Q.1 Attempt the following (any THREE) : [15]

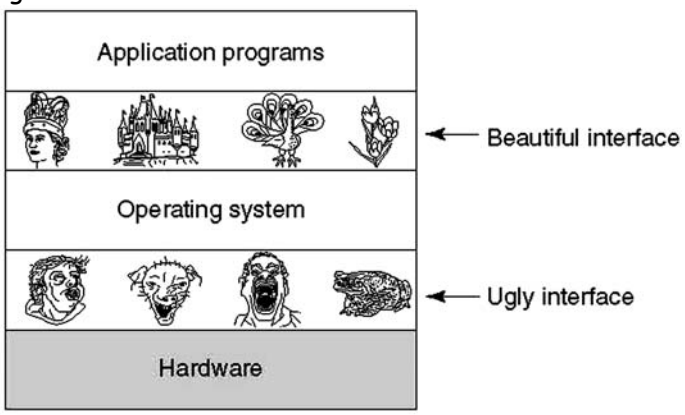
Q.1(a) Define Operating System. Explain its role. [5]

Ans.: It is hard to pin down what an operating system is other than saying it is the software that runs in kernel mode—and even that is not always true. Part of the problem is that operating systems perform two essentially unrelated functions : providing application programmers (and application programs, naturally) a clean abstract set of resources instead of the messy hardware ones and managing these hardware resources. Depending on who is doing the talking, you might hear mostly about one function or the other. Let us now look at both.

The architecture (instruction set, memory organization, I/O, and bus structure) of most computers at the machine-language level is primitive and awkward to program, especially for input/output.

Clearly, no sane programmer would want to deal with this disk at the hardware level. Instead, a piece of software, called a disk driver, deals with the hardware and provides an interface to read and write disk blocks, without getting into the details. Operating systems contain many drivers for controlling I/O devices. But even this level is much too low for most applications. For this reason, all operating systems provide yet another layer of abstraction for using disks: files. Using this abstraction, programs can create, write, and read files, without having to deal with the messy details of how the hardware actually works.

Real processors, memories, disks, and other devices are very complicated and present difficult, awkward, idiosyncratic, and inconsistent interfaces to the people who have to write software to use them. Sometimes this is due to the need for backward compatibility with older hardware. Other times it is an attempt to save money. Often, however, the hardware designers do not realize (or care) how much trouble they are causing for the software. One of the major tasks of the operating system is to hide the hardware and present programs (and their programmers) with nice, clean, elegant, consistent, abstractions to work with instead. Operating systems turn the ugly into the beautiful, as shown in the diagram below:



Q.1(b) Explain first two generations of Operating Systems. [5]

Ans.: (i) The First Generation (1945–55) : Vacuum Tubes

Little progress was made in constructing digital computers until the World War II period, which stimulated an explosion of activity. Professor John Atanasoff and his graduate student Clifford Berry built what is now regarded as the first functioning

digital computer at Iowa State University. It used 300 vacuum tubes. It was very primitive and took seconds to perform even the simplest calculation.

In these early days, a single group of people (usually engineers) designed, built, programmed, operated, and maintained each machine. All programming was done in absolute machine language, or even worse yet, by wiring up electrical circuits by connecting thousands of cables to plugboards to control the machine's basic functions. Programming languages were unknown (even assembly language was unknown). Operating systems were unheard of. The usual mode of operation was for the programmer to sign up for a block of time using the signup sheet on the wall, then come down to the machine room, insert his or her plugboard into the computer, and spend the next few hours hoping that none of the 20,000 or so vacuum tubes would burn out during the run.

(ii) The Second Generation (1955–65) : Transistors and Batch Systems

The introduction of the transistor in the mid-1950s changed the picture radically. Computers became reliable enough that they could be manufactured and sold to paying customers with the expectation that they would continue to function long enough to get some useful work done. For the first time, there was a clear separation between designers, builders, operators, programmers, and maintenance personnel.

These machines, now called mainframes, were locked away in large, specially air-conditioned computer rooms, with staffs of professional operators to run them. Only large corporations or major government agencies or universities could afford the multimillion-dollar price tag. To run a job (i.e., a program or set of programs), a programmer would first write the program on paper (in FORTRAN or assembler), then punch it on cards. He would then bring the card deck down to the input room and hand it to one of the operators and go drink coffee until the output was ready.

When the computer finished whatever job it was currently running, an operator would go over to the printer and tear off the output.

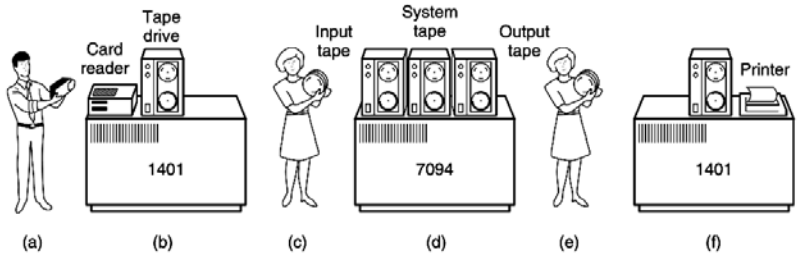


Fig.: An early batch system. (a) Programmers bring cards to 1401, (b) 1401 reads batch of jobs onto tape, (c) Operator carries input tape to 7094, (d) 7094 does computing, (e) Operator carries output tape to 1401, (f) 1401 prints output. (this diagram is not expected to be drawn in exam)

Q.1(c) Explain arrangement of hardware on Computer. [5]

Ans.: An operating system is intimately tied to the hardware of the computer it runs on. It extends the computer's instruction set and manages its resources. To work, it must know a great deal about the hardware, at least about how the hardware appears to the programmer.

Conceptually, a simple personal computer can be abstracted to a model resembling that of the diagram shown below. The CPU, memory, and I/O devices are all connected by a system bus and communicate with one another over it. Modern personal computers have a more complicated structure, involving multiple buses.

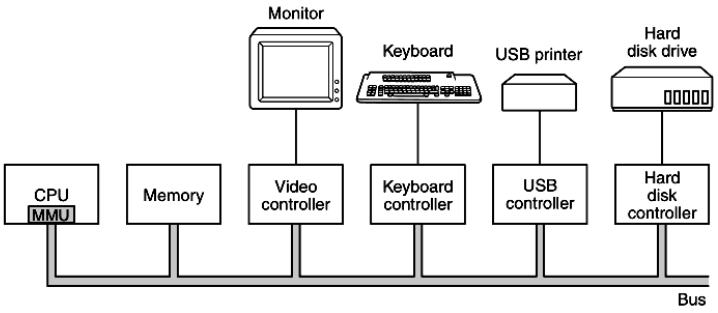


Fig.: Some of the components of a simple personal computer.

Q.1(d) Explain any three types of Operating Systems. **[5]**

Ans.: Operating systems have been around now for over half a century. During this time, quite a variety of them have been developed, not all of them widely known. Following are nine prominent types of Operating Systems:

(a) Mainframe Operating Systems : At the high end are the operating systems for mainframes, those room-sized computers still found in major corporate data centers. These computers differ from personal computers in terms of their I/O capacity. A mainframe with 1000 disks and millions of gigabytes of data is not unusual. The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need prodigious amounts of I/O. They typically offer three kinds of services: batch, transaction processing, and timesharing.

A batch system is one that processes routine jobs without any interactive user present. Transaction-processing systems handle large numbers of small requests, for example, check processing at a bank or airline reservations. Timesharing systems allow multiple remote users to run jobs on the computer at once, such as querying a big database. These functions are closely related; mainframe operating systems often perform all of them. An example mainframe operating system is OS/390, a descendant of OS/360.

(b) Server Operating Systems : One level down are the server operating systems. They run on servers, which are either very large personal computers, workstations, or even mainframes. They serve multiple users at once over a network and allow the users to share hardware and software resources. Internet providers run many server machines to support their customers and Websites use servers to store the Web pages and handle the incoming requests.

(c) Multiprocessor Operating Systems: An increasingly common way to get major-league computing power is to connect multiple CPUs into a single system. Depending on precisely how they are connected and what is shared, these systems are called parallel computers, multi-computers, or multiprocessors. They need special operating systems, but often these are variations on the server operating systems, with special features for communication, connectivity, and consistency.

With the recent advent of multicore chips for personal computers, even conventional desktop and notebook operating systems are starting to deal with at least small-scale multiprocessors and the number of cores is likely to grow over time.

(d) Personal Computer Operating Systems: The next category is the personal computer operating system. Modern ones all support multiprogramming, often with dozens of programs started up at boot time. Their job is to provide good support to a single user. They are widely used for word processing, spreadsheets, games, and Internet access. Common examples are Linux, FreeBSD, Windows 7, Windows 8, and Apple's OS X. Personal computer operating systems are so widely known that probably little introduction is needed. In fact, many people are not even aware that other kinds exist.

(e) Handheld Computer Operating Systems: Continuing on down to smaller and smaller systems, we come to tablets, smartphones and other handheld computers. A handheld computer, originally known as a PDA (Personal Digital Assistant), is a small computer that can be held in your hand during operation. Smartphones and tablets are the best-known examples. As we have already seen, this market is currently dominated by Google's Android and Apple's iOS, but they have many competitors.

(f) Embedded Operating Systems: Embedded systems run on the computers that control devices that are not generally thought of as computers and which do not accept user-installed software. Typical examples are microwave ovens, TV sets, cars, DVD recorders, traditional phones, and MP3 players. The main property which distinguishes embedded systems from handhelds is the certainty that no untrusted software will ever run on it. You cannot download new applications to your microwave oven—all the software is in ROM. This means that there is no need for protection between applications, leading to design simplification. Systems such as Embedded Linux, QNX and VxWorks are popular in this domain.

(g) Sensor-Node Operating Systems: Networks of tiny sensor nodes are being deployed for numerous purposes. These nodes are tiny computers that communicate with each other and with a base station using wireless communication. Sensor networks are used to protect the perimeters of buildings, guard national borders, detect fires in forests, measure temperature and precipitation for weather forecasting, glean information about enemy movements on battlefields, and much more.

Each sensor node is a real computer, with a CPU, RAM, ROM, and one or more environmental sensors. It runs a small, but real operating system, usually one that is event driven, responding to external events or making measurements periodically based on an internal clock. The operating system has to be small and simple because the nodes have little RAM and battery lifetime is a major issue. Also, as with embedded systems, all the programs are loaded in advance; users do not suddenly start programs they downloaded from the Internet, which makes the design much simpler. TinyOS is a well-known operating system for a sensor node.

(h) Real-Time Operating Systems: Another type of operating system is the real-time system. These systems are characterized by having time as a key parameter. For example, in industrial process-control systems, real-time computers have to collect data about the production process and use it to control machines in the factory. Often there are hard deadlines that must be met. For example, if a car is moving down an assembly line, certain actions must take place at certain instants of time. If, for example, a welding robot welds too early or too late, the car will be ruined.

If the action absolutely must occur at a certain moment (or within a certain range), we have a hard real-time system. Many of these are found in industrial process control, avionics, military and similar application areas. These systems must provide absolute guarantees that a certain action will occur by a certain time. A soft real-time system, is one where missing an occasional deadline, while not desirable, is acceptable and does not cause any permanent damage. Digital audio or multimedia systems fall in this category. Smartphones are also soft real-time systems.

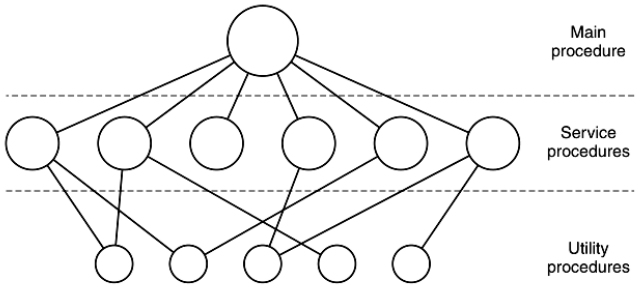
(i) Smart Card Operating Systems: The smallest operating systems run on smart cards, which are credit-card-sized devices containing a CPU chip. They have very severe processing power and memory constraints. Some are powered by contacts in the reader into which they are inserted, but contactless smart cards are inductively powered, which greatly limits what they can do. Some of them can handle only a single function, such as electronic payments, but others can handle multiple functions. Often these are proprietary systems.

Q.1(e) Explain Monolithic Systems and Layered Systems. **[5]**

Ans.: Monolithic Systems : Even in monolithic systems, however, it is possible to have some structure. The services (system calls) provided by the operating system are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction. This instruction switches the machine from user mode to kernel mode and transfers control to the operating system. The operating system then fetches the parameters and determines which system call is to be carried out. After that, it indexes into a table that contains in slot k a pointer to the procedure that carries out system call k.

This organization suggests a basic structure for the operating system :

(1) A main program that invokes the requested service procedure.
(2) A set of service procedures that carry out the systems calls.
(3) A set of utility procedures that help the service procedures.



Layered Systems : A generalization of the approach is to organize the operating system as a hierarchy of layers, each one constructed upon the one below it. The first system constructed in this way was the THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E.W. Dijkstra (1968) and his students. The THE system was a simple batch system for a Dutch computer, the electrologica X8, which had 32K of 27-bit words (bits were expensive back then).

The system had six layers, as shown in Figure. Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired. Above layer 0, the system consisted of sequential processes, each of which could be programmed without having to worry about the fact that multiple processes were running on a single processor. In other words, layer 0 provided the basic multiprogramming of the CPU.

Layer	function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Fig. : Structure of the THE operating system

Q.1(f) Explain FCFS Scheduling Algorithm with example. **[5]**

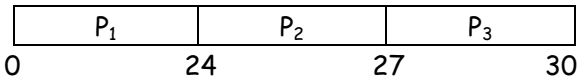
Ans.: (i) FCFS :

- The process that required the CPU first is allocated the CPU first.
  When a process enters a ready queue its PCB is linked onto the tail of the queue when the CPU is free it is allocated to the process at the head of the queue.
- e.g. Consider the set of processes that arrive at time 0.

Process	Burst Time
P_1	24
P_2	3
P_3	3

Processes P_1, P_2 and P_3 arrive in order and served in FCFS manner then results we can show in Gantt as follows :

Gantt Chart :



Waiting time	Process
0	P_1
24	P_2
27	P_3

Average waiting time = 17 ms

Q.2Attempt the following (any THREE) : [15]

Q.2(a) Explain the concept of no memory abstraction. [5]

Ans.: The simplest memory abstraction is to have no abstraction at all. Early mainframe computers (before 1960), early minicomputers (before 1970), and early personal computers (before 1980) had to memory abstraction. Every program simply saw the physical memory. When a program executed an instruction like

MOV REGISTER1, 1000

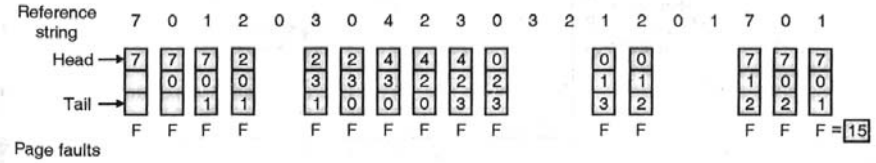
the computer just moved the contents of physical memory location 1000 to REGISTER1. Thus, the model of memory presented to the programmer was simply physical memory, a set of addresses from 0 to some maximum, each address corresponding to a cell containing some number of bits, commonly eight.

Under these conditions, it was not possible to have two running programs in memory at the same time. If the first program wrote a new value to, say, location 2000, this would erase whatever value the second program was storing there. Nothing would work and both programs would crash almost immediately.

Q.2(b) Explain FIFO Page replacement algorithm with example. [5]

Ans.: FIFO Page – Replacement :

- It is the simplest page – replacement algorithm.
- A FIFO replacement arguments with each page the time when that page was brought into memory.
- The oldest page is chosen as a victim for replacement.
- We create a FIFO queue to hold all pages in memory and replace the page at head of the queue.
- The page at the tail of queue is the most recent page.
- This scheme is modeled on the real time situation of a queue, say a bus queue where people standing at the head gets to board the bus first and the people at the tail are the ones who have newly joined the queue.
- For our example reference string, the FIFO scheme is shown in figure.



Q.2(c) Write a short note on Paging. [5]

Ans.:

- As we know there two ways to manage the physical memory:
  (a) Contiguous allocation
  (b) Non-contiguous allocation

- Basically, when we say that memory is contiguous, what we exactly mean is the word adjacent; next i.e. the memory consists of consecutive, adjacent physical addresses.
- Compliers and linkers assume contiguous memory, but using memory-mapping techniques we have logical addresses, which are adjacent. But these adjacent or contiguous logical addresses need not have contiguous physical locations assigned to them.
- We have learnt that if we are to use contiguous allocation of memory then we have to either use fixed partitioning scheme or dynamic partitioning scheme.
- There were several ways in which we could maintain these partitions (like bitmap method and list method)
- Both these schemes suffer from one deadly sting of fragmentation. The fixed sized partitions suffer from internal fragmentation and the dynamic partitions suffer from external fragmentation.
- The OS memory manager manages physical memory space and not the logical one. Thus we can divide the program into several parts and put each part into a separate area in physical memory. Each part will be smaller and hence will fit into smaller free blocks, and so effects of fragmentation will be minimized.

Q.2(d) Explain any five file operations. **[5]**

Ans.: File Operations :

(i) Files exists to store information and allow it be retrieved later. Different systems provide different operations allow storage and retrieval.

(ii) Most common system calls relating to files.

(a) CREATE: Purpose of the call is to announce that the file is coming sand to set some of the attributes.

(b) DELETE: When the file is no longer needed, it has to be deleted to free up disk space.

(c) OPEN: Before using a file, a process must open it. The purpose of this call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.

(d) CLOSE: When all processes are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space.

(e) READ: Data are read from file. Bytes comes from current position. The caller must specify how much data are needed and must also provide a buffer to put them in.

(f) WRITE: Data are written to file at current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file existing data are overwritten and lost forever.

(g) APPEND: It is a restricted form of WRITE. It can only add data to the end of file.

(h) SEEK: For random access files, a method is needed to specify from where to take the data. SEEK, system call repositions, the pointer to the current position to a specific place in the file.

(i) GET ATTRIBUTES: Processes often need to read file attributes to do their work.

(j) SET ATTRIBUTES: Some of the attributes are user settable and can be changed after the file has been created. This system call makes it possible.

(k) RENAME: It frequently happens that a user needs to change the name of an existing file. This system call makes that possible.

Q.2(e) Explain MS-DOS File system. **[5]**

Ans.: The MS-DOS file system is the one the first IBM PCs came with. It was the main file system up through Windows 98 and Windows ME. It is still supported on Windows 2000, Windows XP, and Windows Vista, although it is no longer standard on new PCs now except for floppy disks. However, it and an extension of it (FAT-32) have become widely used for many embedded systems. Most digital cameras use it. Many MP3 players use it exclusively. The popular Apple iPod uses it as the default file system, although knowledgeable hackers can reformat the iPod and install a different file system. Thus the number of electronic

devices using the MS-DOS file system is vastly larger now than at any time in the past, and certainly much larger than the number using the more modern NTFS file system. For that reason alone, it is worth looking at in some detail.

To read a file, an MS-DOS program must first make an open system call to get a handle for it. The open system call specifies a path, which may be either absolute or relative to the current working directory. The path is looked up component by component until the final directory is located and read into memory. It is then searched for the file to be opened.

Although MS-DOS directories are variable sized, they use a fixed-size 32-byte directory entry. The format of an MS-DOS directory entry is shown in Figure. It contains the file name, attributes, creation date and time, starting block, and exact file size. File names shorter than 8 + 3 characters are left justified and padded with spaces on the right, in each field separately. The Attributes field is new and contains bits to indicate that a file is read-only, needs to be archived, is hidden, or is a system file. Read-only files cannot be written. This is to protect them from accidental damage. The archived bit has no actual operating system function (i.e., MS-DOS does not examine or set it). The intention is to allow user-level archive programs to clear it upon archiving a file and to have other programs set it when modifying a file. In this way, a backup program can just examine this attribute bit on every file to see which files to back up. The hidden bit can be set to prevent a file from appearing in directory listings. Its main use is to avoid confusing novice users with files they might not understand. Finally, the system bit also hides files. In addition, system files cannot accidentally be deleted using the del command. The main components of MS-DOS have this bit set.

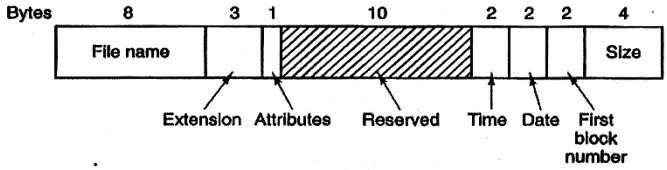


Fig.: The MS-DOS directory entry.

Q.2(f) Describe CD-ROM file system. **[5]**

Ans.:

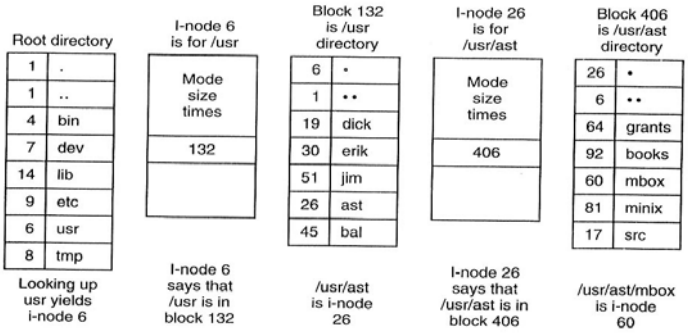


Fig. : The steps in looking up/usr/mbox.

As our last example of a file system, let us consider the file systems used on CD-ROMs. These systems are particularly simple because they were designed for write-once media. Among other things, for example, they have no provision for keeping track of free blocks because on a CD-ROM files cannot be freed or added after the disk has been manufactured. Below we will take a look at the main CD-ROM file system type and two extensions to it. While CD-ROMs are now old, they are also simple, and the file systems used on DVDs and Blu-ray are based on the one for CD-ROMS.

Some years after the CD-ROM made its debut, the CD-R (CD Recordable) was introduced. Unlike the CD-ROM, it is possible to add files after the initial burning, but these are simply appended to the end of the CD-R. Files are never removed (although the directory can be updated to hide existing files). As a consequence of this "append-only" file system, the fundamental properties are not altered. In particular, all the free space is in one contiguous chunk at the end of the CD.

Q.3Attempt the following (any THREE) : [15]

Q.3(a) Explain principles of I/O Software. [5]

Ans.: A key concept in the design of I/O software is known as device independence. What it means is that we should be able to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a hard disk, a DVD, or on a USB stick without having to be modified 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. For example, a USB stick can be mounted on top of the directory /usr/ast/backup so that copying a file to /usr/ast/backup/monday copies the file to the USB stick. In this way, all files and devices are addressed the same way: by a path name.

Another important issue for I/O software is error handling. In general, 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 frequently 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 important issue is that of synchronous (blocking) vs. 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. User programs are much easier to write if the I/O operations are blocking—after a read system call the program is automatically suspended until the data are available in the buffer. It is up to the operating system to make operations that are actually interrupt-driven look blocking to the user programs.

Another issue for the I/O software is buffering. Often data that come off a device cannot be stored directly in their final destination. For example, when a packet comes in off the network, the operating system does not know where to put it until it has stored the packet somewhere and examined it. Also, some devices have severe real-time constraints (for example, digital audio devices), so the data must be put into an output buffer in advance to decouple the rate at which the buffer is filled from the rate at which it is emptied, in order to avoid buffer underruns. Buffering involves considerable copying and often has a major impact on I/O performance.

The final concept is sharable vs. 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. Having two or more users writing characters

intermixed at random to the same page will definitely not work. Introducing dedicated (unshared) devices also introduces a variety of problems, such as deadlocks. Again, the operating system must be able to hanfle both shared and dedicated devices in a way that avoids problems.

Q.3(b) Describe in brief the concept of I/O Software layers with suitable diagram. [5]

Ans.: I/O software is typically organized in four layers, as shown in the diagram below. 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.

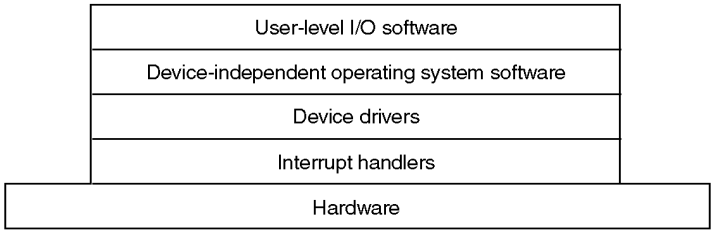


Fig.: Layers of the I/O software system.

(i) Layer 1 : Interrupt Handlers : While 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, for example, by doing a down on a semaphore, a wait on a condition variable, a receive on a message, or something similar.

(ii) Layer 2 : Device Drivers : Each hardware device has a controller which has some device registers used to give it commands or some device registers used to read out its status or both. The number of device registers and the nature of the commands vary radically from device to device. For example, a mouse driver has to accept information from the mouse telling it 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 properly.

Obviously, these drivers will be very different.
Consequently, each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device.

(iii) Layer 3 : Device-independent I/O Software : Although 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 actually be done in the drivers, for efficiency or other reasons.

(iv) Layer 4 : User-Space I/O Software : Although most of the I/O software is within the operating system, a small portion of it consists of libraries linked together with user programs, and even whole programs running outside the kernel. System calls, including the I/O system calls, are normally made by library procedures.

Q.3(c) Explain the concept of Power Management. [5]

Ans.: 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 notebook computers with large bright screens. A rechargeable battery, in contrast, can store enough energy to power a notebook 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. Lithium ion batteries are even better, and may be recharged without first being fully drained, but their capacities are also severely limited.

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.

Q.3(d) Explain the conditions needed for a deadlock to occur. [5]

Ans.: **Deadlock :** Deadlock is set of blocked processes each holding a resource and waiting to acquire a resource held by another process.

Conditions for deadlock :

1. Mutual exclusion : Each resource is either currently assigned to exactly one process or is available.
2. Hold and wait : Process currently holding resources that were granted easier an request new resources.
3. No-preemption : Resources previously granted cannot be forcibly taken away from process. They must be explicitly released by the process holding them.
4. Circular wait : There must be a circular list of 2 or more processes, each of which is waiting for resource held by the next member of the chain.

Q.3(e) Explain how to prevent deadlocks by preventing the conditions for deadlock. [5]

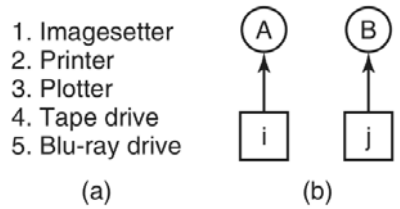
Ans.: Having seen that deadlock avoidance is essentially impossible, because it requires information about future requests, which is not known, how do real systems avoid deadlock? The answer is to go back to the four conditions to see if they can provide a clue. If we can ensure that at least one of these conditions is never satisfied, then deadlocks will be structurally impossible.

(i) Attacking the Mutual-Exclusion Condition : First let us attack the mutual exclusion condition. If no resource were ever assigned exclusively to a single process, we would never have deadlocks. For data, the simplest method is to make data read only, so that processes can use the data concurrently. However, it is equally clear that allowing two processes to write on the printer at the same time will lead to chaos. By spooling printer output, several processes can generate output at the same time. In this model, the only process that actually requests the physical printer is the printer daemon. Since the daemon never requests any other resources, we can eliminate deadlock for the printer.

(ii) Attacking the Hold-and-Wait Condition: The second of the conditions looks slightly more promising. If we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks. One way to achieve this goal is to require all processes to request all their resources before starting execution. If everything is available, the process will be allocated whatever it needs and can run to completion. If one or more resources are busy, nothing will be allocated and the process will just wait.

(iii) Attacking the No-Preemption Condition: Attacking the third condition (no preemption) is also a possibility. If a process has been assigned the printer and is in the middle of printing its output, forcibly taking away the printer because a needed plotter is not available is tricky at best and impossible at worst.

(iv) Attacking the Circular Wait Condition: Only one condition is left. The circular wait can be eliminated in several ways. One way is simply to have a rule saying that a process is entitled only to a single resource at any moment. If it needs a second one, it must release the first one. For a process that needs to copy a huge file from a tape to a printer, this restriction is unacceptable. Another way to avoid the circular wait is to provide a global numbering of all the resources, as shown in Figure below.



Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first a printer and then a tape drive, but it may not request first a plotter and then a printer.

Q.3(f) Describe (i) Two-phase locking (ii) Communication Deadlocks [5]

Ans.: (i) Two-phase locking

Although both avoidance and prevention are not terribly promising in the general case, for specific applications, many excellent special-purpose algorithms are known. As an example, in many database systems, an operation that occurs frequently is requesting locks on several records and then updating all the locked records. When multiple processes are running at the same time, there is a real danger of deadlock.

The approach often used is called two-phase locking. In the first phase, the process tries to lock all the records it needs, one at a time. If it succeeds, it begins the second phase, performing its updates and releasing the locks. No real work is done in the first phase.

If during the first phase, some record is needed that is already locked, the process just releases all its locks and starts the first phase all over. In a certain sense, this approach is similar to requesting all the resources needed in advance, or at least before anything irreversible is done. In some versions of two-phase locking, there is no release and restart if a locked record is encountered during the first phase. In these versions, deadlock can occur.

(ii) Communication Deadlocks

AII of our work so far has concentrated on resource deadlocks. One process wants something that another process has and must wait until the first one gives it up. Sometimes the resources are hardware or software objects, such as Blu-ray drives or database records, but sometimes they are more abstract. Resource deadlock is a problem of competition synchronization. Independent processes would complete service if their execution were not interleaved with competing processes. A process locks resources in order to prevent inconsistent resource states caused by interleaved access to resources. Interleaved access to locked resources, however enables resource deadlock. A semaphore is a bit more abstract than a Blu-ray drive but in this example, each process successfully acquired a resource (one of the semaphores) and deadlocked trying to acquire another one (the other semaphore). This situation is a classical resource deadlock.

While resource deadlocks are the most common kind, they are not the only kind. Another kind of deadlock can occur in communication systems (e.g., networks), in which two or more processes communicate by sending messages. A common arrangement is that process A sends a request message to process B, and then blocks until B sends back a reply message. Suppose that the request message gets lost. A is blocked waiting for the reply. B is blocked waiting for a request asking it to do something. We have a deadlock.

This, though, is not the classical resource deadlock. A does not have possession of some resource B wants, and vice versa. In fact, there are no resources at all in sight. But it is a deadlock according to our formal definition since we have a set of (two) processes, each blocked waiting for an event only the other one can cause. This situation is called a communication deadlock to contrast it with the more common resource deadlock. Communication deadlock is an anomaly of cooperation synchronization. The processes in this type of deadlock could not complete service if executed independently.

Q.4 Attempt the following (any THREE) [15]

Q.4(a) Describe the history of virtualization. [5]

Ans.: History of Virtualization

(i) As early as the 1960s IBM experimented with not just one but two independently developed hypervisors — SIMMON and CP-40.

(ii) CP-40 was reimplemented as CP-67 to form the control program of CP/CMS, a virtual machine operating system for the IBM System/360 Model 67.

(iii) Later, it was reimplemented again and released as VM/370 for the System/370 series in 1972.

(iv) The System/370 line was replaced by IBM in the 1990s by the System/390.

(v) In 2000, IBM released the z-series, which supported 64-bit virtual address spaces but was otherwise backward compatible with the System/360. All of these systems supported virtualization decades before it became popular on the x86.

(vi) The real Disco revolution started in the 1990s, when researchers at Stanford University developed a new hypervisor by that name and went on to found VMware, a virtualization giant that offers type 1 and type 2 hypervisors and now rakes in billions of dollars in revenue.

Q.4(b) Explain the three requirements for virtualization. [5]

Ans.: It is important that virtual machines act just like the real McCoy. In particular, it must be possible to boot them like real machines and install arbitrary operating systems on them, just as can be done on the real hardware. It is the task off the hypervisor to provide this illusion and to do it efficiently. Indeed, hypervisors should score well in three dimensions:

(i) Safety: the hypervisor should have full control of the virtualized resources.

(ii) Fidelity: the behavior of a program on a virtual machine should be identical to that of the same program running on bare hardware.

(iii) Efficiency: much of the code in the virtual machine should run without intervention by the hypervisor.

An unquestionably safe way to execute the instructions is to consider each instruction in turn in an **interpreter** (such as Bochs) and perform exactly what is needed for that instruction. Some instructions can be executed directly, but not too many. For instance, the interpreter may be able to execute an INC (increment) instruction simply as is, but instructions that are not safe to execute directly must be simulated by the interpreter. For instance, we cannot really allow the guest operating system to disable interrupts for the entire machine or modify the page-table mappings. The trick is to make the operating system on top of the hypervisor think that it has disabled interrupts, or changed the machine's page mappings.

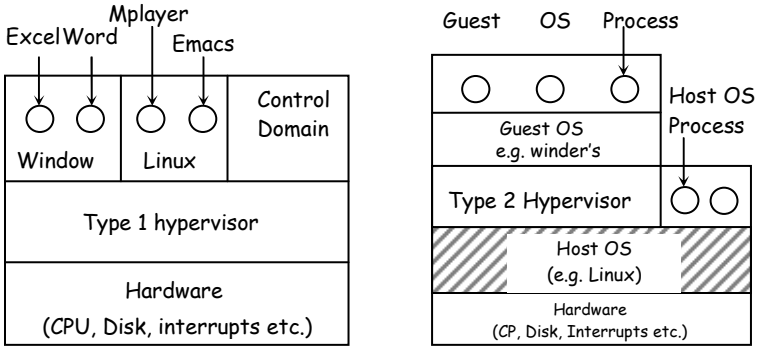
Q.4(c) With suitable diagrams, explain Type 1 and Type 2 Hypervisors. [5]

Ans.: Type 1 hypervisor :

- Runs on 'bare metal' directly on system hardware.
- A bare metal environment is a computer system in which a virtual machine is installed directly on hardware rather than within host OS.
- Offer multiple copies of real hardware called virtual machines.

Type 2 hypervisor :

- Runs on host operating system.
- Depends on host OS to allocate and schedule resources.
- Image file of guest OS is loaded on hard disk of Host OS.



Q.4(d) List the five essential characteristics of clouds. [5]

Ans.: Five essential characteristics of clouds:

1. **On-demand self-service:** Users should be able to provision resources automatically, without requiring human interaction.
2. **Broad network access:** All these resources should be available over the network via standard mechanisms so that heterogeneous devices can make use of them.
3. **Resource pooling:** The computing resource owned by the provider should be pooled to serve multiple users and with the ability to assign and reassign resources dynamically. The users generally do not even know the exact location of "their" resources or even which country they are located in.
4. **Rapid elasticity:** It should be possible to acquire and release resources elastically, perhaps even automatically, to scale immediately with the users' demands.
5. **Measured service:** The cloud provider meters the resources used in a way that matches the type of service agreed upon.

Q.4(e) Describe in brief Multicomputers i.e. Clustered Computers. [5]

Ans.: Clustered Computers

Like multiprocessor systems, clustered systems gather together multiple CPU's to accomplish computational work. Clustered systems differ from multiprocessor systems, however, in that they are composed of two or more individual systems-or nodes-jointed together. The definition of the term clustered is not concrete; many commercial packages wrestle with what a clustered system is and why one form is better than another. The generally accepted definition is that clustered computers share storage and are closely linked via a local-area network (LAN) or a faster interconnect, such as InfiniBand.

Clustering is usually used to provide high-availability service; that is, service will continue even if one or more systems in the cluster fail. High availability is generally obtained by adding a level of redundancy in the system. A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the other (over the LAN). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. The users and clients of the applications see only a brief interruption of service.

Clustering can be structured asymmetrically or symmetrically. In asymmetric clustering, one machine is in hot-standby mode while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server. In symmetric mode, two or more hosts are running applications and are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware. It does require that more than one application be available to run.

As a cluster consists of several computer systems connected via a network, clusters may also be used to provide high-performance computing environments. Such systems can supply significantly greater computational power than single-processor or even SMP systems because they are capable of running an application concurrently on all computers in the cluster. However, applications must be written specifically to take advantage of the cluster by using a technique known as parallelization, which consists of dividing a program into separate components that run in parallel on individual computers in the cluster. Typically, these applications are designed so that once each computing node in the cluster has solved its portion of the problem, the results from all the nodes are combined into a final solution.

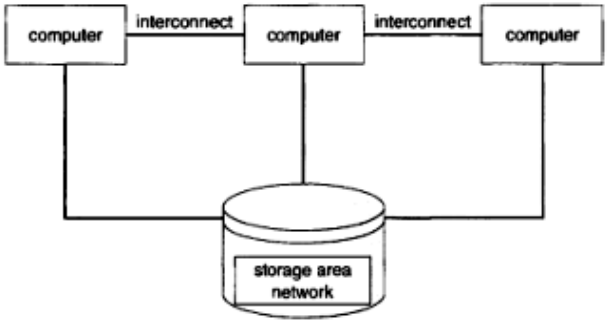


Fig.: General structure of a clustered system.

Q.4(f) Explain the concept of distributed systems. **[5]**

Ans.: A distributed system is a collection of physically separate, possibly heterogeneous computer systems that are networked to provide the users with access to the various resources that the system maintains. Access to a shared resource increases computation speed, functionality, data availability, and reliability. Some operating systems generalize network access as a form of file access, with the details of networking contained in the network interface's device driver. Others make users specifically invoke network functions. Generally, systems contain a mix of the two modes-for example FTP and NFS. The protocols that create a distributed system can greatly affect that system's utility and popularity.

A network, in the simplest terms, is a communication path between two or more systems. Distributed systems depend on networking for their functionality. Networks vary by the protocols used, the distances between nodes, and the transport media. TCP/IP is the most common network protocol, although ATM and other protocols are in widespread use. Likewise, operating system support of protocols varies. Most operating systems support TCP/IP, including the Windows and UNIX operating systems. Some systems support proprietary protocols to suit their needs. To an operating system, a network protocol simply needs an interface device-a network adapter, for example-with a device driver to manage it, as well as software to handle data. These concepts are discussed throughout this book.

Networks are characterized based on the distances between their nodes. A Local-area network (LAN) connects computers within a room, a floor, or a building. A wide-area network (WAN) usually links buildings, cities, or countries. A global company may have a WAN to connect its offices worldwide. These networks may run one protocol or several protocols. The continuing advent of new technologies brings about new forms of networks. For example, a metropolitan-area network (MAN) could link buildings within a city. BlueTooth and 802.11 devices use wireless technology to communicate over a distance of several feet, in essence creating a small-area network such as might be found in a home.

Q.5Attempt the following (any THREE) [15]

Q.5(a) Describe in brief important steps in history of Unix and Linux. [5]

Ans.: History of Unix and Linux:

(i) UNICS: Way back in the 1940s and 1950s, all computers were personal computers in the sense that the then-normal way to use a computer was to sign up for an hour of time and take over the entire machine for that period. M.I.T. joined forces with Bell Labs and General Electric (then a computer vendor) and began designing a second-generation system, MULTICS (MULTiplexed Information and Computing Service). Consequently, one of the other researchers at Bell Labs, Brian Kernighan, somewhat jokingly called it UNICS (UNiplexed Information and Computing Service).

(ii) PDP-11 UNIX: Thompson's work so impressed his colleagues at Bell Labs that he was soon joined by Dennis Ritchie, and later by his entire department. Two major developments occurred around this time. First, UNIX was moved from the obsolete PDP-7 to the much more modern PDP-11/20 and then later to the PDP-11/45 and PDP-11/70. The latter two machines dominated the minicomputer world for much of the 1970s

(iii) Portable UNIX: Now that UNIX was in C, moving it to a new machine, known as porting it, was much easier than in the early days when it was written in assembly language. A port requires first writing a C compiler for the new machine. Then it requires writing device drivers for the new machine's I/O devices, such as monitors, printers, and disks. Although the driver code is in C, it cannot be moved to another machine, compiled, and run there because no two disks work the same way.

(iv) Berkeley UNIX: One of the many universities that acquired UNIX Version 6 early on was the University of California at Berkeley. Because the full source code was available, Berkeley was able to modify the system substantially. Networking was introduced, causing the network protocol that was used, TCP/IP, to become a de facto standard in the UNIX world, and later in the Internet, which is dominated by UNIX-based servers.

(v) Standard UNIX: By the end of the 1980s, two different, and somewhat incompatible, versions of UNIX were in widespread use: 4.3BSD and System V Release 3. In addition, virtually every vendor added its own nonstandard enhancements.

(vi) MINIX: One property that all modern UNIX systems have is that they are large and complicated, in a sense the antithesis of the original idea behind UNIX. Even if the source code were freely available, which it is not in most cases, it is out of the question that a single person could understand it all anymore. This situation led one of the authors of this book (AST) to write a new UNIX-like system that was small enough to understand, was available with all the source code, and could be used for educational purposes. That system consisted of 11,800 lines of C and 800 lines of assembly code.

(vii)Linux: During the early years of MINIX development and discussion on the Internet, many people requested (or in many cases, demanded) more and better features, to which the author often said "No" (to keep the system small enough for students to understand completely in a one-semester university course). This continuous "No" irked many users. At this time, FreeBSD was not available, so that was not an option. After a number of years went by like this, a Finnish student, Linus Torvalds, decided to write another UNIX clone, named Linux, which would be a full blown production system with many features MINIX was initially lacking. The first version of Linux, 0.01, was released in 1991.

Q.5(b) With a suitable diagram, explain the structure of the Linux Operating System. [5]

Ans.: A Linux system can be regarded as a kind of pyramid, as illustrated in figure. At the bottom is the hardware, consisting of the CPU, memory, disks, a monitor and keyboard, and other devices. Running on the bare hardware is the operating system. Its function is to control the hardware and provide a system call interface to all the programs. These system calls allow user programs to create and manage processes, files, and other resources.

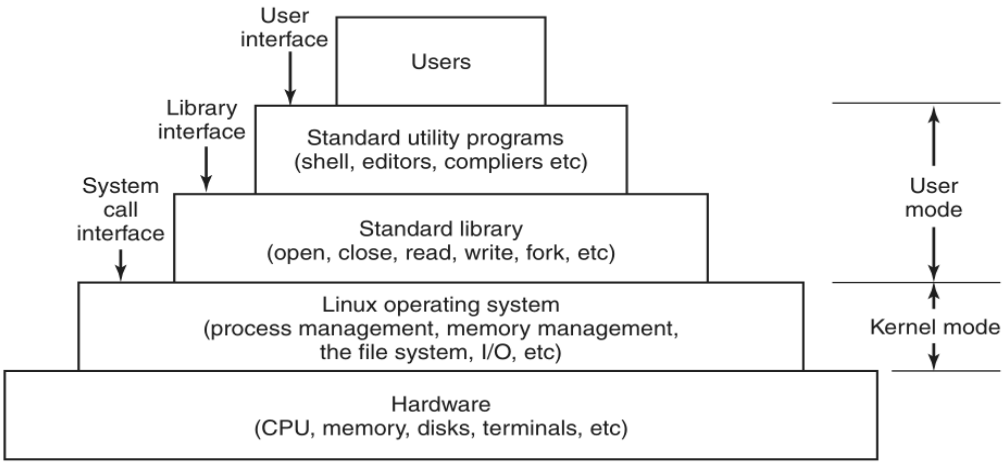


Fig.: The layers in a Linux system.

Q.5(c) Explain the basic concept of process management in Linux. [5]

Ans.: **Process Management in Linux**: Processes are created in Linux in an especially simple manner. The fork system call creates an exact copy of the original process. The forking process is called the parent process. The new process is called the child process. The parent and child each have their own, private memory images. If the parent subsequently changes any of its variables, the changes are not visible to the child, and vice versa.

The fact that the memory images, variables, registers, and everything else are identical in the parent and child leads to a small difficulty: How do the processes know which one should run the parent code and which one should run the child code? The secret is that the fork system call returns a 0 to the child and a nonzero value, the child's PID (Process Identifier), to the parent. Both processes normally check the return value and act accordingly.

A process can send signals only to members of its process group, which consists of its parent (and further ancestors), siblings, and children (and further descendants). A process may also send a signal to all members of its process group with a single system call.

Q.5(d) Describe the Android Operating System. [5]

Ans.: Android is a relatively new operating system designed to run on mobile devices. It is based on the Linux kernel—Android introduces only a few new concepts to the Linux kernel itself, using most of the Linux facilities you are already familiar with (processes, user IDs, virtual memory, file systems, scheduling, etc.) in sometimes very different ways than they were originally intended.

In the five years since its introduction, Android has grown to be one of the most widely used smartphone operating systems. Its popularity has ridden the explosion of smartphones, and it is freely available for manufacturers of mobile devices to use in their products. It is also an open-source platform, making it customizable to a diverse variety of devices. It is popular not only for consumer centric devices where its third-party application ecosystem is advantageous (such as tablets, televisions, game systems, and media players), but is increasingly used as the embedded OS for dedicated devices that need a graphical user interface (GUI) such as VOIP phones, smart watches, automotive dashboards, medical devices, and home appliances.

A large amount of the Android operating system is written in a high-level language, the Java programming language. The kernel and a large number of low-level libraries are written in C and C++. However a large amount of the system is written in Java and, but for some small exceptions, the entire application API is written and published in Java as well. The parts of Android written in Java tend to follow a very object-oriented design as encouraged by that language.

Q.5(e) Draw a table listing the various stages in the history of Windows. **[5]**

Ans.:

Year	MS–DOS	MS-DOS based Windows	NT-based Windows	Modern Windows	Notes
1981	1.0				Initial release for IBM PC
1983	2.0				Support for PC/XT
1984	3.0				Support for PC/AT
1990		3.0			Ten million copies in 2 years
1991	5.0				Added memory management
1992		3.1			Ran only on 286 and later
1993			NT 3.1		
1995	7.0	95			MS-DOS embedded in Win 95
1996			NT 4.0		
1998		98			
2000	8.0	Me	2000		Win Me was inferior to Win 98
2001			XP		Replaced Win 98
2006			Vista		Vista could not supplant XP
2009			7		Significantly improved upon Vista
2012				8	First Modern version
2013				8.1	Microsoft moved to rapid releases

Fig.: Major releases in the history of Microsoft operating systems for desktop PCs.

Q.5(f) List the six security properties that are implemented in Windows. **[5]**

Ans.: Six security properties implemented in Windows:

(i) Secure login with antispoofing measures.
(ii) Discretionary access controls.
(iii) Privileged access controls.
(iv) Address-space protection per process.
(v) New pages must be zeroed before being mapped in.
(vi) Security auditing.

❑ ❑ ❑ ❑ ❑