Unit-1:

INTRODUCTION TO OPERATING SYSTEM

An operating system is a program that manages the computer hardware. It also acts as an intermediary between the user & the computer hardware

What operating systems DO?

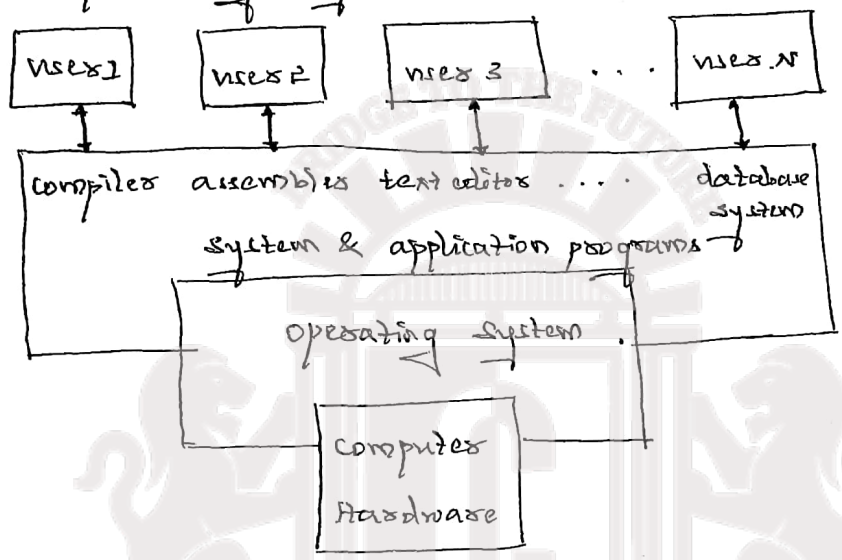


Fig: Abstract view of the components of a computer system.

Any computer system, will be having 4 basic components:

1. Hardware
2. operating system.
3. Application programs
4. users.

The hardware - the central processing unit (CPU), the memory, & the input/output (I/O) devices - provides the basic computing resources for the system.

The application programs - such as word processors, spreadsheets, compilers & web browsers define the ways in which these resources are used to solve users computing problems.

The operating system controls the hardware & coordinates its use among the various application programs for the various users.

Different views of OS:

The OS can be considered of having two views:

1. user view
2. system view.

User view:

The different types of computer systems available can be:

- Personal computers: Most of the users sit in front of a PC. A PC as name suggests, it is personal to the user & the entire hardware is dedicated to the user. Such a system is designed for one user to monopolize its resources.

In this case, the OS is designed mostly for ease of use, with some attention paid to performance & none paid to resource utilization.

- Mini computers / Main frame: A user sits at a terminal connected to a mainframe. other users are accessing the same computer through other terminals. the users share resources & exchange information.

The OS is designed to maximize resource utilization to ensure that all available CPU time, memory & I/O are used efficiently & that no individual user takes more than his/her fair share.

- Workstations: these are systems which are connected to the network of other workstations & servers forming a h/w of communication.

These users have dedicated resources at their disposal, but they also share resources such as networking & servers - files & print servers.

The OS is designed to compromise b/w individual usability & resource ~~utili~~ utilization.

System view:

From the computer's point of view, the OS is the program most intimately involved with the hardware. OS is viewed as a resource allocate.

Computer - System Organization

Before understanding the structure of OS & operations of OS, it is necessary to understand structure of a computer system.

Computer - System operation:

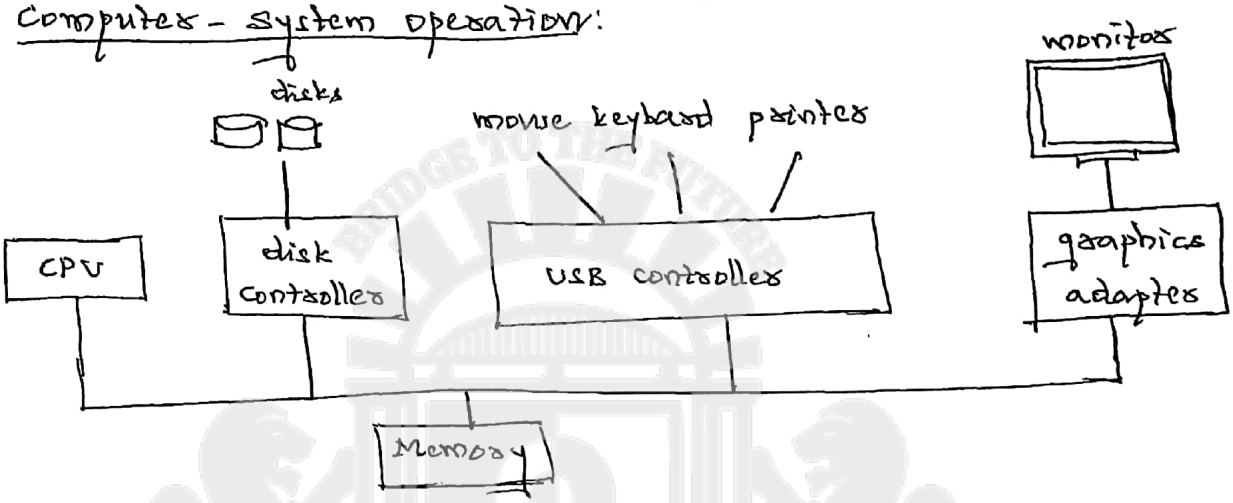


Fig: A modern computer system.

A modern general-purpose computer system consists of one or more CPU's and a number of device controllers connected through a common bus that provides access to the shared memory.

For a computer to start running - for instance, when it is powered up or rebooted - it needs to have an initial program to run.

A software known as bootstrap loader would pick up the OS & would place it in the RAM. The bootstrap Loader will be present on the ROM. ROM is also known as Firmware.

Once the O.S is completly booted, it waits for some event to occur. this nature of OS is known as interrupt driven approach.

Storage structure

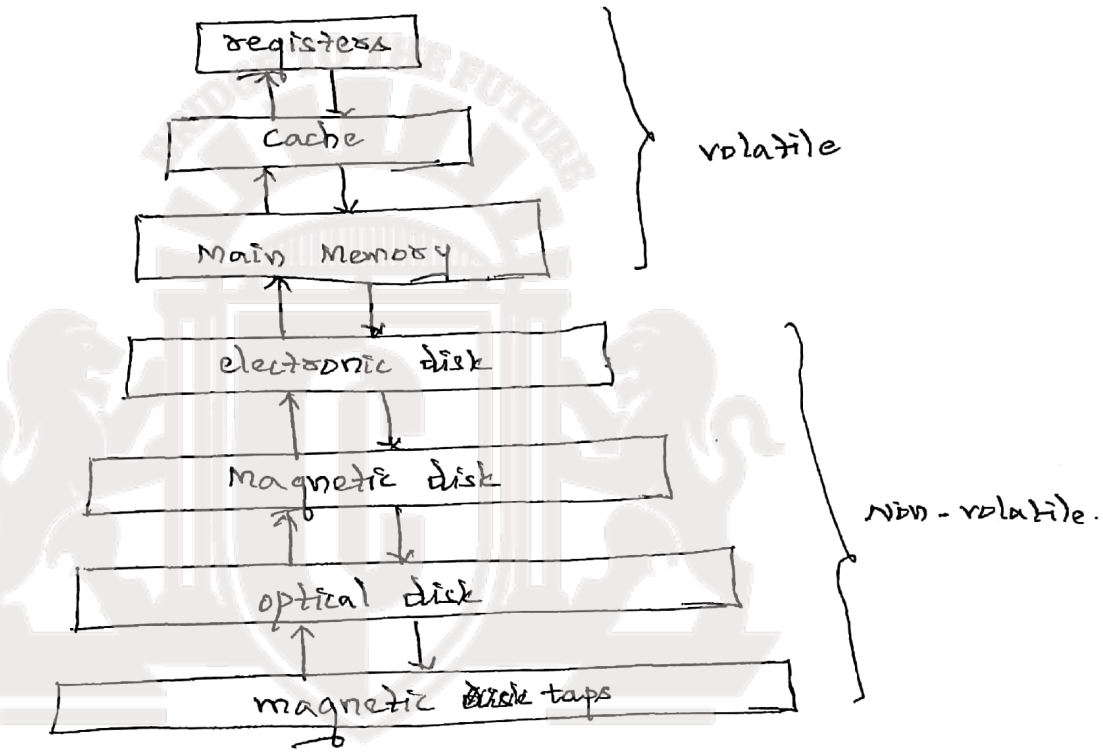


fig: storage-device hierarchy.

The CPU can load instructions only from memory, so any programs to run must be stored there. General-purpose computers run most of their programs from rewriteable memory, called main memory. Main memory commonly is implemented in a semiconductor technology called dynamic random-access memory (DRAM).

All forms of memory provide an array of words. Each word has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses.

The load instruction moves a word from main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to main memory.

We want the programs & data to reside in main memory permanently. This arrangement is not possible for the following two reasons.

1. Main memory is too small to store all needed programs & data permanently.
2. Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

Thus, most computer systems provide secondary storage as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently.

The wide variety of storage systems in a computer system can be organized in a hierarchy as shown in fig. according to speed & cost. The higher levels are expensive, but they are fast.

In addition to differing in speed & cost, the various storage systems are either volatile or nonvolatile. volatile storage loses its contents when the power to the device is removed.

In the hierarchy shown in fig. the storage systems above the electronic disk are volatile, whereas those below are nonvolatile. An electronic disk can be designed to be either volatile or nonvolatile.

I/O structure:

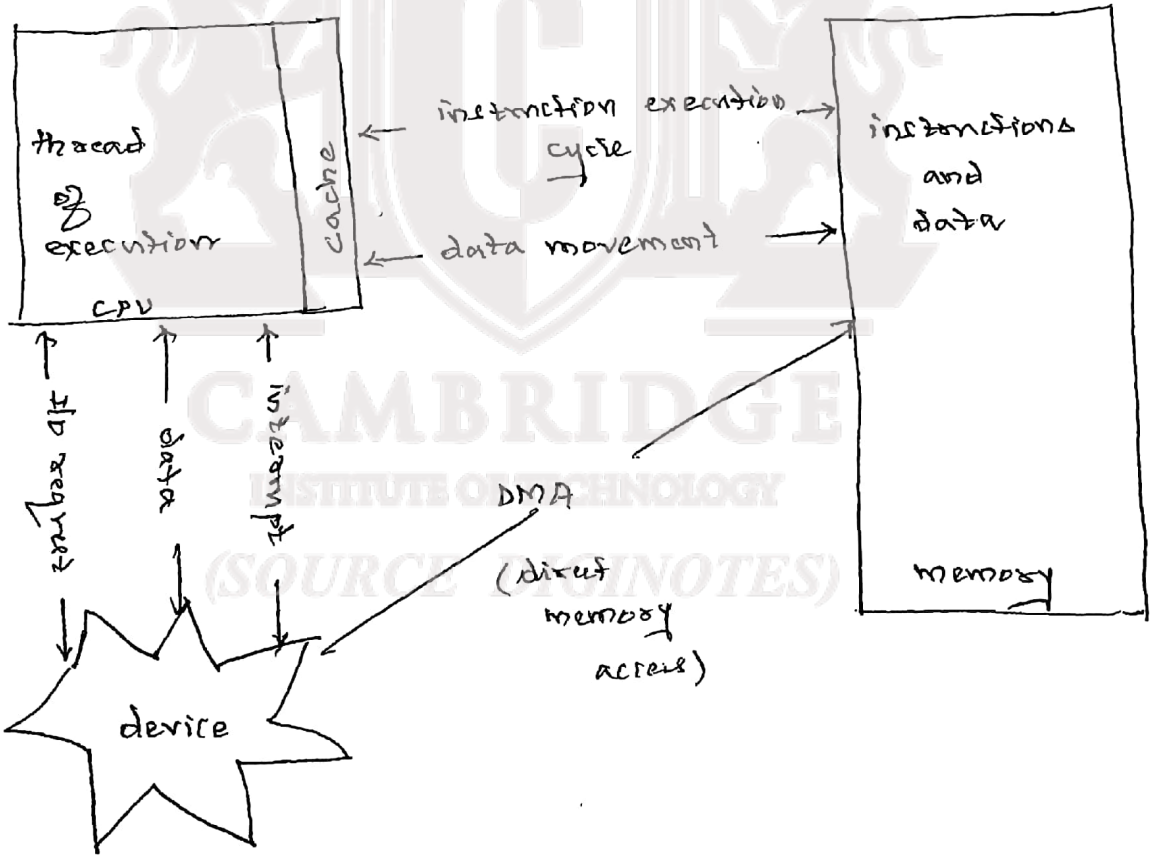
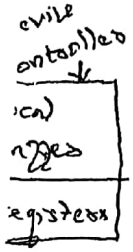


fig: How a modern computer system works.

A general-purpose computer system consists of CPU's and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. Depending on the controller, more than one device may be attached.



A device controller maintains some local buffer storage & a set of special-purpose registers. The device controller is responsible for moving the data b/w the peripheral devices that it controls & its local buffer storage. OS have a device driver for each device controller. This device driver understands the device controllers.

To start an I/O operation, the device driver loads the loads the appropriate registers within the device controller. The device controller, in turn, examines the contents of the register to determine what action to take. (such as read a character from the keyboard).

The controller starts the transfer of data from the device to local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation. The device driver then returns control to the operating system, possibly returning data or pointer to the data if the operation was a read. For other operations, it returns status information.

Computer-System Architecture

A computer system can be organized in a no. of diff. ways according to the no. of general-purpose processors used.

There are 3 types of architecture.

1. Single-processor system. ← single CPU + special purpose processors - low level components
2. Multi-processor system.
   - ↳ Asymmetric Multiprocessing
   - ↳ Symmetric Multiprocessing.
3. Clustered System.

Single-processor system:

Most systems use a single processor, On a single-processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.

Almost all systems have other special-purpose processors as well. They may come in the form of device-specific processors, such as disk, keyboard & graphics controllers.

All of these special-purpose processors run a limited instruction set, they are managed by the OS. The OS sends them information about their task & monitors their status.

Eg: a disk-controller microprocessor receives a sequence of requests from the main CPU & implements its own disk queue & scheduling algorithm.

PCs contain a microprocessor in the keyboard to convert the keystrokes into codes to be sent to the CPU. The special-purpose processors are low-level components built into the hardware & they do their jobs autonomously.

Multiprocessor systems:

- also known as parallel systems / tightly coupled systems.
- have two or more processors in close communication, sharing the computer bus & sometimes the clock, memory & ~~phesi~~ peripheral devices.

Three main advantages:

- Increased throughput
- Economy of scale
- Increased reliability.

Increased throughput:

By increasing the no. of processors, we expect to get more work done in less time. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead lowers expected gain from additional processors.

Economy of scale:

Multiprocessor systems can cost less than equivalent multiple single-processor systems, as they can share peripherals, mass storage & power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk & to have all the processors share them.

Increased reliability: If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

If we have ten processors & one fails, the each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slow rather than failing altogether.

Two types of Multiprocessor systems are

- asymmetric multiprocessor
- symmetric multiprocessor.

In asymmetric multiprocessing, each processor is assigned a specific task. A master processor controls the system, the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the salve processors.

In symmetric multiprocessing (SMP), in which each processor performs all tasks with in the OS. All the processors are peers, no master-slave relationship.

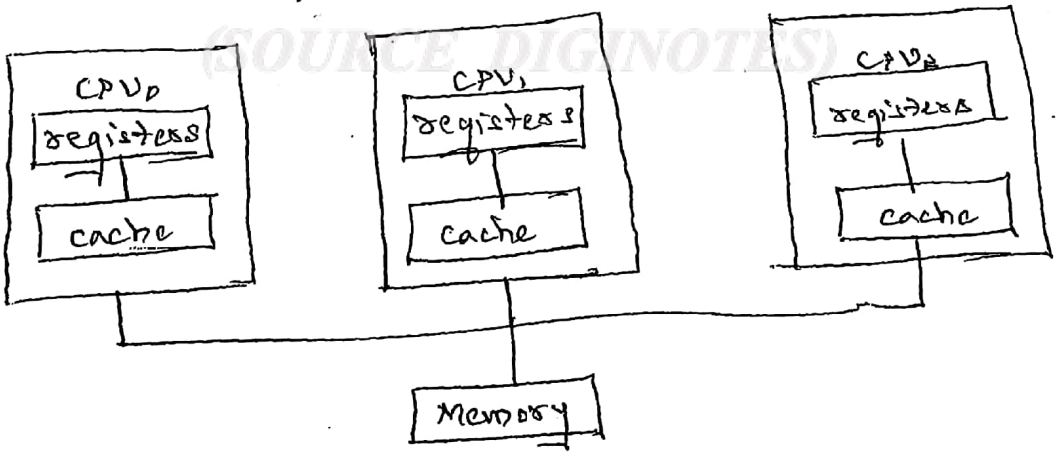


Fig: Symmetric multiprocessing architecture.

In SMP architecture, each processor has its own set of registers & private or local cache, all processors share physical memory. Eg: Solaries.

The benefit of this model is that many processes can run simultaneously without causing a significant deterioration of performance. However carefully control I/O to ensure that the data reach the appropriate processor. Also since the CPUs are separate, one may be sitting idle while other is overloaded, resulting in inefficiencies.

A recent trend in CPU design is to include multiple computing cores on a single chip. They can be more efficient than multiple chips with single cores because on-chip communication is faster than b/w chip communication.

In addition, one chip with multiple cores uses significantly less power than multiple single-core chips. they are well suited for server systems such as database & web servers.

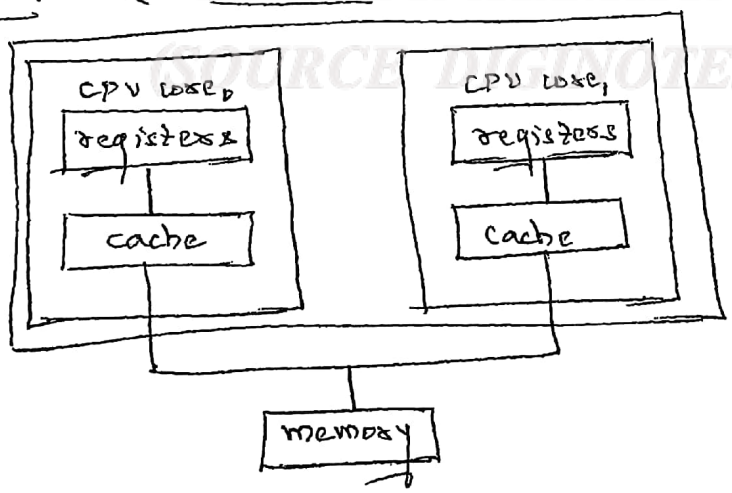
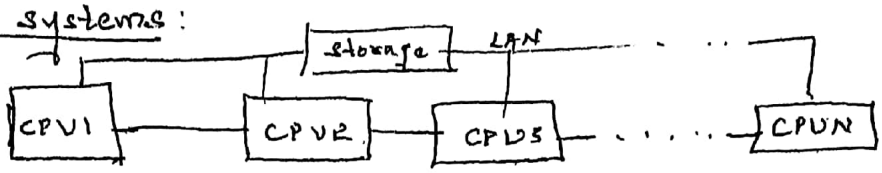


Fig: A dual-core design with two cores placed on the same chip.

Clustered systems:



↳ another type of multiple-CPU system.

↳ clustered systems gather together multiple CPU's to accomplish computational work.

↳ The multiprocessor system consists of two or more processors with in a single CPU where as clustered systems are composed of two or more individual systems or nodes - joined together.

↳ Clustered systems share storage & are closely linked via a local-area network.

Clustering is usually used to provide high-availability service. i.e. service will continue even if one or more systems in the cluster fail.

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 & restart the applications that were running on the failed machine.

Two modes of clustering are

- symmetric mode
- asymmetric mode

In asymetaic clustering, one machine is in hot-standby mode while the other is running the applications. The hot-standby host machine does nothing but monitors the active severs. If that sever fails, the hot-standby host beomes the active severs.

In symmetaic mode, two or more hosts are running applications & are monitoring each other. This mode is efficient, as it uses all of the available hardware.

Operating-System Structure:

- process
- Multiprogramming
- Multitasking or Time sharing systems.
- job scheduling, CPU scheduling, swapping, virtual memory.

Process:

→ A program loaded into memory & executing is called a process.

Multiprogramming:

A single program cannot keep the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running.

When two or more programs are in memory at the same time, sharing the processor is referred to as multiprograming system.

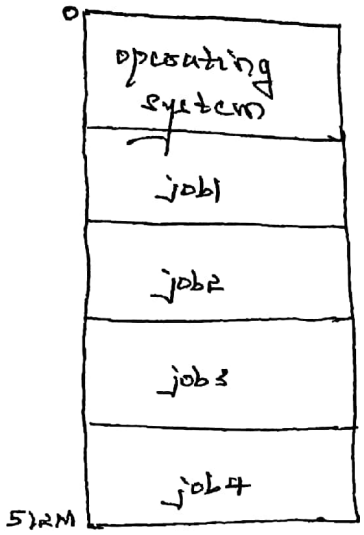


Fig: Memory Layout for a multiprogramming system.

The OS keeps several jobs in memory simultaneously, since in general, main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the **job pool**. This job pool consists of all process residing on disk awaiting allocation of main memory.

The OS picks & begins to execute one of the jobs in memory. If the job has to perform some I/O operation to complete its execution, the CPU may have to wait.

Advantages of multiprogramming:

→ the CPU is never idle.
→ High CPU utilization.
→ multiple tasks can be executed simultaneously.

Disadvantages of multiprogramming:

→ When CPU switches from one job to another, the information about the previous job has to be maintained in order to resume the execution from point where it had left.
→ many complex algorithms has to be implemented.
Eg: CPU scheduling, swapping, virtual memory management etc.

In a non-multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the OS simply switches to & executes another job. When that job needs to wait, the CPU is switched to another job & so on. Eventually, the first job finishes waiting & gets the CPU back. As long as atleast one job needs to execute, the CPU is never idle.

Multitasking or Time sharing system

The extension of multiprogramming is multitasking or time sharing system. Each job that is in the main memory would get some time allocated known as time slice. The CPU executes a job only during this time slice, when the time slice expires, the CPU would switch to another job & so on. The switches occurs so frequently that the users can interact with each prog. while it is running.

Time sharing requires an interactive / hands-on computer system, which provides direct communication b/w the user & the system, where as no user interaction possible in multiprogrammed system.

Time sharing & multiprogramming require that several jobs be kept simultaneously in memory. If several jobs are ready to be brought into memory, & if there is not enough room for all of them, then the system must choose among them. Making this decision is job scheduling.

When the operating system selects a job from the job pool, it loads that job into memory for execution. Having several programs in memory at the same time requires some form of memory management. If several jobs are ready to run at same time, the system must choose among them. Making this decision is CPU scheduling.

In a time-sharing system, the OS must ensure reasonable response time, which is sometimes accomplished through swapping, where processes are swapped in & out of main memory to the disk.

A common method for achieving this goal is virtual memory, a technique that allows execution of a process that is not completely in memory. The main advantage of the virtual memory scheme is that it enables users to run programs that are larger than actual physical memory.

Operating-system Operations:

The modern operating systems are interrupt driven. Events are almost signaled by the occurrence of an interrupt or a trap.

A trap or an exception is a software-generated interrupt caused either by an error (eg: division by zero or invalid memory access) or by a specific request from a users prog. that an OS service be performed.

For each type of interrupt, separate segments of code in the OS determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

Dual-Mode operation:

In order to ensure the proper execution of the OS, we must be able to distinguish b/w the execution of OS code & user-defined code.

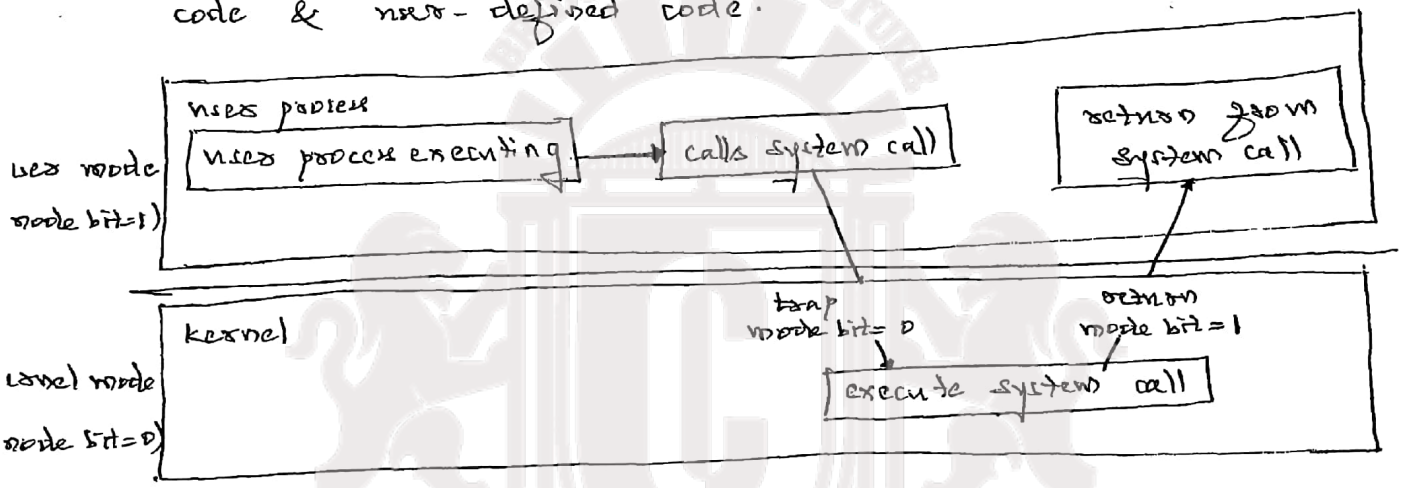


Fig: Transition from user to kernel mode.

Two modes of operation:

1. user mode.
2. kernel mode (also called supervisor mode, system mode or privileged mode).

A bit, called mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).

When the mode bit = 0, the tasks are executed on behalf of the kernel. When the mode bit = 1, then the tasks are executed on behalf of the user.

At system boot time, the hardware starts in kernel mode. The OS is then loaded & starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode. Thus whenever the OS gain control of the computer, it is in kernel mode. The system always switches to user mode before passing control to a user program.

The dual mode of operation provides us with the means for protecting the OS from errant users. This is accomplished by designing some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal & traps it to the OS.

The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/O control, timer management & interrupt management.

Timer:

The OS should have control over the CPU, it can't allow a user program to get stuck in an infinite loop or to fail to call system services & never return control back. Timer is used to accomplish this goal. A timer can be set to interrupt the computer after a specified period. The period may be fixed (eg: 1/60 sec) or variable (eg: from 1 ms to 1s).

A timer is used to prevent a user prog. from running too long. A simple technique is to initialize a counter with the amount of time that a prog. is allowed to run. A prog. with a 7-minute time limit, for eg. would have its counter initialized to 420. Every second, the timer interrupts & the counter is decremented by 1. As long as the counter is positive, control is returned to the user program. When the counter becomes negative, the OS terminates the prog. for exceeding the assigned time limit.

Process Management:

A program does nothing unless its instructions are executed by a CPU. A program in execution, is a process. Eg: A word-processing prog. being run by an user is a process. A system task such as sending output to a printer, can also be a process.

A process needs certain resources - including CPU time, memory, files & I/O devices, to accomplish its task. These resources are either given to the process when it is created or allocated to it while it is running.

Eg: Consider a process whose function is to display the status of a file on the screen of a terminal. The process will be given as an input the name of the file & will execute appropriate instructions & system calls to obtain and display on the terminal desired information. When the process terminates, the OS will reclaim any reusable resources.

The OS is responsible for the following activities in connection with process management.

1. Scheduling processes & threads on the CPU's
2. Creating & deleting both user & system processes.
3. Suspending & resuming processes.
4. Providing mechanisms for process synchronization.
5. providing mechanisms for process communication.

Memory Management:

The main memory is central to the OS. Main memory is a large array of words or bytes. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU & I/O devices.

The CPU reads instructions from main memory during the instruction-fetch cycle & both reads & writes data from main memory during data-fetch cycle.

For a prog. to be executed, it must be mapped to absolute addresses and loaded into memory. As the prog. executes, it accesses prog. instructions & data from memory by generating these absolute addresses. Eventually prog. terminates, its memory space is declared available & the next prog. can be loaded & executed.

To improve both the utilization of the CPU & the speed of the computer's response to its users, computers must keep several programs in the memory, creating need for memory management.

The responsibilities of OS in connection with memory management are:

1. Keeping track of which parts of memory are currently being used & by whom.
2. Deciding which processes & data to move into & out of memory.
3. Allocating & deallocating memory space as needed.

Storage Management:

To make the computer system convenient for users, the OS provides a uniform, logical view of information storage. The OS abstracts from the physical properties of its storage devices to define a logical storage unit, the file.

File-System Management:

A file is a collection of related information. The files represent programs (both source & object forms) & data. Data files may be numeric, alphabetic, alphanumeric or binary. Files may be free-form (eg: text files) or they may be formatted rigidly eg: fixed fields.

The OS implements the file by managing mass-storage media such as tapes & disks and the devices that control them. Files are normally organized into directories to make them easier to use.

The OS is responsible for the following activities in connection with file management.

1. Creating & deleting files
2. Creating & deleting directories to organize files
3. Supporting primitives for manipulating files & directories
4. Mapping file onto secondary storage
5. Backing up files on stable (nonvolatile) storage media.

Mass-storage Management:

Main memory is too small to accommodate all data & programs & because data that it holds are lost when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disk for this purpose.

Most programs including compilers, assemblers, word processors, editors & formatters are stored on a disk until loaded into main memory & then use the disk as both the source & destination of their processing. Hence the proper management of disk storage is important.

The responsible of OS in connection with Mass-Storage or disk management are:

1. Free-space management
2. Storage allocation
3. Disk Scheduling.

Caching:

The prog. that needs to be executed has to be present on the main memory. The CPU picks up the instruction of the program one after the another from main memory & executes it. As the instructions are used, it is copied into a faster storage system known as cache.

When a particular information is required, the CPU checks whether it is available on the cache. If it is available, it is accessed directly. If it is not then the informations are fetched from main memory The cache management is very important as its size is limited.

The programmer or compiler implements the register-allocation & register-replacement algorithms to decide which information to keep in registers & which to keep in main memory. There are also caches that are implemented totally in hardware.

Performance of various levels of storage:

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size.	<1KB	<16MB	<64GB	>100GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off chip. CMOS SRAM	CMOS DRAM.	magnetic disk
Access time(ns)	0.25 - 0.5	0.5 - 25	80 - 250	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5000 - 10,000	1000 - 5000	20 - 150
Managed by	compiled	hardware	operating system.	operating system.
Backed by	Cache	main memory	disk	CD or tape

The movement of information b/w levels of a storage hierarchy may be explicit or implicit. For instance, data transfer from cache to CPU & registers is usually a hardware function, with no OS intervention. In contrast, transfer of data from disk to memory is usually controlled by the OS.

In a hierarchical storage structure, the same data may appear in different levels of the storage system.

Example:

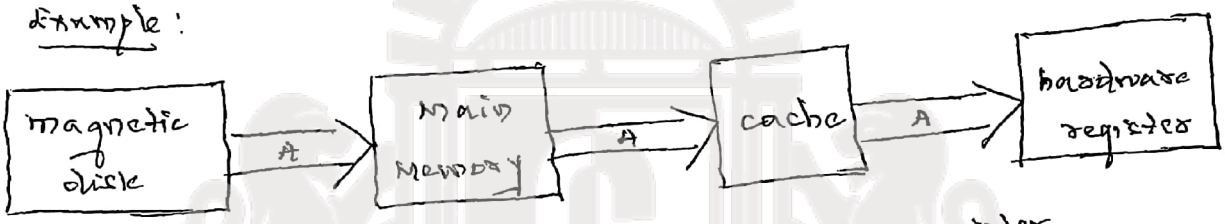


Fig: Migration of integer A from disc to register.

Suppose that an integer A that is to be incremented by 1 is located in file B, and file B resides on magnetic disk. The increment operation proceeds by first issuing an I/o request to copy the disk block on which A resides to main memory. This operation is followed by copying A to the cache & to an internal register. Thus, the copy of A appears in several places i.e. on magnetic disk, main memory, cache & internal register. The value of A becomes the same only after the new value of A is rewritten from the internal register back to magnetic disk.

Cache Coherency:

In a multiprocessor environment, in addition to the internal registers, each of the CPUs also contain a local cache. In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPUs can all execute concurrently, we must make sure that an update to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called cache coherency, it is usually a hardware problem.

I/O systems:

- One of the purpose of an OS is to hide the peculiarities of specific hardware devices from the user.
- Eg: In UNIX, the peculiarities of I/O devices are hidden from the bulk of the OS itself by the I/O subsystem.

The I/O subsystem consists of several components:

1. A memory-management component that includes buffering, caching & spooling.
2. A general device-driver interface
3. Drivers for specific hardware devices.

Protection & security

In an multiprocessor, multitasking & multiprogramming environment, it is very important to ensure that files, CPU & other resources should be operated by only those process that have a previliged access.

The mechanism for controlling the access of processes or users to the resources defined by a computer system is known as Protection.

By increasing the protection, the reliability of the system can be increased. this is done by detecting the errors at the interface.

A system can have adequate protection but still be prone to failure and allow inappropriate access. Consider a user whose authentication information is stolen. Her data could be copied or deleted, even though file & memory protection are working. It is the job of security to defend a system from external & internal attack.

Distributed Systems:

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.

Distributed systems increases computation speed, functionality, data availability & reliability.

A network is a communication path b/w two or more systems. Distributed systems rely on networking for their functionality. Networks vary by the protocols used, the distance b/w nodes and the transport media. Eg: TCP/IP is the most common network protocol.

Networks are characterized based on the distance b/w their nodes.

1. Local area network (LAN): It connects computers within a room, a floor or a building.
2. Wide area network (WAN): It links buildings, cities or countries.
3. Metropolitan-area network (MAN): could link buildings with in a city.

A network operating system is an OS that provides features such as file sharing, message passing, sharing of devices etc.

Special-purpose systems

- Real-time embedded systems
- Multimedia systems
- Handheld systems.

Real-time Embedded systems :

Embedded systems / computers are the most prevalent form of computers in existence. These devices are found everywhere, from car engines & manufacturing robots to DVDs & Microwave ovens

They tend to have very specific tasks. The systems they run on are usually primitive & so the OS provides limited features. Usually they have little or no user interface, preferring to spend their time monitoring & managing hardware devices such as automobile engines & robotic arms.

Embedded systems almost always run real-time OS. A real-time system is used when rigid time requirements have been placed on the operation of a processor.

A real-time system has well-defined, fixed time constraints Processing must be done within the defined constraints or the system will fail.

A hard real-time system is dedicated to processing real-time applications & provably meets the response requirement of an application under all conditions.

A soft real-time system makes the best effort to meet the response requirement of a real-time application but can't guarantee that it will be able to meet it under all conditions.

Multimedia systems:

Most OS are designed to handle conventional data such as text files, programs, word-processing documents & spreadsheets. However, a recent trend in technology is the incorporation of multimedia data into computer systems.

Multimedia data consists of audio & video files as well as conventional files. These data differs from conventional data in that multimedia data such a frames of video - must be delivered according to certain time restrictions. (Eg: 30 frames per sec).

Multimedia consists of wide range of applications like. video conferencing, Audio conferencing, webcasts etc.

Handheld systems:

In recent days handheld systems are getting more prominances. Handheld systems include PDAs i.e. personal digital assistants, cellular phones, Pocket-PCs, Palm Pcs.

Developers of handheld systems & applications face many challenges. Because of their size, most handheld systems / devices have small amounts of memory, slow processors, & small display screen.

The amount of physical memory would range from 1MB to 1GB (depending on the devices). As a result, the OS & applications must manage memory efficiently.

The second issue concerned with handheld systems are speed of processor used in the devices. It is very difficult to have a faster processor because they would require more power which intern Larger batteries. Most handheld devices are smaller, slower processors that consume less power. Therefore os & applications must be designed not to put load on processor.

Another issue concerned with handheld devices is I/O. As the screen / display of handheld systems are very small, lots of techniques have to be implemented os employed to output the results.

One approach for displaying the contents in web pages is web clipping, where only a small subset of the web page is delivered & displayed on handheld device.

Computing Environments

- Traditional computing
- client-server computing
- Peer-to Peer Computing
- Web-Based Computing

Traditional Computing:

The typical office environment is an example for traditional computing. The environment consisted of PCs connected to a network with servers providing file & print services. Remote access was awkward & portability was achieved by use of laptop computers.

At home, most users had a single computer with a slow modem connection to the office internet.

When computers came into existance for the first time, most of OS were batch systems or interactive. Batch systems processed jobs in bulk with predetermined input. Interactive systems waited for the inputs from users.

Client-server computing

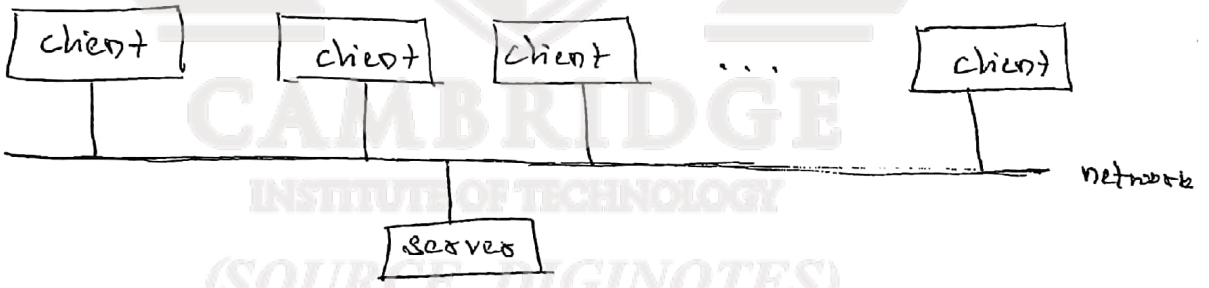


Fig: General structure of a client-server system.

A client-server system is a specialized form of distributed systems. Two or more systems are connected, one of them is likely to form a server system & the other behaves as a client system.

The system which requests for some service will be the client system. The system which responds to the request & services the request is known as server system.

Servers systems can be broadly categorized as

1. Compute - servers system
2. File - server system.

The compute - servers system provides an interface to which a client can send the request to perform an action such as reading a data, executing a prog. in response the server executes the action & sends the back the result to client.

Eg: A servers running a database.

The file - servers system provide a file - system interface where client can create, update, read & delete files. An example of such a system is a web servers that delivers file to clients running web browsers.

Peer - to - peer Computing

Another structure for a distributed system is the peer - to - peer (P2P) system. In this model, clients & servers are not distinguished from one another instead all nodes within the system are considered peers & each may act as either a client or a server, depending on whether it is requesting or providing a service.

Peer-to-peer systems offer an advantage over traditional client-server system. In a client server system, the server is a bottleneck but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.

To participate in a peer-to-peer system, a node must first join the network of peers. Once a node has joined the network, it can begin providing services to & requesting service from other nodes in the network.

Determining what services are available is accomplished in one of two general ways

- When a node joins a n/w, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contact this centralized lookup service to determine which node provides the service. The remainder of the communication takes place b/w the client & the service provider.
- A peer acting as a client must first discover what node provides a desired service by broadcasting a request for the service to all other nodes in the n/w. The nodes providing that service response to the peer making the request. To support this approach, a discovery protocol must be provided that allows peers to discover services provided by other peers in the n/w.

SYSTEM STRUCTURES:

Operating system services:

An OS provides an environment for execution of programs. It provides certain service to programs and to the users of those programs.

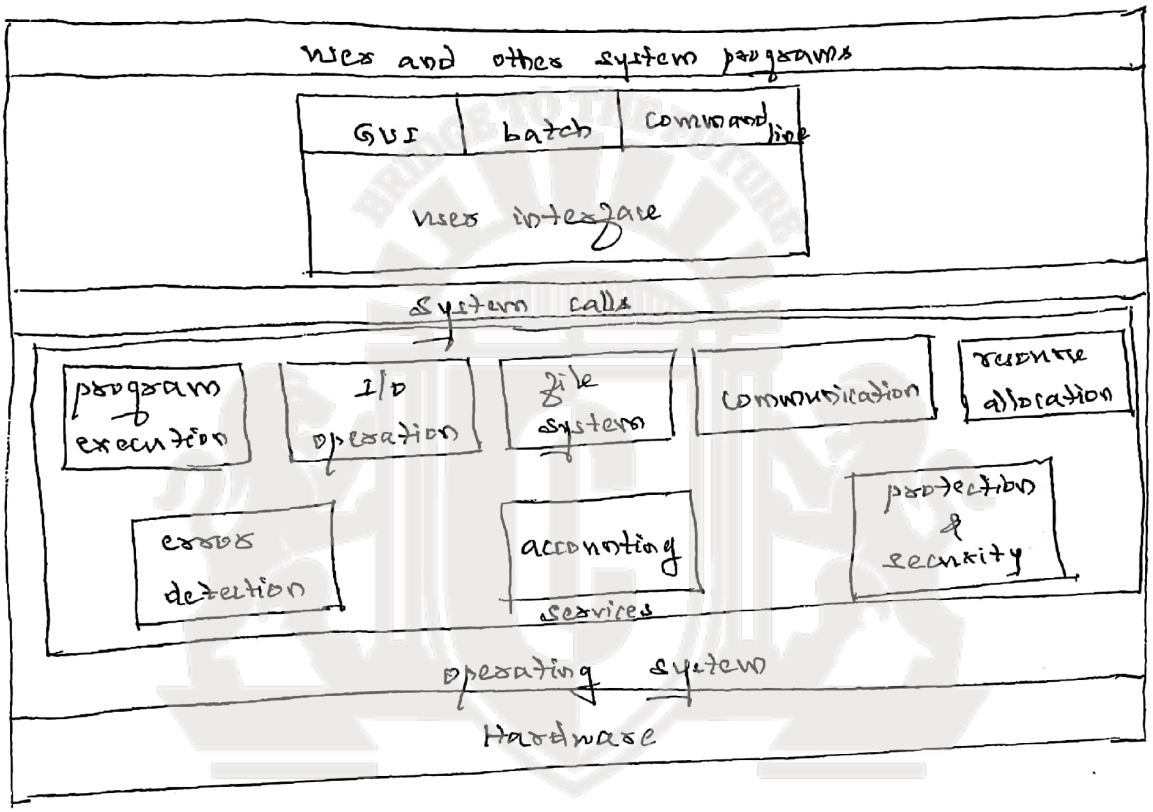


Fig: A view of OS services.

Operating-system provides following services to the users:

1. Users interface: Almost all OS have users interface. This interface can take several forms.
   - command-line interface (CLI) - which uses text commands & a method for entering them.
   - Batch interface: - use commands & directives, commands are entered into files & those files are executed.

- graphical user interface (GUI): It a window system with a pointing device to direct I/O, choose from menus & make selections & a keyboard to enter text.

2. Program execution: The system must be able to load a prog. into memory & to run that program. The prog. must be able to end its execution, either normally or abnormally.

3. I/O operation: A running prog. may require I/O, which may involve a file or an I/O device. For specific devices, special functions may be desired.

4. File-system manipulation: The prog. or the user of the OS might want to do some operation on the file like creating, deleting, saving a file & so on. The OS must provide a means to manage file system.

5. Communication: There are many circumstances in which two or more processes need to communicate/ exchange information. Communication may occur b/w the processes of same computer or processes b/w the network. Communication may be implemented via shared memory or through message passing.

6. Errors detection: The OS needs to be aware of possible errors that may occur in the CPU, memory hardware, I/O devices or user programs. For each type of errors, the OS should take the appropriate action to ensure correct & consistent computing.

Another set of OS functions exist not for helping the user but rather for ensuring the efficient operation of the system itself.

1. Resource allocation: When there are multiple users or multiple jobs running at same time, resources must be allocated to each of them. Many diff. types of resources are managed by OS. Some (such as CPU cycles, main memory & file storage) may have special allocation code whereas others (such as I/O devices) may have much more general request & release code.

2. Accounting: We want to keep track of which users use how much & what kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.

3. Protection & security: The owners of information stored in a multiuser or networked computer system may want to control use of that information. When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the OS itself.

Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.

System calls:

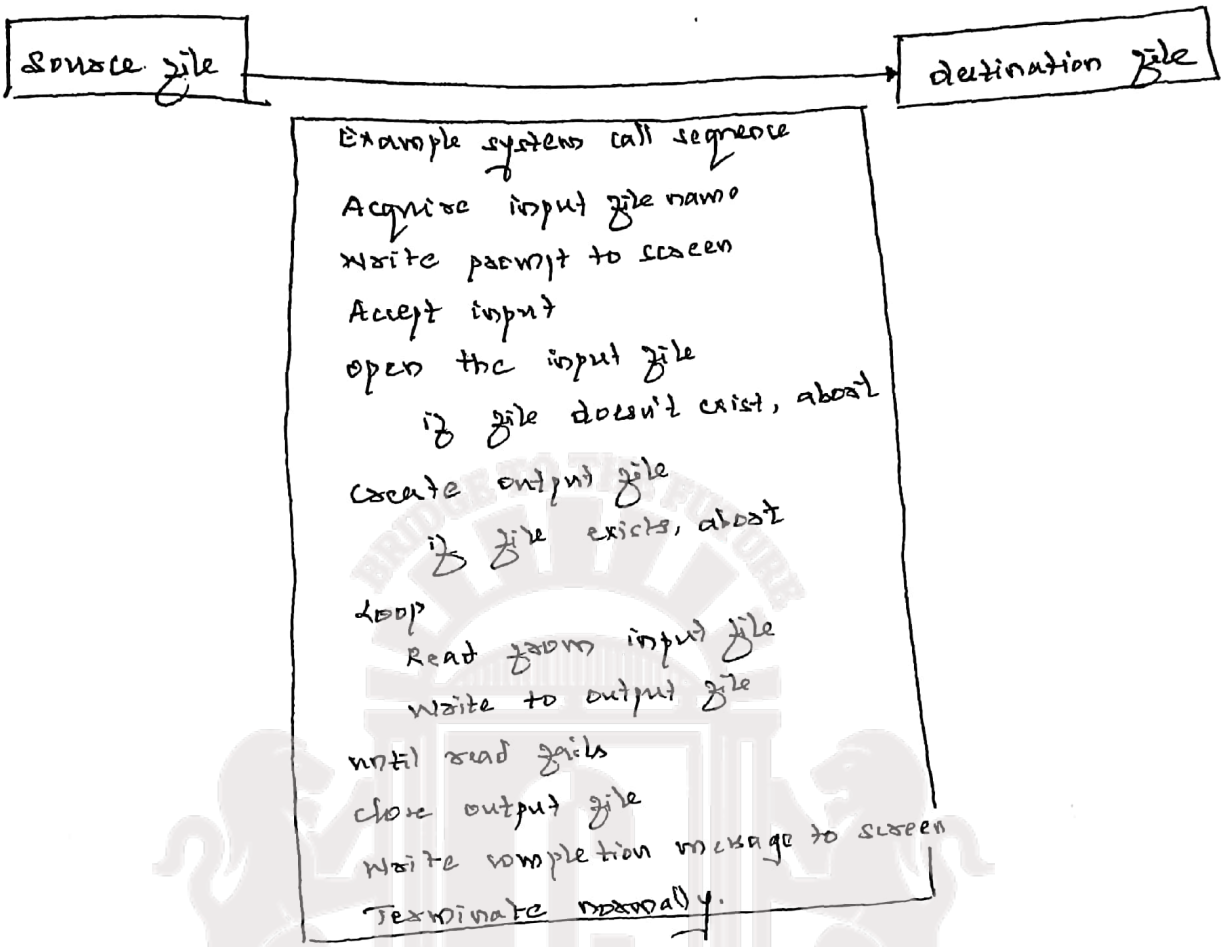
System calls provide an interface to the services made available by an operating system. these calls are generally available as routines written in C & C++

Example to illustrate how system calls are used:

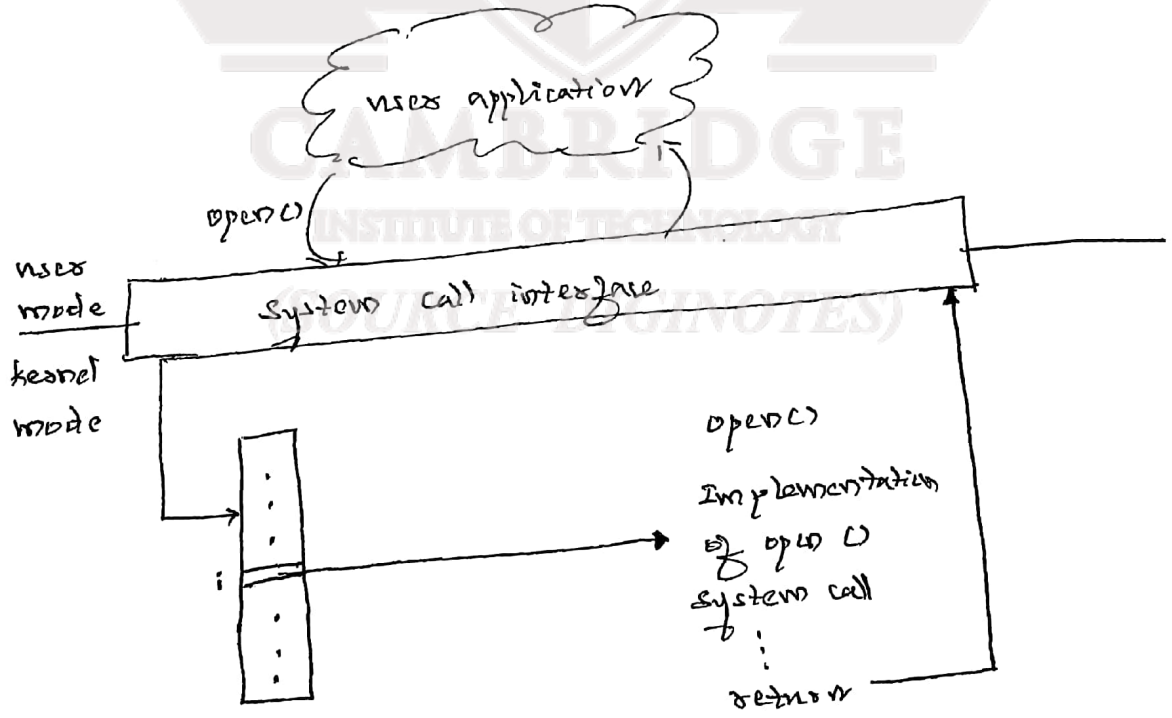
Writing a simple prog. to read data from one file & copy them to another file. the first input that the prog. will need is the names of the two files: input file & o/p file

Once the two file names are obtained, the prog. must open the input file & create the output file. Each of these operations requires another system call. there are also possible error conditions for each operation. when the prog. tries to open the i/p file, it may find that there is no file of that name or that file is protected against access. In these cases, the prog. should print a message on the console (another system calls) & then terminate abnormally. (another system call). If the i/p file exists, then new o/p file is created.

When both the files are set up, data is read & written to the o/p file with the help of loop (a system call). Finally, the prog. may close both files (a system call) & terminate normally (a system call).



Eg: Example of how system calls are used.



Eg: the handling of user application invoking the open() system call.

Types of system calls

System calls can be grouped roughly into six major categories:

1. **Process Control**
   - end, abort
   - load, execute
   - create process, terminate process
   - get process attribute, set process attributes
   - wait for time
   - wait event, signal event
   - allocate & free memory.

2. **File management:**
   - create file, delete file
   - open, close
   - read, write, reposition
   - get file attributes, set file attributes.

3. **Device Management**
   - request device, release device
   - read, write, reposition
   - get device attributes, set device attributes
   - logically attach or detach device.

4. **Information maintenance:**
   - get time or date, set time or date
   - get system data, set system data
   - get process, file or device attributes
   - set process, file or device attributes.

5. Communication:
   - create, delete communication connection
   - send, receive messages
   - transfer status information
   - attach or detach remote devices.

6. Protection:
   - set permission, get permission.
   - allow user, deny user.

Process Control:

A running prog. needs to be able to halt its execution either normally (end) or abnormally (abort). load & store system calls are used by a process or job to execute another program.

Get process attribute & set process attribute system calls are used to determine the attributes of process like process name, process id, process size, execution time of process etc.

A process may have to wait for other process to complete execution. To make a process to wait for a certain amount of time to pass wait time system call is used. or to wait for a specific event occurs system call wait event is used.

File Management:

All the file related activities like create, delete, open, close, update etc are achieved using system calls.

File attributes include the file name, file type, protection codes, accounting information & so on. To know this attributes get file attribute & set file attribute system calls are used.

Device Management:

A process may need several resources to execute - main memory, disk drives, access to files & so on. If the resources are available, they can be granted & control can be returned to the user process.

If the process is running in multiuser environment, it requests the resources before the execution & once the execution is completed it release the resources. These are achieved using request device & release device systems calls.

Once the device has been allocated, read, write & reposition system calls can be used, just as with files.

Information Maintenance:

The operating system keeps information about all its processes. System calls are used to access this information.

Example: Most systems have a system call to return the current time & date, no. of current users, the version number of the OS, amount of free memory or disk space etc.

Communication:

There are two common models of interprocess communication: the message passing model & the shared memory model.

In message passing model, the communicating processes exchange messages with one another to transfer information. Before communication can take place, a connection must be opened. the recipient process must give its permission for communication to take place with an accept connection call. the client & server can exchange messages by using read message & write message system calls. The close connection call terminate the communication.

In the shared memory model, processes use the shared memory create & shared memory attach system calls to create & gain access to regions of memory owned by other processes.

Protection:

Protection provides a mechanism for controlling access to the resources provided by a computer system. system calls providing protection include get permission & set permission, which manipulate the permission setting of resources such as files & disks.

the allow user & deny user system calls specify whether particular user can or cannot be allowed to access to certain resources.

OS Design & Implementation

- Design Goals
- Mechanisms & policies
- Implementation.

Design Goals:

The first problem in designing a system is to define goals & specifications. At the highest level, the design of the system will be affected by the choice of hardware & the type of system : batch, time-shared, single-user, multiuser, distributed, real-time or general purpose.

The requirements may be much harder to specify. Requirements can be grouped into two types. They are

1. user goals
2. system goals.

Users desire certain obvious properties in a system. The system should be convenient to use, easy to learn & to use, reliable, safe & fast.

A similar set of requirements can be defined by those people who must design, create, maintain & operate the system. The system should be easy to design, implement & maintain & it should be flexible, reliable, error free & efficient.

Mechanisms & Polices

One important principle is the separation of policy from mechanism. Mechanism determines how to do something, polices determine what will be done. Eg: the timer construct is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision.

The separation of policy & mechanism is important for flexibility. Policies are likely to change across places or over time.

Implementation:

Once the OS is designed, it must be implemented. Traditionally, OS have been written in assembly language Now, however, they are most commonly written in higher-level language such as C or C++.

The advantages of using a high-level language - the code can be written faster, is more compact & is easier to understand & debug.

Operating-System Structure:

- Simple structure
- Layered Approach
- Microkernels
- Modules

simple structure (Monolithic structure).

These os are small, simple & didn't have well defined structure. MS-DOS is an example. It was originally developed by a few people who had no idea that it would become so popular. It was written to provide the most functionality in the least space.

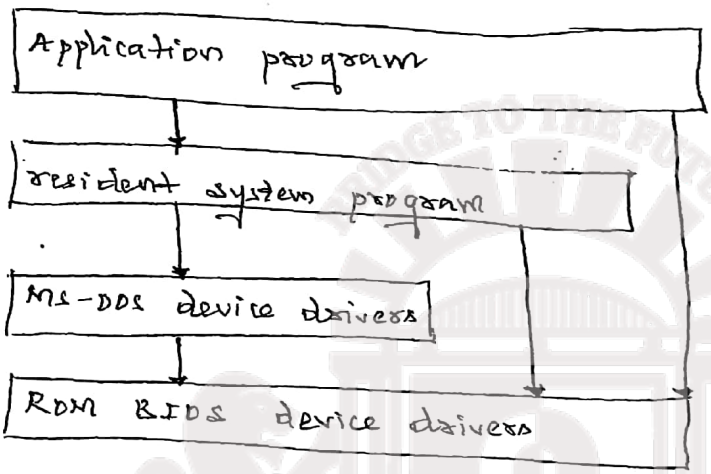


Fig: MS-DOS Layer structure.

In MS-DOS, the interfaces & levels of functionality are not well separated. The programs were able to access I/O device directly. Such freedom leaves MS-DOS vulnerable to errant (malicious) programs, causing entire system crashes when the virus prog. fails. It was limited by hardware functionality.

Another example of limited structuring is the UNIX OS. It consists of two separate parts: the kernel & the system program. The kernel is further separated into a series of interfaces & device drivers. The kernel provides all the functionalities. This monolithic structure was difficult to implement & maintain.

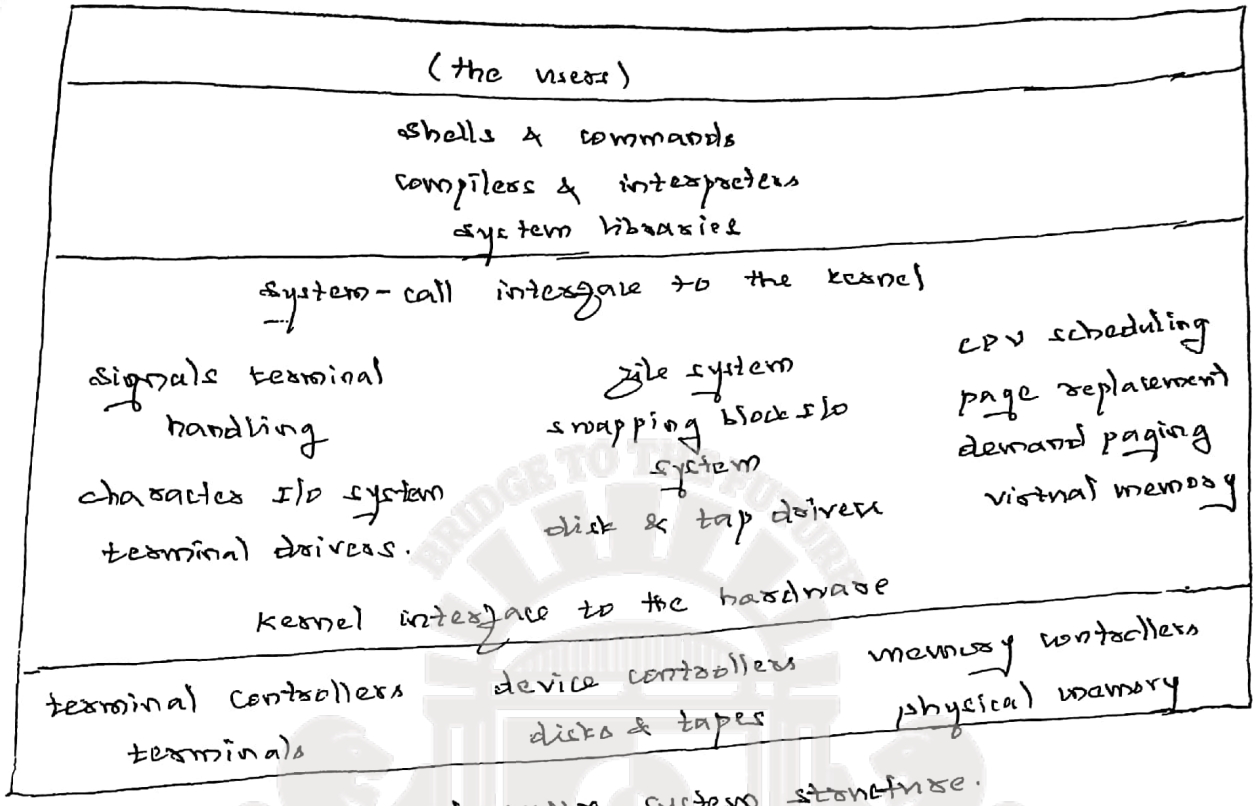


Fig: Traditional UNIX system structure.

Layered Approach:

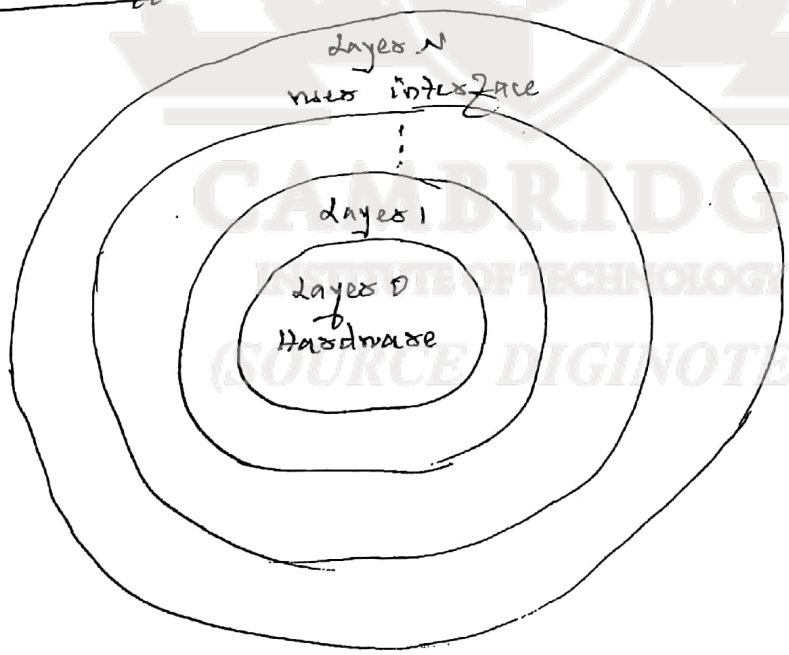


Fig: A layered operating system.

with proper hardware support, OS can be broken into pieces that are smaller & more appropriate known as layers. The top-down approach was used to separate the overall functionality & features.

The layered approach may have N no. of layers. The bottom layer (layer 0) is the hardware, the highest (layer N) is the user interface.

An OS layer is an implementation of an abstract object made up of data & operations that can manipulate those data. A typical layer consists of data structures & set of routines that can be invoked by higher level layers.

Advantages:

→ simplicity of construction
→ easier to debug
→ each layer doesn't need to know the operation of other layers.

Disadvantages:

→ appropriately defining the various layers.
→ they tend to be less efficient than other types.

Microkernels:

Earlier the kernel was large & difficult to manage. Microkernel approach is new method to overcome this. This method structures the OS by removing all nonessential components from the kernel & implementing them as system & user-level programs.

Microkernels provide minimal process & memory management. The main function of the microkernel is to provide a communication facility b/w the client program & the various service that are also running in user space. Communication is provided by message passing. For example; if the client prog. wishes to access a file, it must interact with the file server. The client prog. & service never interact directly, they communicate indirectly by exchanging messages with the kernel.

Advantage:

One benefit of the microkernel approach is ease of extending the OS. All new services are added to user space & don't require modification of the kernel.

The OS is easier to port from one hardware design to another. The microkernel also provides more security & reliability, since most services are running as user rather than kernel processes. If a service fails, the rest of the OS remains untouched.

Ex: Tandy UNIX, QNX etc follow this approach.

Disadvantage: Microkernels can suffer from performance decreases due to increased system function overhead.

Modules :

The best current methodology for OS design involves using object-oriented programming techniques to create a modular kernel.

The kernel has a set of core components & links in additional services either during boottime or runtime.

Eg : Solaris, Linux. & Mac OS X.

The Solaris OS structure shown below, is organized around a core kernel with several types of loadable kernel modules.

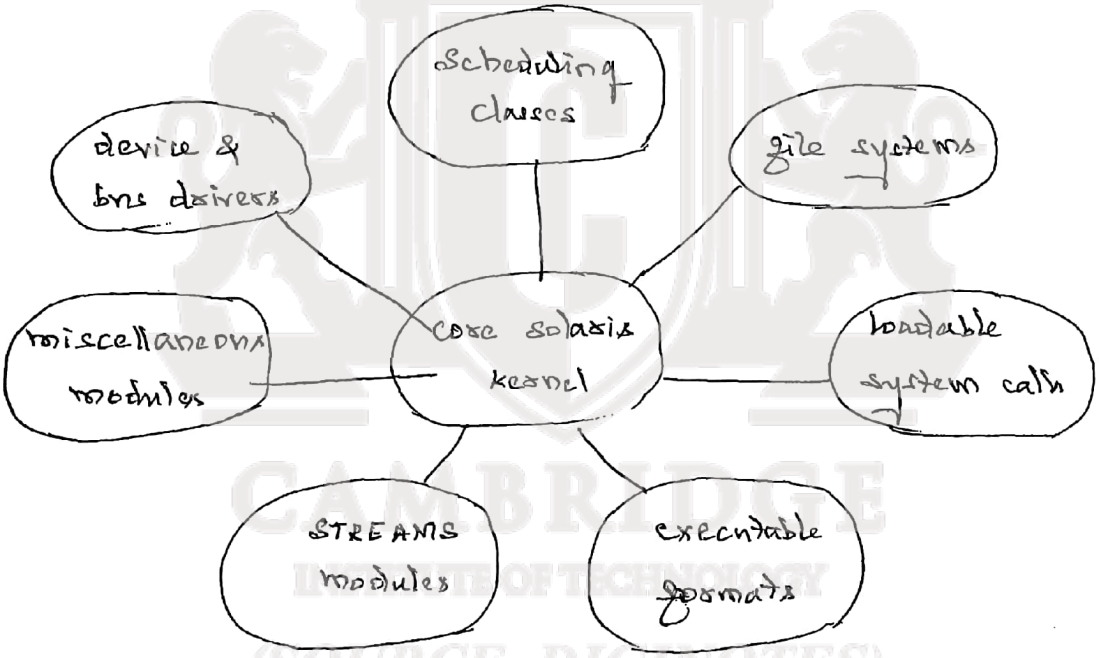


Fig: Solaris loadable modules.

This approach is similar to layered system but it is more flexible & any module can call any other module. The kernel will have only core functions making kernel very small & similar to microkernel.

Virtual Machines.

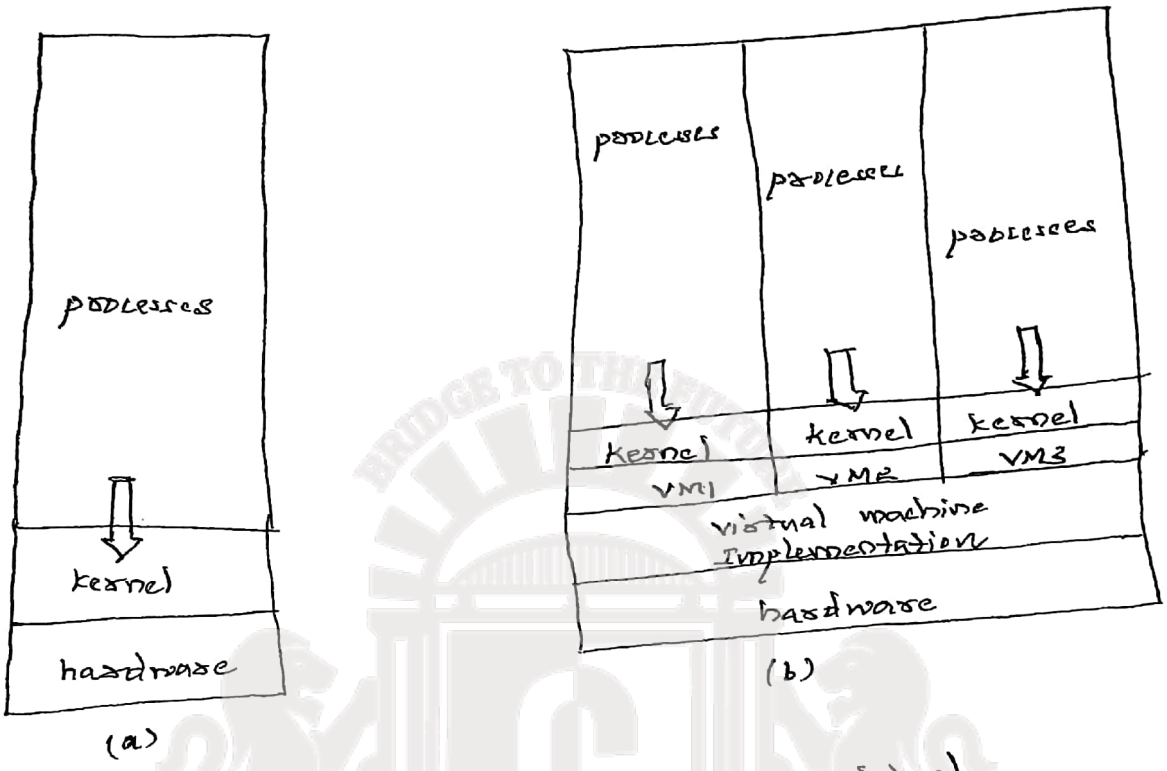


Fig: System model. (a) Non-virtual machine. (b) virtual machine.

The fundamental idea behind a virtual machine is to abstract the hardware of a single compute i.e. the CPU, memory, disk drives, network interface card etc. into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

By using CPU scheduling & virtual memory techniques, an OS host can create the illusion that a process has its own processor with its own (virtual) memory.

Benefits

One important advantage is that the host system is protected from the virtual machines. A virus inside a guest OS might damage that OS but is unlikely to affect the host or the other guests. Because each virtual machine is completely isolated from all other virtual machines, there are no protection problems.

There is no direct sharing of resources. Two approaches to provide sharing have been implemented. First, it is possible to share a file-system volume & thus to share files. Second, it is possible to define a network of virtual machines, each of which can send information over the virtual communications network.

Operating systems are large & complex programs, & it is difficult to change to one part will not cause obscure bugs to appear in some other part as it executes in kernel mode. A virtual machine system can eliminate much of this problem. System programmers are given their own virtual machine & system development is done on virtual m/c instead of on a physical m/c.

Another advantage of virtual m/c for developers is that multiple OS can be running on developers workstation concurrently. This virtualized workstation allows for rapid porting & testing of programs in varying environments. Similarly, quality assurance engineers can test their applications in multiple environments without buying, powering & maintaining a computer for each environment.

Examples: VMware:

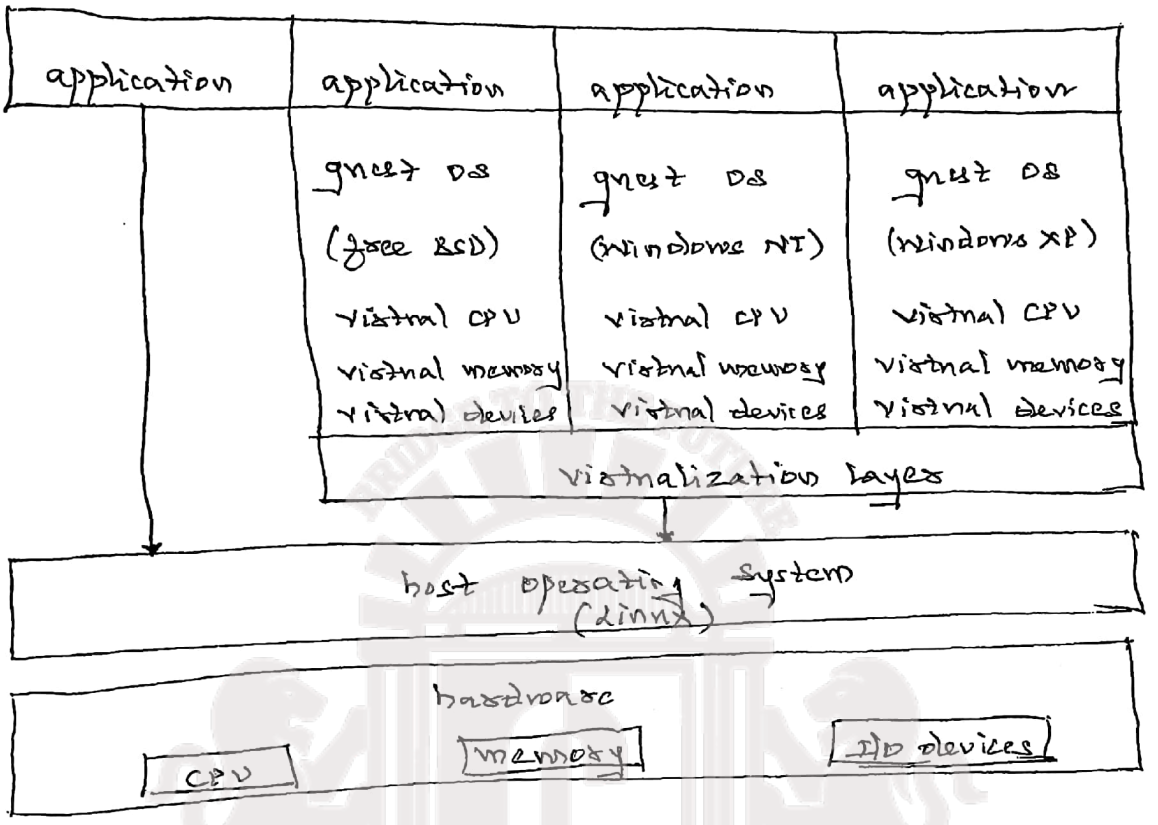


fig: VMware architecture

VMware workstation is a popular commercial application that abstracts Intel x86 and compatible hardware into isolated virtual machines. VMware workstation runs as an application on a host OS such as windows OS linux & allows this host system to concurrently and several different guest OS as independent virtual machines.

The virtualization layer is the heart of VMware, as it abstracts the physical hardware into isolated virtual machines running as guest OS. Each virtual machine has its own virtual CPU, memory, disk drives, network interfaces & so forth.

The Java virtual machine (JVM)

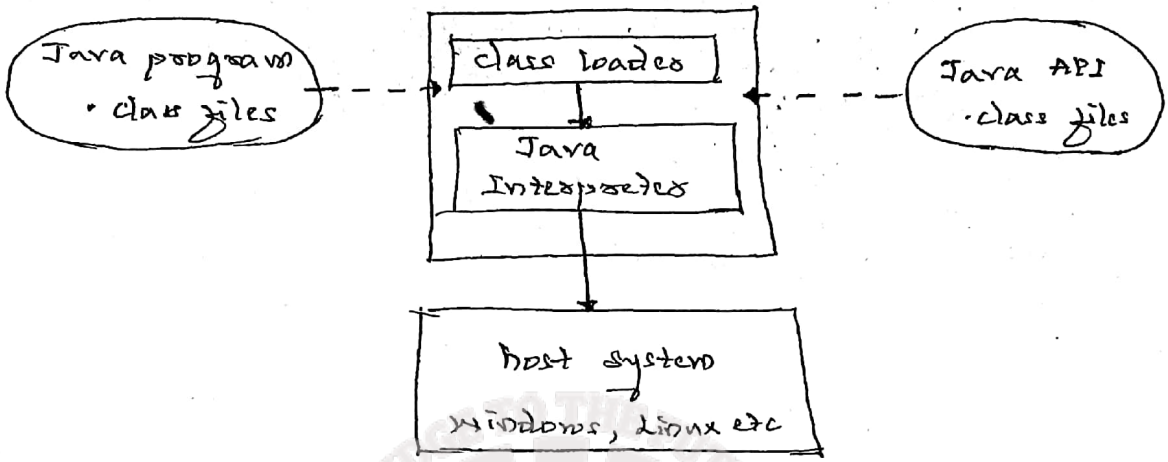


Fig: The Java virtual m/c.

Java is a popular object-oriented programming language. Java provides a specification for a java virtual m/c (JVM)

Java objects are specified with the class constructs, a java prog. consists of one or more classes. For each java class, the compiler produces an architecture neutral bytecode output (.class file) that will run on any implementation of the JVM.

The JVM is a specification for an abstract computer. It consists of a loader & java interpreter that execute bytecode. The class loader loads the compiled .class file from both java prog. & API for execution by java interpreter. After a class is loaded, the verifier checks that the .class file is valid java bytecode. If the class passes verification, it is run by java interpreter.