250} + \frac{1024}{250 \times 262144}$

= 10.0156 ms

Hence, access time for 2 KB block $= Ts + \frac{1}{2r} + \frac{1024}{r \times N}$

$= 8 \times 10^{-3} + \frac{1}{2 \times 250} + \frac{1024 \times 2}{250 \times 262144}$

= 10.03125 ms

Hence, access time for 4 KB block $= Ts + \frac{1}{2r} + \frac{1024}{r \times N}$

$= 8 \times 10^{-3} + \frac{1}{2 \times 250} + \frac{1024 \times 4}{250 \times 262144}$

= 10.0625 ms

DISK SCHEDULING

A hard disk drive is a collection of plates called platters. The surface of each platter is divided into circular tracks. Furthermore, each track is divided into smaller pieces called sectors. Disk I/O is done sector by sector. A group of tracks that are positioned on top of each other form a cylinder. There is a head connected to an arm for each surface, which handles all I/O operations. For each I/O request, first head is selected. It is then moved over the destination track. The disk is then rotated to position the desired sector under the head and finally, the read/write operation is performed. There are two objectives for any disk scheduling algorithm:

- **Maximize the throughput:** the average number of requests satisfied per time unit.
- **Minimize the response time:** the average time that a request must wait before it is satisfied.

Although there are other algorithms that reduce the seek time of all requests, we will only concentrate on the following major disk scheduling algorithms:

a. First Come-First Serve (FCFS)
b. Shortest Seek Time First (SSTF)
c. Elevator (SCAN)
d. Circular SCAN (C-SCAN)
e. LOOK
f. C-LOOK

a. **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. Since no reordering of request takes place the head may move almost randomly across the surface of the disk. This policy aims to minimize response time with little regard for throughput.

Features of FCFS

- perform operations in order requested
- no reordering of work queue
- **no starvation:** every request is serviced
- poor performance

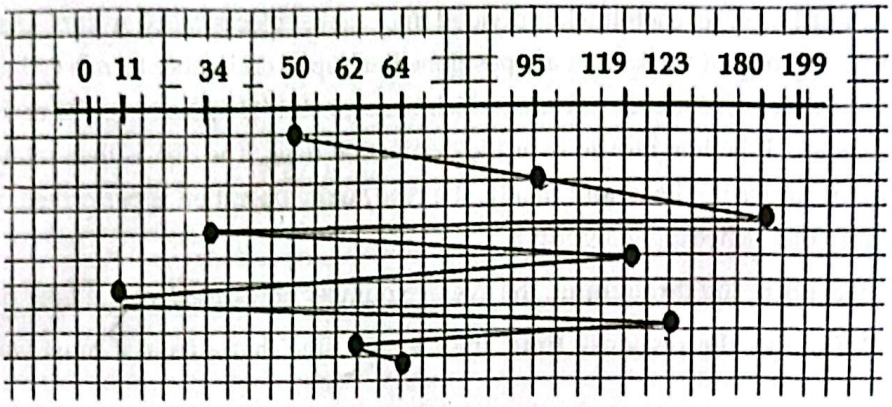
Advantages

- Every request gets a fair chance
- No indefinite postponement

Disadvantages

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

Example: Given the following queue {95, 180, 34, 119, 11, 123, 62, 64} with the Read-write head initially at the track 50 and the tail track being at 199. Then find total head movement by using FCFS disk scheduling algorithm.



Total disk movements = (95-50)+(180-95)+(180-34)+(119-34)+(119-11)+(123-11)+(123-62)+(64-62)
= 45 + 85 + 146 + 85 + 108 + 112 + 61 + 2
= 644 tracks

b. Shortest Seek Time First (SSTF)

In SSTF requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in 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. This policy will have better throughput than FCFS but a request may be delayed for a long period if many closely located requests arrive just after it.

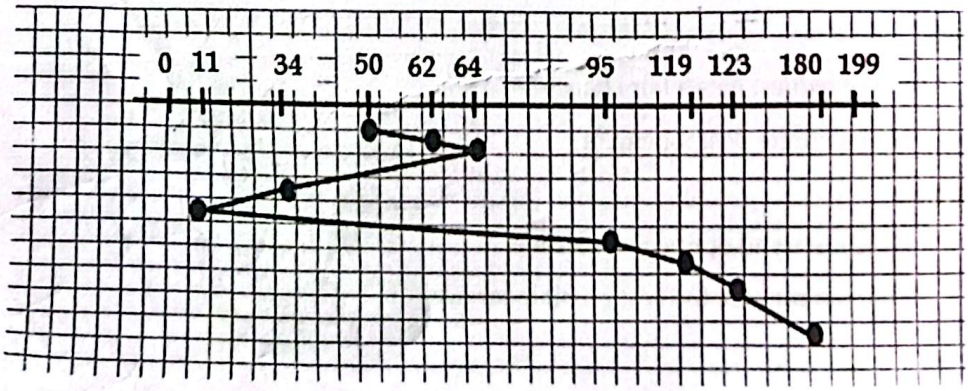
Advantages

- 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 favors only some requests

Example: Given the following queue {95, 180, 34, 119, 11, 123, 62, 64} with the Read-write head initially at the track 50 and the tail track being at 199. Then find total head movement by using SSTF disk scheduling algorithm.



Total disk movements = (62-50)+(64-62)+(64-34)+(34-11)+(95-11)+(119-95)+(123-119)+(180-123)
= 12+2+30+23+84+24+4+57
= 236 tracks

c. **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 like 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.

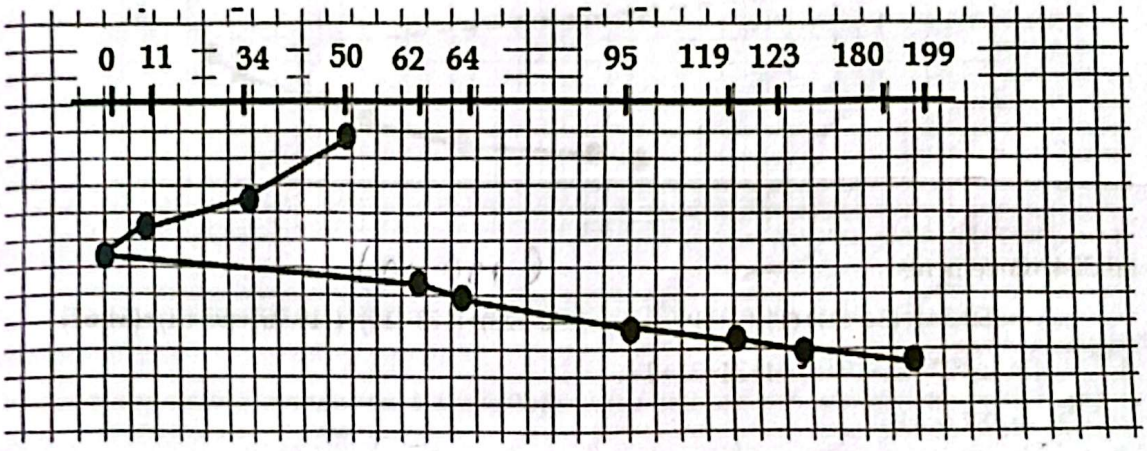
Advantages

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

Disadvantages

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

Example: Given the following queue {95, 180, 34, 119, 11, 123, 62, 64} with the Read-write head initially at the track 50 and the tail track being at 199. Then find total head movement by using SCAN disk scheduling algorithm.



Total disk movements = (50-34)+(34-11)+(11-0)+(62-0)+(64-62)+(95-64)+(119-95)+(123-119)+(180-123)
= 16+23+11+62+2+31+24+4+57
= 230 tracks.

d. **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 CSAN 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 Circular SCAN.

Advantages

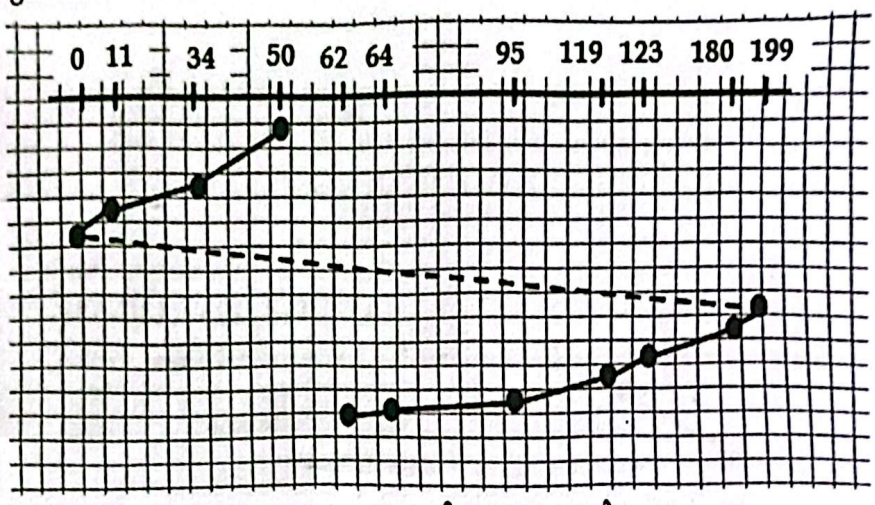
- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
- It provides uniform waiting time.
- It provides better response time.

Disadvantages

- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Example:

Given the following queue {95, 180, 34, 119, 11, 123, 62, 64} with the Read-write head initially at the track 50 and the tail track being at 199. Then find total head movement by using CSCAN disk scheduling algorithm.



Total disk movements (199 - 0)

= (50-34)+(34-11)+(11-0)+(199-180)+(180-123)+(123-119)+(119-95)+(95-64)+(64-62)

= 16+23+11+19+57+4+24+31+2

= 187 tracks

e. **LOOK**

It is similar to the SCAN disk scheduling algorithm except 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.

The main difference between SCAN Algorithm and LOOK Algorithm is:

- SCAN Algorithm scans all the cylinders of the disk starting from one end to the other end even if there are no requests at the ends.
- LOOK Algorithm scans all the cylinders of the disk starting from the first request at one end to the last request at the other end.

Advantages

- It does not cause the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head.

f. **C-LOOK**

Circular-LOOK Algorithm is an improved version of the LOOK Algorithm. Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between. After reaching the last request at the other end, head reverses its direction. It then returns to the first request at the starting end without servicing any request in between. The same process repeats.

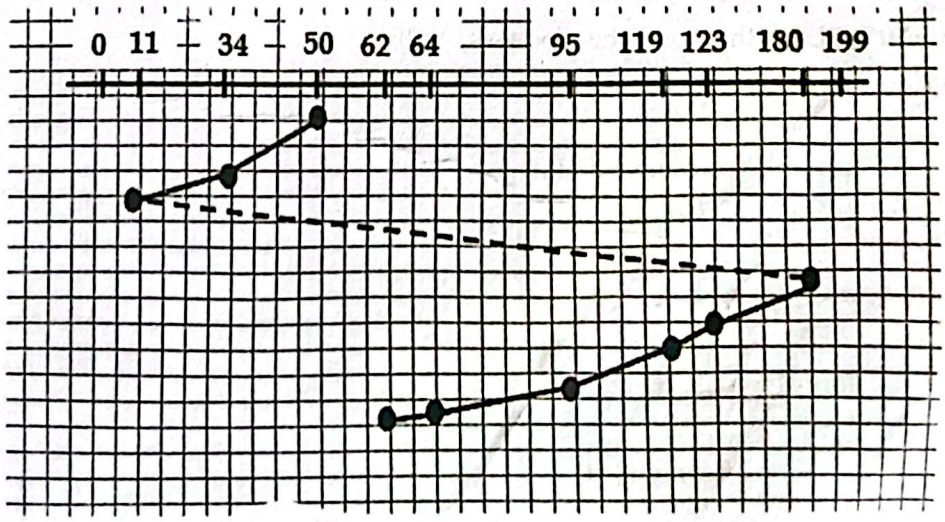
Advantages

- It does not cause the head to move till the ends of the disk when there are no requests to be serviced.
- It reduces the waiting time for the cylinders just visited by the head.
- It provides better performance as compared to LOOK Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages

- There is an overhead of finding the end requests.

Example: Given the following queue {95, 180, 34, 119, 11, 123, 62, 64} with the Read-write head initially at the track 50 and the tail track being at 199. Then find total head movement by using C-LOOK disk scheduling algorithm.



Total disk movements = (50-34)+(34-11)+(180-123)+(123-119)+(119-95)+(95-64)+(64-62)
= 16+23+57+4+24+31+2
= 157 tracks

Laboratory Works

Simulate free space management techniques and disk scheduling algorithms.

Program 1: Best Fit Algorithm

- Get no. of Processes and no. of blocks.
- After that get the size of each block and process requests.
- Then select the best memory block that can be allocated using the above definition.
- Display the processes with the blocks that are allocated to a respective process.
- Value of Fragmentation is optional to display to keep track of wasted memory.

Source code

```
#include<stdio.h>
void main()
{
        int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest=9999;
        static int barray[20], parray[20];
        printf("\n\t\t\t Memory Management Scheme - Best Fit");
        printf("\n Enter the number of blocks:");
        scanf("%d", &nb);
        printf("Enter the number of processes:");
        scanf("%d", &np);

        printf("\n Enter the size of the blocks:-\n");
        for(i=1; i<=nb; i++)
       {
                printf("Block no.%d:",i);
                scanf("%d",&b[i]);
        }
        printf("\n Enter the size of the processes :-\n");
        for(i=1;i<=np;i++)
        {
                printf("Process no.%d:",i);
                scanf("%d", &p[i]);
        }
        for(i=1; i<=np; i++)
        {
                for(j=1; j<=nb; j++)
                {
                        if(barray[j]!=1)
```

```
    {
        printf("Element[%d]:\t", count + 1);
        scanf("%d", &elements[count]);
    }
    sorting(elements, limit);
    division(elements, limit, disk_head);
    getch();
    return 0;
}
```

EXERCISE

Multiple Choice Questions

1. A _____ is a collection of electronics that can operate a port, a bus, or a device.
   a) Controller b) driver
   c) Host d) bus
2. The hardware mechanism that allows a device to notify the CPU is called _____
   a) Polling b) interrupt
   c) Driver d) controlling
3. The CPU hardware has a wire called _____ that the CPU senses after executing every instruction.
   a) Interrupt request line b) interrupt bus
   c) Interrupt receive line d) interrupt sense line
4. The data structure used for file directory is called _____
   a) Mount table b) hash table
   c) File table d) process table
5. In which type of allocation method each file occupy a set of contiguous block on the disk?
   a) Contiguous allocation b) dynamic-storage allocation
   c) Linked allocation d) indexed allocation
6. The DMA transfers are performed by a control circuit called as _____
   a) Device interface b) DMA controller
   c) Data controller d) Over-looker
7. After the completion of the DMA transfer, the processor is notified by _____
   a) Acknowledge signal b) Interrupt signal
   c) WMFC signal d) none of the mentioned

The controller uses _____ to help with the transfers when handling network interfaces.

a) Input Buffer storage b) Signal enhancers
c) Bridge circuits d) All of the mentioned

9. The set of tracks that are at one arm position make up a _____

a) Magnetic disks b) electrical disks
c) Assemblies d) cylinders

10. The time taken to move the disk arm to the desired cylinder is called the _____

a) Positioning time b) random access time
c) Seek time d) rotational latency

Subjective Questions

1. Which of the following disk scheduling techniques has a drawback of starvation?
2. Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:
   a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long)
   b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?
3. In what situations would use memory as a RAM disk be more useful than using it as a disk cache?
4. None of the disk-scheduling disciplines, except FCFS, are truly fair (starvation may occur).
   a. Explain why this assertion is true.
   b. Describe a way to modify algorithms such as SCAN to ensure fairness.
   c. Explain why fairness is an important goal in a time-sharing system.
   d. Give three or more examples of circumstances in which it is important that the operating system be unfair in serving I/O requests.
5. Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is
   {86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130}
   Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?
   a. FCFS c. SSTF d. SCAN e. LOOK
   f. C-SCAN g. C-LOOK
6. Is disk scheduling, other than FCFS scheduling, useful in a single-user environment? Explain your answer.
7. Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request's service), the variation in response time, and the effective band width. How does performance depend on the relative sizes of seek time and rotational latency?

8. Why rotational latency is usually not considered in disk scheduling? How would you modify SSTF, SCAN, and C-SCAN to include latency optimization?
9. Why is it important to balance file system I/O among the disks and controllers on a system in a multitasking environment?
10. On a disk with 1000 cylinders, numbers 0 to 999, compute the number of tracks the disk arm must move to satisfy all the requests in the disk queue. Assume the last request serviced was at track 345 and the head is moving toward track 0. The queue in FIFO order contains request for the following tracks 123, 874, 692, 475, 105, 376. Perform the computation for the following scheduling algorithms.
    a) FCFS b) SSTF
    c) SCAN d) LOOK
11. Discuss the relative advantages and disadvantages of sector sparing and sector slipping.
12. What would be the effects on cost and performance if tape storage had the same areal density as disk storage?
13. If magnetic hard disks eventually have the same cost per gigabyte as do tapes, will tapes become obsolete, or will they still be needed? Explain your answer.
14. What is a bad sector? Explain the purpose of interleave?
15. What you understand by access time. Explain different types of timings in a hard disk drive?
16. What are the impact of buffering in data transfer between I/O devices and operating system kernel?
17. Suppose that a disk drive has 5000 cylinders numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order is:
    85, 1465, 920, 1784, 948, 1510, 1025, 1745, 128
    Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms?
    a. SSTF b. SCAN c. C-SCAN d. C-LOOK
    e. LOOK f. FIFO
18. Determine the total head movement for algorithms SSTF and C-LOOK for the following requests (in track number) where the current position of head is on track number 0.
    98, 181, 23, 65, 122, 14, 72, 36, 26, 67
19. A disk has an average seek time of 5ms, a rotational speed of 15,000 rpm, and 500 sectors per track. What is the average access time to read a single sector? What is the expected time to read 500 contiguous sectors on the same track? What is the expected time to read 500 sectors scattered over the disk?
20. What are file operations? Explain about hierarchical directory system with diagrammatic examples.

ANSWERS KEY

1. (a)	2. (b)	3. (d)	4. (b)	5. (a)	6. (b)	7. (b)	8. (a)	9. (d)	10. (c)

□□□