Unit III- Input/Output

1. Write in detail about the performance of I/O interface.

- In addition to providing abstractions such as processes (and threads), address spaces, and files, an operating system also controls all the computer's I/O (Input/Output) devices. It must issue commands to the devices, catch interrupts, and handle errors. It should also provide an interface between the devices and the rest of the system that is simple and easy to use. To the extent possible, the interface should be the same for all devices (device independence). The I/O code represents a significant fraction of the total operating system.

PRINCIPLES OF I/O HARDWARE

- Different people look at I/O hardware in different ways. Electrical engineers look at it in terms of chips, wires, power supplies, motors, and all the other physical components that make up the hardware. Programmers look at the interface presented to the software - the commands the hardware accepts, the functions it carries out, and the errors that can be reported back. In this blog we are concerned with programming I/O devices, not designing, building, or maintaining them, so our interest will be restricted to how the hardware is programmed, not how it works inside.

I/O Devices

- I/O devices can be roughly divided into two categories: block devices and character devices. A block device is one that stores information in fixed-size blocks, each one with its own address. Common block sizes range from 512 bytes to 32,768 bytes. All transfers are in units of one or more entire (consecutive) blocks. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. Hard disks, CD-ROMs, and USB sticks are common block devices.
- The other type of I/O device is the character device. A character device delivers or accepts a stream of characters, without regard to any block structure. It is not addressable and does not have any seek operation. Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices that are not disk-like can be seen as character devices.
- I/0 devices cover a huge range in speeds, which puts considerable pressure on the software to perform well over many orders of magnitude in data rates. Figure 1 shows the data rates of some common devices. Most of these devices tend to get faster as time goes on.

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
SATA disk drive	300 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec

Figure 1. Some typical device, network, and bus data rates.

Device Controllers

- I/O units usually consist of a mechanical component and an electronic component. It is often possible to separate the two portions to provide a more modular and general design. The electronic component is called the device controller or adapter. On personal computers, it often takes the form of a chip on the parentboard or a printed circuit card that can be inserted into a (PCI) expansion slot. The mechanical component is the device itself.

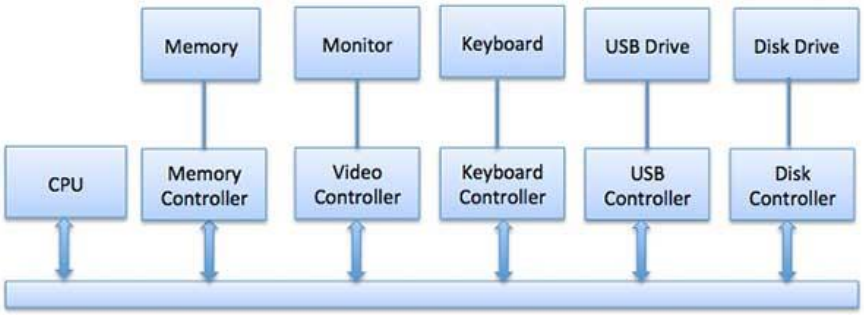
2. Which are two categories of I/O devices?

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

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.

- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

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



Synchronous vs asynchronous I/O

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

Communication to I/O Devices

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

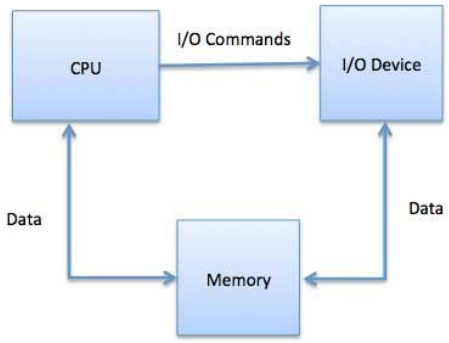
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

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

Memory-mapped I/O

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



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

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

Polling vs Interrupts I/O

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

Polling I/O

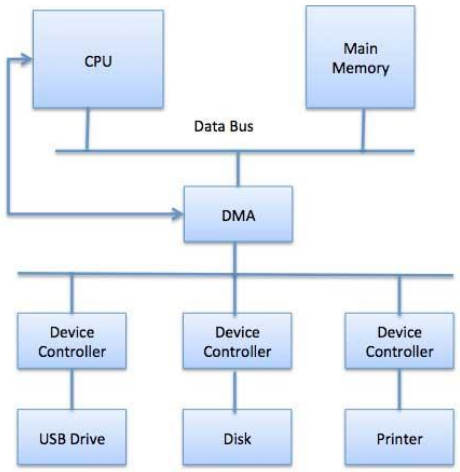
Polling is the simplest way for an I/O device to communicate with the processor the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor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

Interrupts I/O

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

3. Explain direct memory access.

- Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.
- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.
- Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows –

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

- In order to explain how DMA works, let us first consider how disk reads occur when DMA is not used. First the disk controller reads the block (one or more sectors) from the drive serially, bit by bit, until the entire block is in the controller's internal buffer. Next, it computes the checksum to verify that no read errors have occurred. Then the controller causes an interrupt. When the operating system starts running, it can read the disk block from the controller's buffer a byte or a word at a time by executing a loop, with each iteration reading one byte or word from a controller device register and storing it in main memory.
- When DMA is used, the procedure is different. First the CPU programs the DMA controller by setting its registers so it knows what to transfer where.

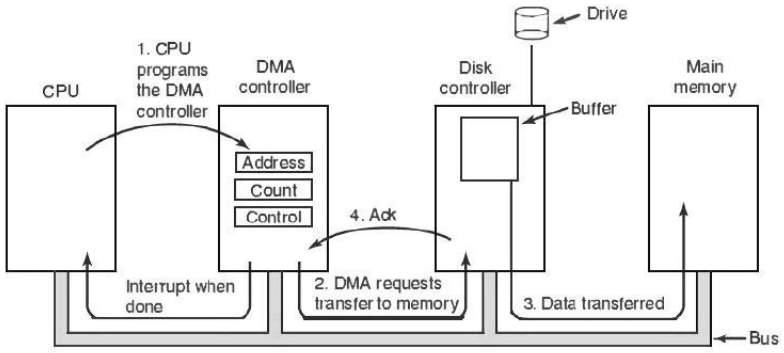


Figure 1. Operation of a DMA transfer.

- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum. When valid data are in the disk controller's buffer, DMA can begin.

- The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller (step 2). This read request looks like any other read request, and the disk controller does not know or care whether it came from the CPU or from a DMA controller. Usually, the memory address to write to is on the bus' address lines so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle (step 3). When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus (step 4). The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0. At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete. When the operating system starts up, it does not have to copy the disk block to memory; it is already there.

4. What is interrupt and give its types?

- When a Process is executed by the CPU and when a user Request for another Process then this will create disturbance for the Running Process. This is also called as the **Interrupt**.

- Interrupts can be generated by User, Some Error Conditions and also by Software's and the hardware's. But CPU will handle all the Interrupts very carefully because when Interrupts are generated then the CPU must handle all the Interrupts Very carefully means the CPU will also Provide Response to the Various Interrupts those are generated. So that When an interrupt has Occurred then the CPU will handle by using the Fetch, decode and Execute Operations.

- In a typical personal computer system, the interrupt structure is as illustrated in Figure 1. At the hardware level, interrupts work as follows. When an I/O device has finished the work given to it, it causes an interrupt (assuming that interrupts have been enabled by the operating system). It does this by asserting a signal on a bus line that it has been assigned. This signal is detected by the interrupt controller chip on the parentboard, which then decides what to do.

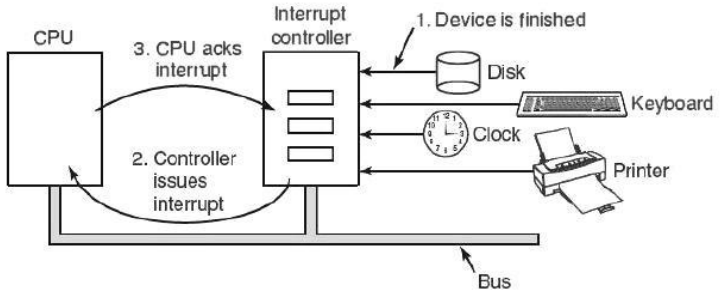


Figure 1. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

Types of Interrupts and How to Handle Interrupts

- In early years of computing processor has to wait for the signal for processing, so processor has to check each and every hardware and software program in the system if it has any signal to process. This method of checking the signal in the system for p040rocessing is called polling method. In this method the problem processor has to waste number of clock cycles just for checking the signal in the system, by this processor will become busy unnecessarily. If any signal came for the process, processor will take some time to process the signal due to the polling process in action. So system performance also will be degraded and response time of the system will also decrease.

- So to over this problem engineers introduced a new mechanism, in this mechanism processor will not check for any signal from hardware or software but instead hardware/software will only send the signal to the processor for processing. The signal from hardware or software should have highest priority because processor should leave the current process and process the signal of hardware or software. This mechanism of processing the signal is called interrupt of the system.

What is an Interrupt?

Interrupt is a signal which has highest priority from hardware or software which processor should process its signal immediately.

Types of Interrupts:

Although interrupts have highest priority than other signals, there are many type of interrupts but basic type of interrupts are:

Hardware Interrupts: If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are:

- **Maskable Interrupt:** The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.
- **Non Maskable Interrupt:** The hardware which cannot be delayed and should process by the processor immediately.

Software Interrupts: Software interrupt can also divided in to two types. They are:

- **Normal Interrupts:** the interrupts which are caused by the software instructions are called software instructions.
- **Exception:** unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

Classification of Interrupts According to Periodicity of Occurrence:

- **Periodic Interrupt:** If the interrupts occurred at fixed interval in timeline then that interrupts are called periodic interrupts
- **Aperiodic Interrupt:** If the occurrence of interrupt cannot be predicted then that interrupt is called aperiodic interrupt.

- **Classification of Interrupts According to the Temporal Relationship with System Clock:**
- **Synchronous Interrupt:** The source of interrupt is in phase to the system clock is called synchronous interrupt. In other words interrupts which are dependent on the system clock. Example: timer service that uses the system clock.
- **Asynchronous Interrupts:** If the interrupts are independent or not in phase to the system clock is called asynchronous interrupt.

Precise and Imprecise Interrupts

Precise Interrupt (Precise Exception)

- An interrupt or exception is called precise if the saved processor state corresponds with the sequential model of program execution where one instruction execution ends before the next begins.
- Precise exception means that all instructions before the faulting instruction are committed and those after it can be restarted from scratch.
- If an interrupt occurred, all instructions that are in program order before the interrupt signaling instruction are committed, and all later instructions are removed.
- Depending on the architecture and the type of exception, the faulting instruction should be committed or removed without any lasting effect.

Imprecise exception

- A synchronous exception that does not adhere to the precise exception model. In the Cell Broadband Engine, single-precision floating-point operations generate imprecise exceptions.

5. Explain goals of the I/O software.

Goals of the I/O Software

- A key concept in the design of I/O software is known as device independence. What it means is that it should be possible to write programs that can access any I/O device without having to specify the device in advance. For instance, a program that reads a file as input should be able to read a file on a hard disk, a CD-ROM, a DVD, or a USB stick without having to change the program for each different device.
- Closely related to device independence is the goal of uniform naming. The name of a file or a device should 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.
- Another important issue for I/O software is error handling. Generally, errors should be handled as close to the hardware as possible. 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. Many errors are transient, such as read errors caused by specks of dust on the read head, and will often go away if the operation is repeated. Only if the lower layers are not able to deal with the problem should the upper layers be told about it. In many cases, error recovery can be done transparently at a low level without the upper levels even knowing about the error.

- Still another key issue is that of synchronous (blocking) versus asynchronous (interrupt-driven) transfers. Most physical I/O is asynchronous - the CPU starts the transfer and goes off to do something else until the interrupt arrives.
- Another issue for the I/O software is buffering. Often data that come off a device cannot be stored directly in its final destination.

Programmed I/O

- There are three basically different ways that I/O can be performed. In this section we will consider the first one (programmed I/O). In the next two sections we will look at the others (interrupt-driven I/O and I/O using DMA). The simplest form of I/O is to have the CPU do all the work. This method is called programmed I/O.

Interrupt-Driven I/O

- Now let us look at the case of printing on a printer that does not buffer characters but prints each one as it arrives. If the printer can print, say 100 characters/sec, each character takes 10 msec to print. This means that after every character is written to the printer's data register, the CPU will sit in an idle loop for 10 msec waiting to be allowed to output the next character. This is more than enough time to do a context switch and run some other process for the 10 msec that would otherwise be wasted.

I/O Using DMA

- An obvious disadvantage of interrupt-driven I/O is that an interrupt happens on every character. Interrupts take time, so this scheme wastes a certain amount of CPU time. A solution is to use DMA. Here the idea is to let the DMA controller feed the characters to the printer one at time, without the CPU being bothered. In essence, DMA is programmed I/O, only with the DMA controller doing all the work, instead of the main CPU. This strategy requires special hardware (the DMA controller) but frees up the CPU during the I/O to do other work.

6. Explain the types of Interrupts Handlers.

- I/O software is normally organized in four layers, as illustrated in Figure 1. Each layer has a well-defined function to perform and a well-defined interface to the adjacent layers. The functionality and interfaces differ from system to system, so the discussion that follows, which examines all the layers starting at the bottom, is not specific to one machine.

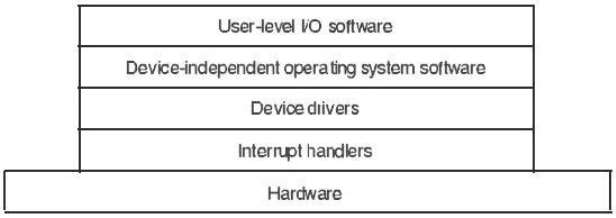


Figure 1. Layers of the I/O software system.

Interrupt Handlers

- As programmed I/O is occasionally useful, for most I/O, interrupts are an unpleasant fact of life and cannot be avoided. They should be hidden away, deep in the bowels of the operating system, so that as little of the operating system as possible knows about them. The best way to hide them is to have the driver starting an I/O operation block until the I/O has completed and the interrupt occurs. The driver can block itself by doing a down on a semaphore, a wait on a condition variable, a receive on a message, or something similar, for instance.

- When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt. Then it can unblock the driver that started it. In some cases it will just complete up on a semaphore. In others it will do a signal on a condition variable in a monitor. In still others, it will send a message to the blocked driver. In all cases the net effect of the interrupt will be that a driver that was previously blocked will now be able to run. This model works best if drivers are structured as kernel processes, with their own states, stacks, and program counters.

- Certainly, reality is not quite so simple. Processing an interrupt is not just a matter of taking the interrupt, doing an up on some semaphore, and then executing an IRET instruction to return from the interrupt to the previous process. There is a great deal more work involved for the operating system. We will now give an outline of this work as a series of steps that must be performed in software after the hardware interrupt has completed. It should be noted that the details are very system dependent, so some of the steps listed below may not be required on a specific machine and steps not listed may be needed. Also, the steps that do occur may be in a different order on some machines.

1. Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
2. Set up a context for the interrupt service procedure. Doing this may involve setting up the TLB, MMU and a page table.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, re-enable interrupts.

5. Copy the registers from where they were saved (possibly some stack) to the process table.
6. Run the interrupt service procedure. It will extract information from the interrupting device controller's registers.
7. Choose which process to run next. If the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now.
8. Set up the MMU context for the process to run next. Some TLB setup may also be required.
9. Load the new process' registers, including its PSW.
10. Start running the new process.

7. What is mean by program status word?

8. Explain Device Drivers and Stable Storage.

- The number of device registers and the nature of the commands vary completely from device to device. For instance, a mouse driver has to accept information from the mouse telling how far it has moved and which buttons are currently depressed. In contrast, a disk driver may have to know all about sectors, tracks, cylinders, heads, arm motion, motor drives, head settling times, and all the other mechanics of making the disk work correctly. Apparently, these drivers will be very different.

- As a consequence, each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the device driver, is usually written by the device's manufacturer and delivered along with the device. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

- Each device driver usually handles one device type, or at most, one class of closely related devices. For instance, a SCSI disk driver can usually handle multiple SCSI disks of different sizes and different speeds, and perhaps a SCSI CD-ROM as well. However, a mouse and joystick are so different that different drivers are usually required. However, there is no technical restriction on having one device driver control multiple unrelated devices. It is just not a good idea.

- Since the designers of every operating system know that pieces of code (drivers) written by outsiders will be installed in it, it needs to have an architecture that allows such installation. This means having a well-defined model of what a driver does and how it interacts with the rest of the operating system. Device drivers are generally positioned below the rest of the operating system, as is shown in Figure 1.

- Operating systems generally classify drivers into one of a small number of categories. The most common categories are the block devices, such as disks, which contain several

data blocks that can be addressed independently, and the character devices, such as keyboards and printers, which generate or accept a stream of characters.

- In some systems, the operating system is a single binary program that contains all of the drivers that it will need compiled into it. This scheme was the norm for years with UNIX systems because they were run by computer centers and I/O devices rarely changed. If a new device was added, the system administrator simply recompiled the kernel with the new driver to build a new binary.

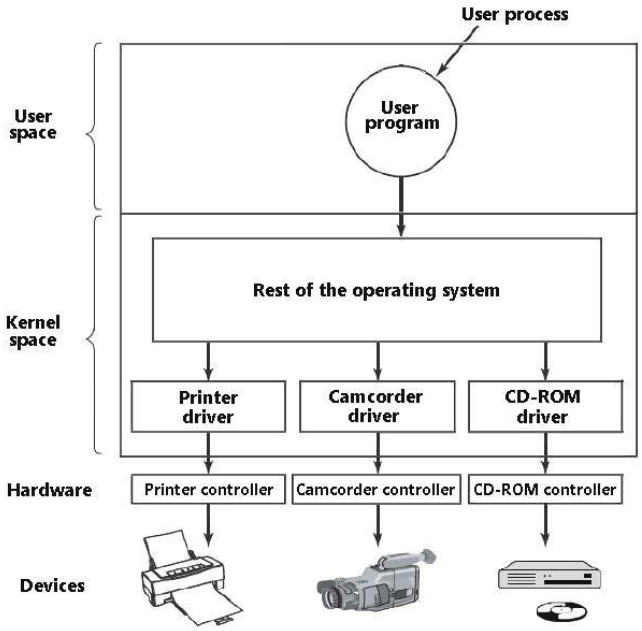


Figure 1. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

- A device driver has numerous functions. The most obvious one is to accept abstract read and write requests from the device- independent software above it and see that they are carried out.

- Next the driver may check if the device is currently in use. If it is, the request will be queued for later processing.

- In a hot pluggable system, devices can be added or removed while the computer is running. Consequently, while a driver is busy reading from some device, the system may inform it that the user has suddenly removed that device from the system. Not only must the current I/O transfer be aborted without damaging any kernel data structures, but any pending requests for the now-vanished device must also be gracefully removed from the system and their callers given the bad news. Moreover, the unexpected addition of new devices may cause the kernel to juggle resources (e.g., interrupt request lines), taking old ones away from the driver and giving it new ones in their place.

9. Explain Device Independent I/O software and User Space I/O software.

Device-Independent I/O Software

- Though some of the I/O software is device specific, other parts of it are device independent. The exact boundary between the drivers and the device-independent software is system (and device) dependent, because some functions that could be done in a device-independent way may in fact be done in the drivers, for efficiency or other reasons. The functions shown in Figure 1 are usually done in the device-independent software.

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

Figure 1. Functions of the device-independent I/O software.

- The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software.

Uniform Interfacing for Device Drivers

- A main issue in an operating system is how to make all I/O devices and drivers look more or less the same. If disks, printers, keyboards, and so on, are all interfaced in different ways, every time a new device comes along, the operating system must be modified for the new device. Having to hack on the operating system for each new device is not a good idea.

- In Figure 2(a) we show a situation in which each device driver has a different interface to the operating system. What this means is that the driver functions available for the system to call differ from driver to driver. It might also mean that the kernel functions that the driver needs also differ from driver to driver. Taken together, it means that interfacing each new driver needs a lot of new programming effort.

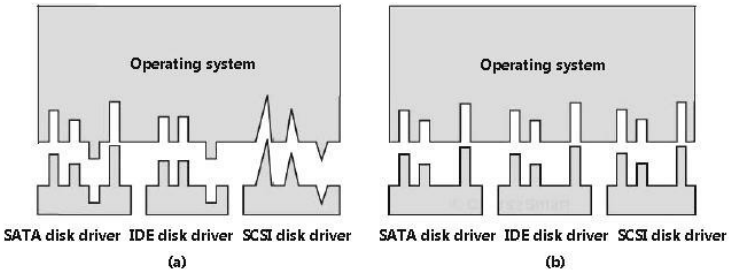


Figure 2. (a) Without a standard driver interface. (b) With a standard driver interface.

- On the contrary, in Figure 2(b), we illustrate a different design in which all drivers have the same interface. Now it becomes much easier to plug in a new driver, providing it conforms to the driver interface. It also means that driver writers know what is expected of them. In fact, not all devices are absolutely identical, but generally there are only a small number of device types and even these are usually almost the same.

Buffering

- Buffering is also an issue, both for block and character devices, for a variety of reasons. To see one of them, consider a process that wants to read data from a modem. One possible strategy for dealing with the incoming characters is to have the user process do a read system call and block waiting for one character. Each arriving character causes an interrupt.

- Buffering is a broadly used technique, but it has a downside as well. If data get buffered too many times, performance suffers. Consider, for example, the network of Figure 4. Here a user does a system call to write to the network. The kernel copies the packet to a kernel buffer to allow the user to proceed immediately (step 1). At this point the user program can reuse the buffer.

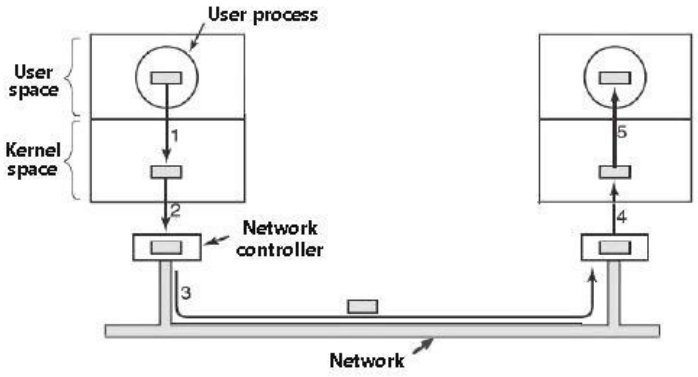


Figure 4. Networking may involve many copies of a packet.

Error Reporting

- Errors are far more common in the context of I/O than in other contexts. When they take place, the operating system must handle them as best it can. Many errors are device-specific and must be handled by the appropriate driver, but the framework for error handling is device independent. One class of I/O errors is programming errors. These occur when a process asks for something impossible, such as writing to an input device (keyboard, scanner, mouse, etc.) or reading from an output device (printer, plotter, etc.).

Allocating and Releasing Dedicated Devices

- Some devices, such as CD-ROM recorders, can be used only by a single process at any given moment. It is up to the operating system to examine requests for device usage and accept or reject them, depending on whether the requested device is available or not.

Device-Independent Block Size

- Different disks may have different sector sizes. It is up to the device-independent software to hide this fact and provide a uniform block size to higher layers, for instance, by treating various sectors as a single logical block.

User-Space I/O Software

- Though most of the I/O software is within the operating system, a small portion of it comprises libraries linked together with user programs, and even whole programs running outside the kernel. System calls, including the I/O system calls, are usually made by library procedures.

- One example from C is printf, which takes a format string and possibly some variables as input, builds an ASCII string, and then calls write to output the string. As an example of printf, look at the statement.

```
printf("The square of %3d is  %6d\n", i, i*i);
```

- It formats a string consisting of the 14-character string "The square of" followed by the value i as a 3-character string, then the 4-character string "is", then i2 as six characters, and lastly a line feed.
- An example of a similar procedure for input is scanf which reads input and stores it into variables explained in a format string using the same syntax as printf. The standard I/O library includes a number of procedures that involve I/O and all run as part of user programs.

- Figure 1 summarizes the I/O system, showing all the layers and the principal functions of each layer. Starting at the bottom, the layers are the hardware, interrupt handlers, device drivers, device-independent software, and lastly the user processes.

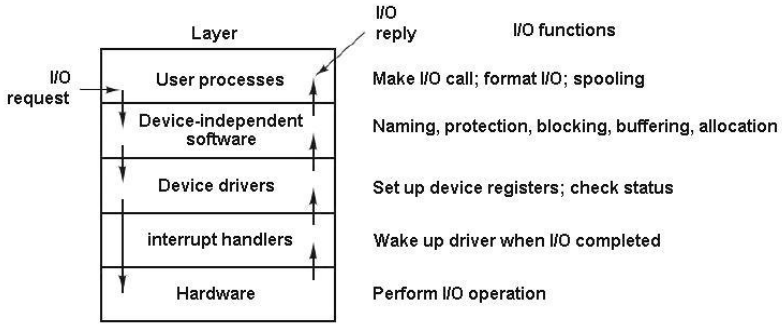


Figure 1. Layers of the I/O system and the main functions of each layer.

10. Explain Error Handling.

- Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized programs, called error handlers, are available for some applications. The best programs of this type forestall errors if possible, recover from them when they occur without terminating the application, or (if all else fails) gracefully terminate an affected application and save the error information to a log file.

- In programming, a development error is one that can be prevented. Such an error can occur in syntax or logic. Syntax errors, which are typographical mistakes or improper use of special characters, are handled by rigorous proofreading. Logic errors, also called bugs, occur when executed code does not produce the expected or desired result. Logic errors are best handled by meticulous program debugging. This can be an ongoing process that involves, in addition to the traditional debugging routine, beta testing prior to official release and customer feedback after official release.

- A run-time error takes place during the execution of a program, and usually happens because of adverse system parameters or invalid input data. An example is the lack of sufficient memory to run an application or a memory conflict with another program. On the Internet, run-time errors can result from electrical noise, various forms of malware or an exceptionally heavy demand on a server. Run-time errors can be resolved, or their impact minimized, by the use of error handler programs, by vigilance on the part of network and server administrators, and by reasonable security countermeasures on the part of Internet users.

Interrupts and Error Handling

- Interrupts can be generated by hardware or software. Computers and operating systems vary in the types of interrupts that they support and the mechanisms by which they respond to them. In recent Microsoft Windows operating systems an interrupt management scheme hides these processing differences and the hardware abstraction layer (HAL) provides a virtual interrupt mechanism to the kernel.
- The kernel's trap handler is responsible for handling of all interrupts, exceptions (ie: errors), system service calls and virtual memory management. The major difference between interrupts and exceptions is that interrupts occur asynchronously (eg: when a hardware peripheral device needs some CPU time), whereas exceptions occur as a part of standard application execution (eg: when the result of a calculation is too large to be stored). Exceptions can usually be reproduced under test conditions, but interrupts involve timing relationships that are difficult to replicate.
- Some interrupts have higher priority than others, eg: timer (system clock) interrupts are critical to many Windows functions, including pre-emptive multitasking, so they are allocated a higher priority than interrupts caused by peripheral devices such as printers and modems.
- When an interrupt occurs, it is handled by an interrupt service routine (ISR), sometimes known as an error handler or exception handler. The kernel includes ISRs for many system interrupts such the system clock, power failure and process scheduling. Other ISRs are provided by the device drivers that manage peripheral hardware devices such as network adapters, keyboards and disk drives.
- In some operating systems, further interrupts are blocked or masked while an interrupt is being processed. In recent Windows systems, low-priority interrupts are masked while a higher priority interrupt is being processed, but a high priority interrupt will take precedence over a lower priority interrupt.

11.Explain any two Disk Scheduling Algorithms.

Disk Scheduling Algorithms

1. **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.

Advantages:

- Every request gets a fair chance
- No indefinite postponement

Disadvantages:

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

2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in 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.

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

3. **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

4. **CSCAN**: 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 C-SCAN (Circular SCAN).

Advantages:

- Provides more uniform wait time compared to SCAN

5. **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.

6. **CLOOK:** As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm inspite 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.

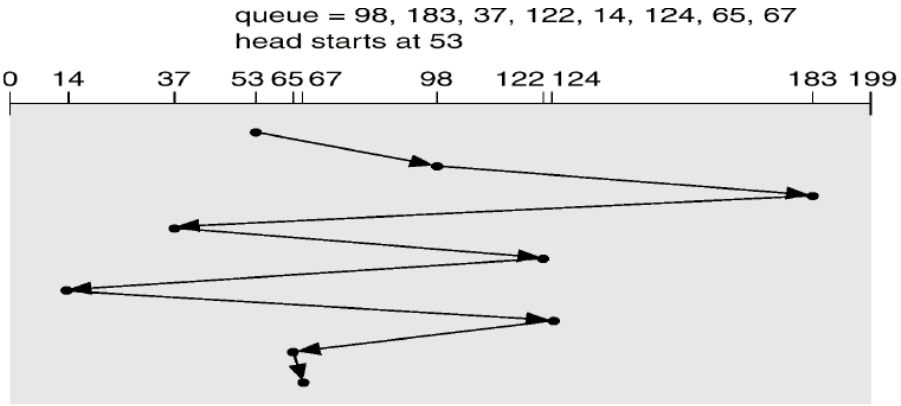
The time required to read or write a disk block determined by 3 factors.

- Seek time: The time to move the arm to the proper cylinder.
- Rotational delay: the time for the proper sector to rotate under the head.
- Actual data transfer time

Among these, the Seek time dominates. Error checking is done by controllers.
We illustrate them with a request queue (0199).

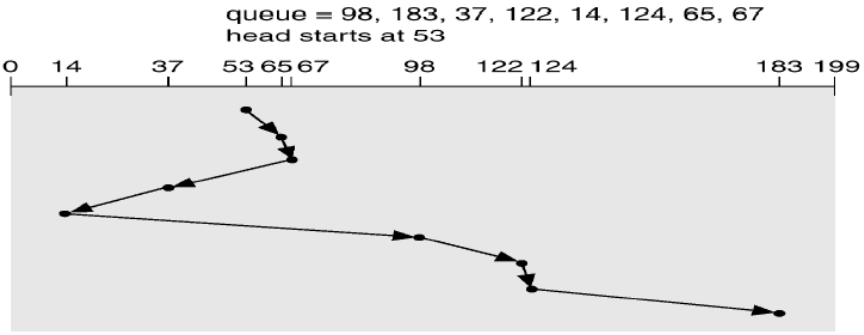
98, 183, 37, 122, 14, 124, 65, 67 Head pointer 53

FCFS:
When the current request is finished, the disk driver has to decide which request to handle next. Using FCFS, it would go to next cylinder 1, then 36 and so on.



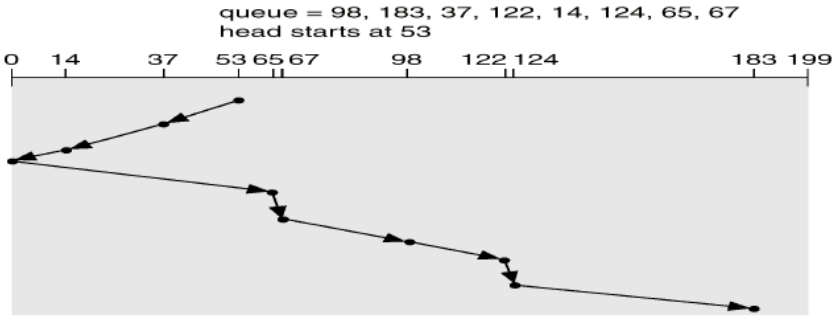
SSF (Shortest Seek First):

- Also called SSTF (shortest seek time first). Selects the request with the minimum seek time from the current head position. SSF scheduling is a form of SJF scheduling; may cause starvation of some requests.



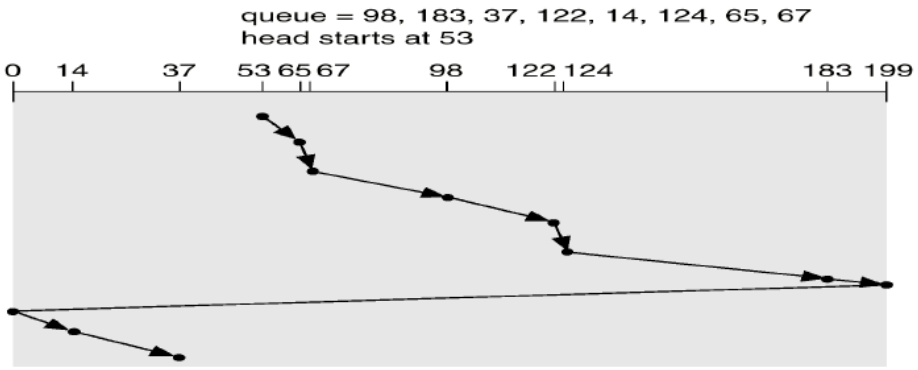
SCAN:

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. Sometimes called the *elevator algorithm*. Illustration shows total head movement of 208 cylinders.



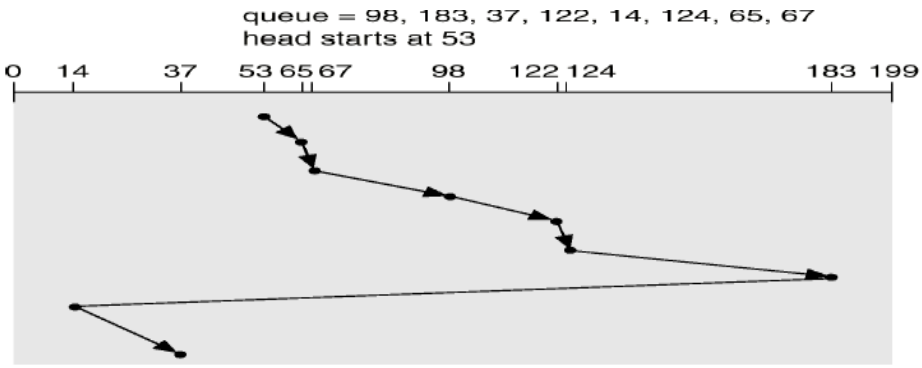
CSCAN:

- It provides a more uniform wait time than SCAN. The head moves from one end of the disk to the other. Servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. It treats the cylinders as a circular list that wraps around from the last cylinder to the first one.



CLOOK:

- It is the Version of CSCAN. The arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



12. What is clock? And give its types.

- Clocks (also called timers) are necessary to the operation of any multiprogrammed system for a variety of reasons. They keep the time of day and prevent one process from monopolizing the CPU, among other things. The clock software can take the form of a device driver, even though a clock is neither a block device, like a disk, nor a character device, like a mouse.

Clock Hardware

- Two types of clocks are normally used in computers, and both are quite different from the clocks and watches used by people. The simpler clocks are tied to the 110- or 220-volt power line and cause an interrupt on every voltage cycle, at 50 or 60 Hz. These clocks used to dominate, but are uncommon nowadays.

- The other kind of clock is built out of three components: a crystal oscillator, a counter, and a holding register, as shown in Figure 1. When a piece of quartz crystal is accurately cut and mounted under tension, it can be made to generate a periodic signal of very great accuracy, normally in the range of several hundred megahertz, depending on the crystal chosen. Using electronics, this base signal can be multiplied by a small integer to get frequencies up to 1000 MHz or even more. At least one such circuit is generally found in any computer, providing a synchronizing signal to the computer's various circuits. This signal is fed into the counter to make it count down to zero. When the counter gets to zero, it causes a CPU interrupt.

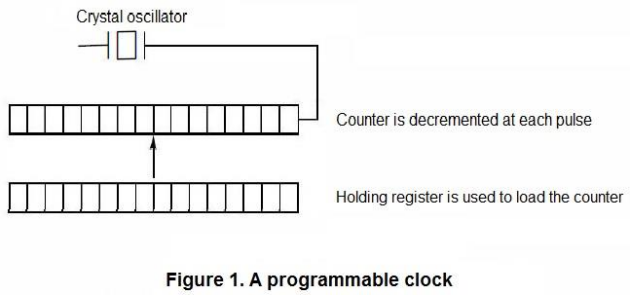


Figure 1. A programmable clock

- Programmable clocks normally have several modes of operation. In one-shot mode, when the clock is started, it copies the value of the holding register into the counter and then decrements the counter at each pulse from the crystal. When the counter gets to zero, it causes an interrupt and stops until it is explicitly started again by the software. In square-wave mode, after getting to zero and causing the interrupt, the holding register is automatically copied into the counter, and the whole process is repeated again indefinitely. These periodic interrupts are called clock ticks.

- The advantage of the programmable clock is that its interrupt frequency can be controlled by software. If a 500-MHz crystal is used, then the counter is pulsed every 2 nsec. With (unsigned) 32-bit registers, interrupts can be programmed to happen at intervals from 2 nsec to 8.6 sec. Programmable clock chips generally contain two or three independently programmable clocks and have many other options as well (e.g., counting up instead of down, interrupts disabled, and more).

- To prevent the current time from being lost when the computer's power is turned off, most computers have a battery-powered backup clock, implemented with the kind of low-power circuitry used in digital watches. The battery clock can be read at startup. If the backup clock is not present, the software may ask the user for the current date and time. There is also a standard way for a networked system to get the current time from a remote host.

Clock Software

- All the clock hardware generate interrupts at known intervals. Everything else involving time must be done by the software, the clock driver. The exact duties of the clock driver vary among operating systems, but generally include most of the following:

1. Maintaining the time of day.
2. Preventing processes from running longer than they are allowed to.
3. Accounting for CPU usage.
4. Handling the alarm system call made by user processes.
5. Providing watchdog timers for parts of the system itself.
6. Doing profiling, monitoring, and statistics gathering.

13.Explain any two user interfaces.

14. What is mean by input software and output software?

- Every general-purpose computer has a keyboard and monitor (and usually a mouse) to allow people to interact with it. Although the keyboard and monitor are technically separate devices, they work closely together. On mainframes, there are frequently many remote users, each with a device containing a keyboard and an attached display as a unit. These devices have historically been called terminals.

Input Software:

- User input comes primarily from the keyboard and mouse, so let us look at those. On a personal computer, the keyboard contains an embedded microprocessor which generally communicates through a specialized serial port with a controller chip on the parentboard (although increasingly keyboards are connected to a USB port). An interrupt is generated whenever a key is struck and a second one is generated whenever a key is released. At each of these keyboard interrupts, the keyboard driver extracts the information about what happens from the I/O port associated with the keyboard. Everything else happens in software and is pretty much independent of the hardware. Most of the rest of this section can be best understood when thinking of typing commands to a shell window (command line interface). This is how programmers usually work.

Keyboard Software

- The number in the I/O port is the key number, called the scan code, not the ASCII code. Keyboards have fewer than 128 keys, so only 7 bits are required to represent the key number. The eighth bit is set to 0 on a key press and to 1 on a key release. It is up to the driver to keep track of the status of each key (up or down). When the A key is struck, for instance, the scan code (30) is put in an I/O register. It is up to the driver to determine whether it is lower case, upper case, CTRL-A, ALT-A, CTRL-ALT-A, or some other combination. Since the driver can tell which keys have been struck but not yet released (e.g., SHIFT), it has enough information to do the job.

Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

Figure 1. Characters that are handled specially in canonical mode.

Mouse Software

- Most PCs have a mouse, or sometimes a trackball, which is just a mouse lying on its back. One common type of mouse has a rubber ball inside that protrudes through a hole in the bottom and rotates as the mouse is moved over a rough surface. As the ball rotates, it rubs against rubber rollers placed on orthogonal shafts. Motion in the east-west direction causes the shaft parallel to the y-axis to rotate; motion in the north-south direction causes the shaft parallel to the x-axis to rotate. Another popular mouse type is the optical mouse, which is equipped with one or more light-emitting diodes and photodetectors on the bottom.
- Some people call this unit a mickey. Mice (or occasionally, mouses) can have one, two, or three buttons, depending on the designers' estimate of the users' intellectual ability to keep track of more than one button. Some mice have wheels that can send additional data back to the computer. Wireless mice are the same as wired mice except instead of sending their data back to the computer over a wire, they use low-power radios, for instance, using the Bluetooth standard.

Output Software

Text Windows

- Output is simpler than input when the output is sequentially in a single font, size, and color. For the most part, the program sends characters to the current window and they are displayed there. Normally, a block of characters, for instance, a line, is written in one system call. Screen editors and many other sophisticated programs need to be able to update the screen in complex ways such as replacing one line in the middle of the screen. To accommodate this need, most output drivers support a series of commands to move the cursor, insert and delete characters or lines at the cursor, and so on.

15.Write a short note on X Window System.

- Nearly all UNIX systems base their user interface on the X Window System (often just called X), developed at M.I.T. as part of project Athena in the 1980s. It is very portable and runs entirely in user space. It was originally intended for connecting a large number of remote user terminals with a central compute server, so it is logically split into client software and host software, which can potentially run on different computers. On modern personal computers, both parts can run on the same machine.

- On Linux systems, the popular Gnome and KDE desktop environments run on top of X. When X is running on a machine, the software that collects input from the keyboard and mouse and writes output to the screen is called the X server. It has to keep track of which window is currently selected (where the mouse pointer is), so it knows which

client to send any new keyboard input to. It communicates with running programs (possible over a network) called X clients. It sends them keyboard and mouse input and accepts display commands from them.

- It may seem odd that the X server is always inside the user's computer while the X client may be off on a remote compute server, but just think of the X server's main job: displaying bits on the screen, so it makes sense to be near the user. From the program's point of view, it is a client telling the server to do things, like display text and geometric figures. The server (in the local PC) just does what it is told, as do all servers.

Escape sequence	Meaning
ESC [*n*A	Move up *n* lines
ESC [*n*B	Move down *n* lines
ESC [*n*C	Move right *n* spaces
ESC [*n*D	Move left *n* spaces
ESC [*m*;*n*H	Move cursor to (*m*,*n*)
ESC [*s*J	Clear screen from cursor (0 to end, 1 1from start, 2 all)
ESC [*s*K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n*L	Insert *n* lines at cursor
ESC [*n*M	Delete *n* lines at cursor
ESC [*n*P	Delete *n* chars at cursor
ESC [*n*@	Insert *n* chars at cursor
ESC [*n*m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

Figure 1. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and n, m, and s are optional numeric parameters.

- The arrangement of client and server is shown in Figure 2 for the case where the X client and X server are on different machines. But when running Gnome or KDE on a single machine, the client is just some application program using the X library talking to the X server on the same machine (but using a TCP connection over sockets, the same as it would do in the remote case).

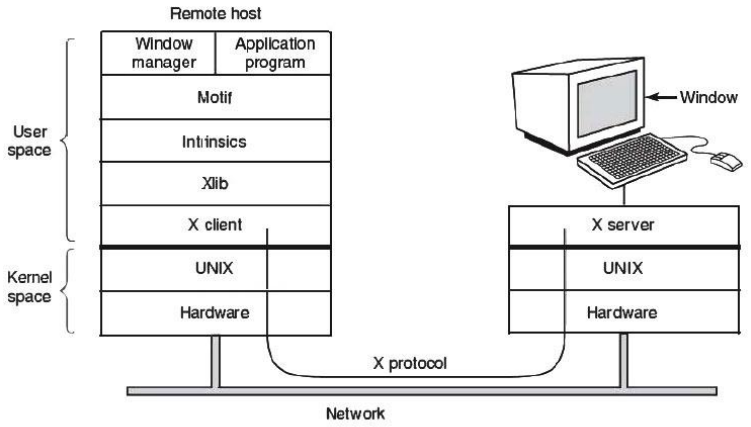


Figure 2. Clients and servers in the M.I.T. X Window System.

- The reason it is possible to run the X Window System on top of UNIX (or another operating system) on a single machine or over a network is that what X really defines is the X protocol between the X client and the X server, as shown in Figure 2. It does not matter whether the client and server are on the same machine, separated by 100 meters over a local area network, or are thousands of kilometers apart and connected by the Internet.

- The protocol and operation of the system is identical in all cases. X is just a windowing system. It is not a complete GUI. To get a complete GUI, others layer of software are run on top of it. One layer is Xlib, which is a set of library procedures for accessing the X functionality. These procedures form the basis of the X Window System and are what we will examine below, but they are too primitive for most user programs to access directly. For instance, each mouse click is reported separately, so that determining that two clicks really form a double click has to be handled above Xlib.

16. Explain the terms:

- **Graphical User Interfaces**
- **Bitmaps**

Graphical User Interface(GUI)

- Short for Graphical User Interface, a GUI (pronounced as either G-U-I or gooey) allows the use of icons or other visual indicators to interact with electronic devices, rather than using only text via the command line. For example, all versions of Microsoft Windows utilize a GUI, whereas MS-DOS does not. The GUI was first developed at Xerox PARC by Alan Kay, Douglas Engelbart, and a group of other researchers in 1981. Later, Apple introduced the Lisa computer, the first commercially available computer, on January 19, 1983. Below is a picture of the Windows 7 Desktop and an example of a GUI.

How does a GUI work?

A GUI uses windows, icons, and menus to carry out commands, such as opening, deleting, and moving files. Although many GUI operating systems are navigated through the use of a mouse, the keyboard can also be utilized by using keyboard shortcuts or arrow keys.

What are the benefits of GUI?

Unlike a command line operating system or CUI, like Unix or MS-DOS, GUI operating systems are much easier to learn and use because commands do not need to be memorized. Additionally, users do not need to know any programming languages. Because of their ease of use, GUI operating systems have become the dominant operating system used by today's end-users.

What are examples of a GUI operating system?

- Microsoft Windows
- Apple System 7 and Mac OS
- Chrome OS
- Linux

Are all operating systems GUI?

No. Early command line operating systems like MS-DOS and even some versions of Linuxtoday have no GUI interface.

What are examples of a GUI interface?

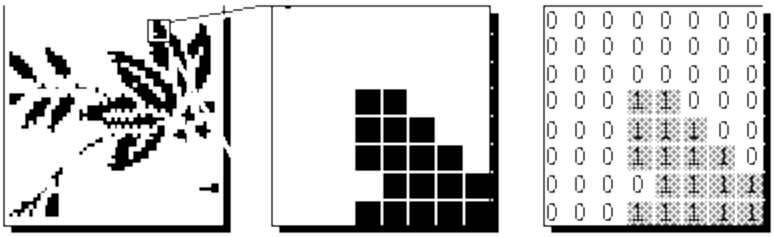
- GNOME
- KDE
- Any Microsoft program (i.e. Word, Excel, Outlook)
- Internet browser (i.e. Internet Explorer, Chrome, Firefox)

Bitmap:

- A bitmap (or raster graphic) is a digital image composed of a matrix of dots. When viewed at 100%, each dot corresponds to an individual pixel on a display. In a standard bitmap image, each dot can be assigned a different color. Together, these dots can be used to represent any type of rectangular picture.
- There are several different bitmap file formats. The standard, uncompressed bitmap format is also known as the "BMP" format or the device independent bitmap (DIB) format. It includes a header, which defines the size of the image and the number of colors the image may contain, and a list of pixels with their corresponding colors. This

simple, universal image format can be recognized on nearly all platforms, but is not very efficient, especially for large images.

- Other bitmap image formats, such as JPEG, GIF, and PNG, incorporate compression algorithms to reduce file size. Each format uses a different type of compression, but they all represent an image as a grid of pixels. Compressed bitmaps are significantly smaller than uncompressed BMP files and can be downloaded more quickly. Therefore, most images you see on the web are compressed bitmaps.
- If you zoom into a bitmap image, regardless of the file format, it will look blocky because each dot will take up more than one pixel. Therefore, bitmap images will appear blurry if they are enlarged. Vector graphics, on the other hand, are composed of paths instead of dots, and can be scaled without reducing the quality of the image.



17.What is touch screens? What are its types?

- Buttons on electronic devices are now passé: touch screens are rapidly becoming the ubiquitous interface for controlling them.
- While many of us have started using touch screens only after they went mainstream in modern smart phones, the technology itself has been around for decades.
- Among other places, touch screens are found in handheld devices used in industrial settings, navigational equipment, retail point of sale terminals, vending machines, kiosks and ATMs.
- Touch screen technologies can be mainly divided into two types- overlay based and perimeter based.
- In overlay based touch screen technologies, the screen is integral to the mechanics of registering input. Touch events are registered when the object touches the screen.

There are two types of overlay based touch screens:

Capacitive Touch Technology – Capacitive touch screens take advantage of the conductivity of the object to detect location of touch. While they are durable and last for a long time, they can malfunction if they get wet. Their performance is also compromised if a non conductor like a gloved finger presses on the screen. Most smart phones and tablets have capacitive touch screens.

Resistive Touch Technology – Resistive touch screens have moving parts. There is an air gap between two layers of transparent material. When the user applies pressure to the outer layer,

it touches the inner layer at specific locations. An electric circuit is completed and the location can be determined. Though they are cheaper to build compared to capacitive touch screens, they are also less sensitive and can wear out quickly.

- **Perimeter based touch screen** technologies work differently. In this case, the screen is incidental to the main process of registering input. These screens have beam break sensors or cameras embedded in the bezel. These sensors monitor light or acoustic waves emitted by LEDs or transducers, also set in the bezel.
- When an object touches the screen, there is a break in the path of the rays. This creates a shadow or a dark area which is detected by the sensors. Using complex mathematical calculations, and in some cases heavy data processing the location of the object can be determined.

There are mainly three types of perimeter based technologies:

Infrared Touch Technology – This technology uses beams of infrared lights to detect touch events.
Surface Acoustic Wave Touch Technology – This type of touch screen uses ultrasonic waves to detect touch events.
Optical Touch Technology – This type of perimeter based technology uses optical sensors, mainly CMOS sensors to detect touch events.

- All of these touch screen technologies can also be integrated on top of a non touch based system like an ordinary LCD and converted into Open Frame Touch Monitors.
- **ShadowSense** is a form of perimeter based technology. It also uses light emitted by LEDs and the shadows thrown by the object to calculate size and location on the screen. However, the implementation is simpler and the screen itself runs without any drivers. ShadowSense touch screens also have numerous other advantages over conventional touch screens.

18.Explain thin client and give its benefits.

- A thin client is a lightweight computer that is purpose-built for remote access to a server (typically cloud or desktop virtualization environments). It depends heavily on another computer (its server) to fulfill its computational roles. The specific roles assumed by the server may vary, from hosting a shared set of virtualized applications, a shared desktop stack or virtual desktop, to data processing and file storage on the client's or user's behalf. This is different from the desktop PC (fat client), which is a computer designed to take on these roles by itself.
- Thin clients occur as components of a broader computing infrastructure, where many clients share their computations with a server or server farm. The server-side infrastructure makes use of cloud computing software such as application virtualization,

hosted shared desktop (HSD) or desktop virtualization (VDI). This combination forms what is known today as a cloud based system where desktop resources are centralized into one or more data centers.

- Thin client hardware generally consists of a computer terminal which provides I/O for a keyboard, mouse, monitor, jacks for sound peripherals, and open ports for USB devices, e.g. printer, flash drive, web cam, card reader, smartphone, etc. Some thin clients include legacy serial and/or parallel ports to support older devices such as receipt printers, scales, time clocks, etc. Thin client software typically consists of a GUI (graphical user interface), cloud access agents (e.g. RDP, ICA, PCoIP), a local web browser, terminal emulations (in some cases), and a basic set of local utilities.

- There are many thin client benefits for different enterprises and their IT infrastructure. IT departments are migrating to unique platforms to centralize business through today's virtual desktop technologies.

1) Cost Savings

- Thin Clients Reduce Multiple Costs:
- IT support costs
- Upfront purchasing costs
- Capital costs
- Use of space in data center
- Licensing costs
- Total administration and operating cost reduction up to 70%
- Reduces Energy Bill by 97% [www.itgct.com, Information Technology Group]
- Thin clients consume an average of 8-20 watts compared to a 150 watt PC
- This reduces carbon footprint
- Cost savings from electricity can be reinvested

2) Simplified Management

The Benefits of Thin Client Management are:

- All software and hardware upgrades, security policies, application changes, etc. can be made in the data center
- IT personnel are not required (as they are with PCs) to fix individual problems at the end user desktop location
- Less downtime, increasing productivity amongst End Users and IT personnel
- Centralized and simplified back up of desktops, laptops, and other client access devices

3) Enhanced Security

The Benefits of Thin Client Security Include:

- Thin clients are protected from the use of unauthorized software or the introduction of viruses
- Data cannot be copied to a disk or saved to any other location than the server
- Centralized processing makes it easy to manage and monitor the system
- Simplify security, protect intellectual property, ensure data privacy

4) Increased Productivity

- Systems can be Virtually Preconfigured, Packaged and Put into Operation in Minutes
- Quickens setup and enables flexibility, without needing a specialist staff
- Productivity can increase, while standard PCs have long repair times that cause delays and higher costs
- Access the same apps and data from virtually anywhere

By creating a dynamic and flexible IT environment with thin clients, IT departments worldwide are simplifying their IT infrastructures.
Thin Clients can be a valuable technology platform and solves several business problems. They increase device usage flexibility since the user can access their virtual desktop from their home PC, tablet, or Smartphone creating a more secure platform

19.Write a note on power management.

- The first general-purpose electronic computer, the ENIAC, had 18,000 vacuum tubes and consumed 140,000 watts of power. As a result, it ran up a nontrivial electricity bill. After the invention of the transistor, power usage dropped dramatically and the computer industry lost interest in power requirements. However, nowadays power management is back in the spotlight for several reasons, and the operating system is playing a role here.
- Let us start with desktop PCs. A desktop PC often has a 200-watt power supply (which is typically 85% efficient, that is, loses 15% of the incoming energy to heat). If 100 million of these machines are turned on at once worldwide, together they use 20,000 megawatts of electricity. This is the total output of 20 average-sized nuclear power plants. If power requirements could be cut in half, we could get rid of 10 nuclear power plants. From an environmental point of view, getting rid of 10 nuclear power plants (or an equivalent number of fossil fuel plants) is a big win and well worth pursuing.
- The other place where power is a big issue is on battery-powered computers, including notebooks, laptops, palmtops, and Webpads, among others. The heart of the problem is that the batteries cannot hold enough charge to last very long, a few hours at most. Furthermore, despite massive research efforts by battery companies, computer

companies, and consumer electronics companies, progress is glacial. To an industry used to a doubling of the performance every 18 months (Moore's law), having no progress at all seems like a violation of the laws of physics, but that is the current situation. As a consequence, making computers use less energy so existing batteries last longer is high on everyone's agenda. The operating system plays a major role here, as we will see below.

- There are two general approaches to reducing energy consumption. The first one is for the operating system to turn off parts of the computer (mostly I/O devices) when they are not in use because a device that is off uses little or no energy. The second one is for the application program to use less energy, possibly degrading the quality of the user experience, in order to stretch out battery time. We will look at each of these approaches in turn, but first we will say a little bit about hardware design with respect to power usage.

Hardware Issues

- Batteries come in two general types: disposable and rechargeable. Disposable batteries (most commonly AAA, AA, and D cells) can be used to run handheld devices, but do not have enough energy to power laptop computers with large bright screens. A rechargeable battery, in contrast, can store enough energy to power a laptop for a few hours. Nickel cadmium batteries used to dominate here, but they gave way to nickel metal hydride batteries, which last longer and do not pollute the environment quite as badly when they are eventually discarded.
- The general approach most computer vendors take to battery conservation is to design the CPU, memory, and I/O devices to have multiple states: on, sleeping, hibernating, and off. To use the device, it must be on. When the device will not be needed for a short time, it can be put to sleep, which reduces energy consumption. When it is not expected to be needed for a longer interval, it can be made to hibernate, which reduces energy consumption even more. The trade-off here is that getting a device out of hibernation often takes more time and energy than getting it out of sleep state. Finally, when a device is off, it does nothing and consumes no power. Not all devices have all these states, but when they do, it is up to the operating system to manage the state transitions at the right moments.
- Some computers have two or even three power buttons. One of these may put the whole computer in sleep state, from which it can be awakened quickly by typing a character or moving the mouse. Another may put the computer into hibernation, from which wakeup takes much longer. In both cases, these buttons typically do nothing except send a signal to the operating system, which does the rest in software. In some countries, electrical devices must, by law, have a mechanical power switch that breaks a circuit and removes power from the device, for safety reasons. To comply with this law, another switch may be needed.

20.What are the operating system issues?

- The operating system plays a key role in energy management. It controls all the devices, so it must decide what to shut down and when to shut it down. If it shuts down a device and that device is needed again quickly, there may be an annoying delay while it is restarted. On the other hand, if it waits too long to shut down a device, energy is wasted for nothing.
- The trick is to find algorithms and heuristics that let the operating system make good decisions about what to shut down and when. The trouble is that ''good'' is highly subjective. One user may find it acceptable that after 30 seconds of not using the computer it takes 2 seconds for it to respond to a keystroke.

The Display

- Let us now look at the big spenders of the energy budget to see what can be done about each one. The biggest item in everyone's energy budget is the display. To get a bright sharp image, the screen must be backlit and that takes substantial energy. Many operating systems attempt to save energy here by shutting down the display when there has been no activity for some number of minutes. Often the user can decide what the shutdown interval is, pushing the trade-off between frequent blanking of the screen and using the battery up quickly back to the user (who probably really does not want it). Turning off the display is a sleep state because it can be regenerated (from the video RAM) almost instantaneously when any key is struck or the pointing device is moved.

The Hard Disk

- Another major villain is the hard disk. It takes substantial energy to keep it spinning at high speed, even if there are no accesses. Many computers, especially laptops, spin the disk down after a certain number of minutes of activity. When it is next needed, it is spun up again. Unfortunately, a stopped disk is hibernating rather than sleeping because it takes quite a few seconds to spin it up again, which causes noticeable delays for the user.

The CPU

- The CPU can also be managed to save energy. A laptop CPU can be put to sleep in software, reducing power usage to almost zero. The only thing it can do in this state is wake up when an interrupt occurs. Therefore, whenever the CPU goes idle, either waiting for I/O or because there is no work to do, it goes to sleep.
- On many computers, there is a relationship between CPU voltage, clock cycle, and power usage. The CPU voltage can often be reduced in software, which saves energy but also reduces the clock cycle (approximately linearly). Since power consumed is

proportional to the square of the voltage, cutting the voltage in half makes the CPU about half as fast but at 1/4 the power.

The Memory

- Two possible options exist for saving energy with the memory. First, the cache can be flushed and then switched off. It can always be reloaded from main memory with no loss of information. The reload can be done dynamically and quickly, so turning off the cache is entering a sleep state.

Wireless Communication

- Increasingly many portable computers have a wireless connection to the outside world (e.g., the Internet). The radio transmitter and receiver required are often first-class power hogs. In particular, if the radio receiver is always on in order to listen for incoming email, the battery may drain fairly quickly. On the other hand, if the radio is switched off after, say, 1 minute of being idle, incoming messages may be missed, which is clearly undesirable.

Thermal Management

- A somewhat different, but still energy-related issue, is thermal management. Modern CPUs get extremely hot due to their high speed. Desktop machines normally have an internal electric fan to blow the hot air out of the chassis. Since reducing power consumption is usually not a driving issue with desktop machines, the fan is usually on all the time.
- With laptops, the situation is different. The operating system has to monitor the temperature continuously. When it gets close to the maximum allowable temperature, the operating system has a choice. It can switch on the fan, which makes noise and consumes power. Alternatively, it can reduce power consumption by reducing the backlighting of the screen, slowing down the CPU, being more aggressive about spinning down the disk, and so on.

Battery Management

- In olden days, a battery just provided current until it was drained, at which time it stopped. Not any more. Laptops use smart batteries now, which can communicate with the operating system. Upon request they can report on things like maximum voltage, current voltage, maximum charge, current charge, maximum drain rate, current drain rate, and more. Most laptop computers have programs that can be run to query and display all these parameters. Smart batteries can also be instructed to change various operational parameters under control of the operating system.