Operating Systems

Introduction to Operating Systems

- An *operating system* is a program that acts as an intermediary between a user of a computer and the computer hardware.

- The purpose of an operating system is to provide an environment in which a user can execute programs.

- An operating system is an important part of almost every computer system.

Operating system goals:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

- In brief, an operating system is the set of programs that controls a computer. Some examples of operating systems are UNIX, Mach, MS-DOS, MS-Windows, Windows/NT, OS/2 and MacOS.

- A computer system can be divided roughly into four components:
- the *hardware,*
- the *operating system,*
- the *application programs* and
- the *users* (Figure 1.1).

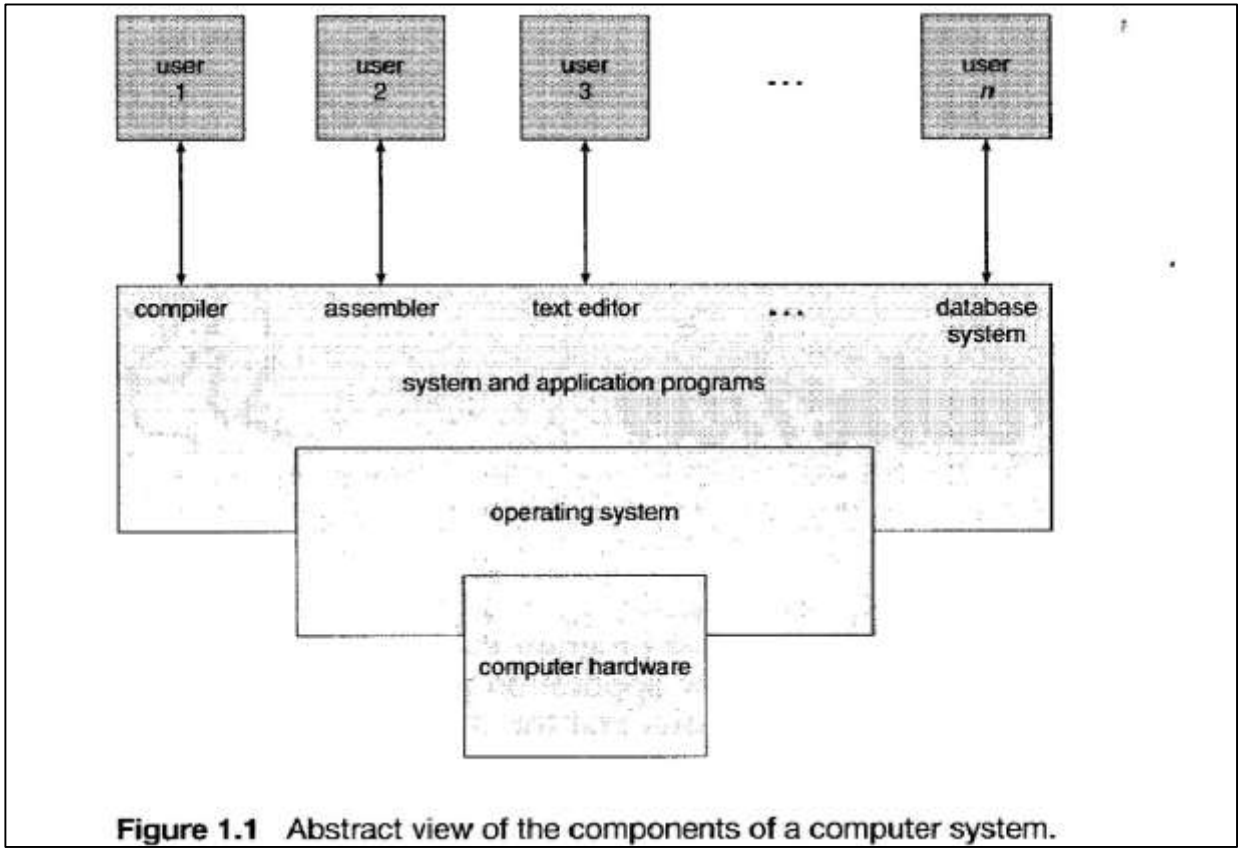


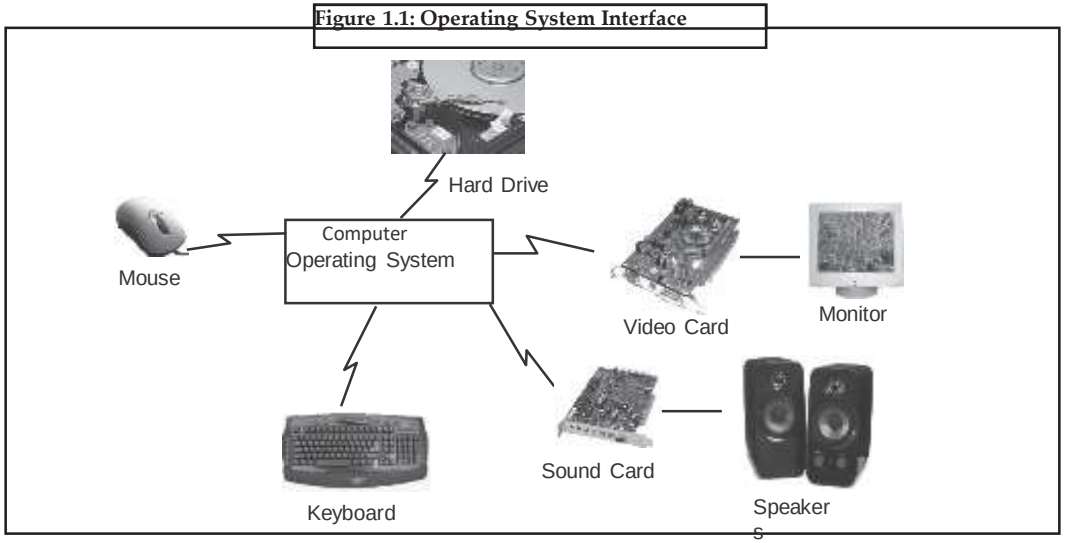
Figure 1.1 Abstract view of the components of a computer system.

- Computer system can be divided into four components
  - **Hardware** – provides basic computing resources
    - CPU, memory, I/O devices
  - **Operating system**
    - Controls and coordinates use of hardware among various applications and users
  - **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - **Users**
    - People, machines, other computers

OS Definition

- OS is a resource allocator
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer

Figure 1.1: Operating System Interface



OPERATING SYSTEM

Responsible for common tasks

Controls hardware

Manages and allocates resources

Provides an interface

History of Operating Systems

- **The 1940's - First Generations**

  The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of.

- **The 1950's - Second Generation**

  By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time.

History of Operating Systems

- **The 1960's - Third Generation**

  The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once.

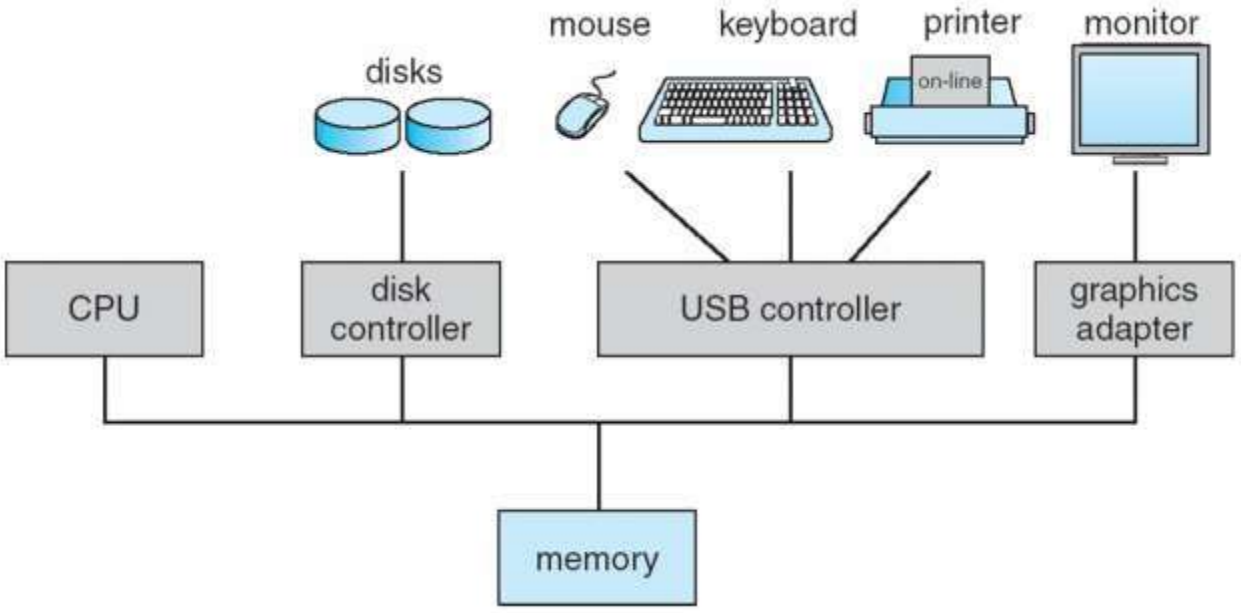
- **Fourth Generation**

  With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it becomes possible to build desktop computers as powerful as the mainframes of the 1970s.

Computer Startup

- bootstrap program is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as firmware
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

Computer System Organisation



- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller

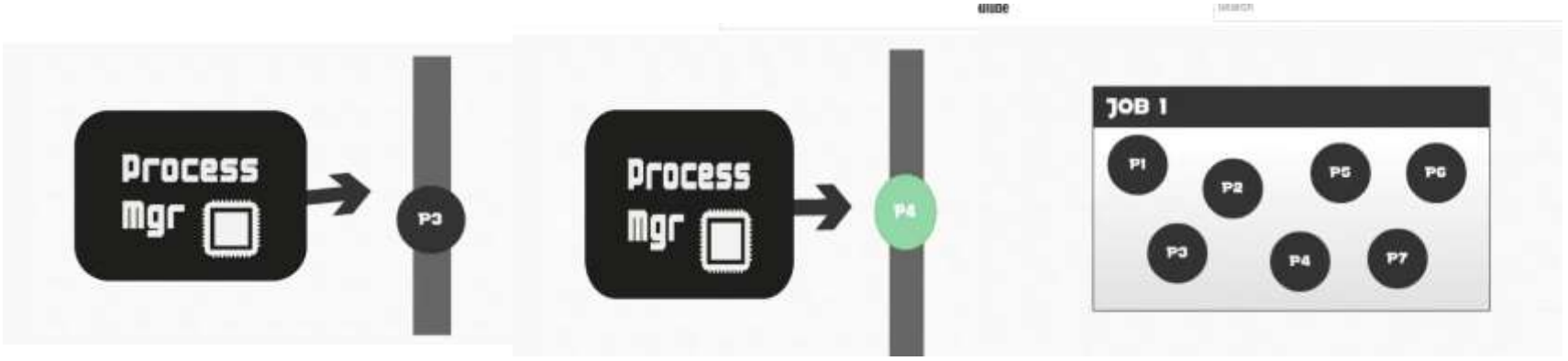
Operating System Operations

- **Process Management**

  A *process* is only ONE instant of a program in execution.There are many processes can be running the same program.

- The operating system is responsible for managing the processes i.e assigning the processor to a process at a time. This is known as **process scheduling.**

- The different algorithms used for process scheduling are FCFS (first come first served), SJF (shortest job first), priority scheduling, round robin scheduling etc.
- There are many scheduling queues that are used to handle processes in process management.
- When the processes enter the system, they are put into the job queue.



- The processes that are ready to execute in the main memory are kept in the ready queue.
- The processes that are waiting for the I/O device are kept in the device queue.

The five major activities of an operating system in regard to process management are:

- *Creation and deletion of user and system processes.*
- *Suspension and resumption of processes.*
- *A mechanism for process synchronization.*
- *A mechanism for process communication.*
- *A mechanism for deadlock handling.*

- **Main-Memory Management**

➢ Memory management plays an important part in operating system.

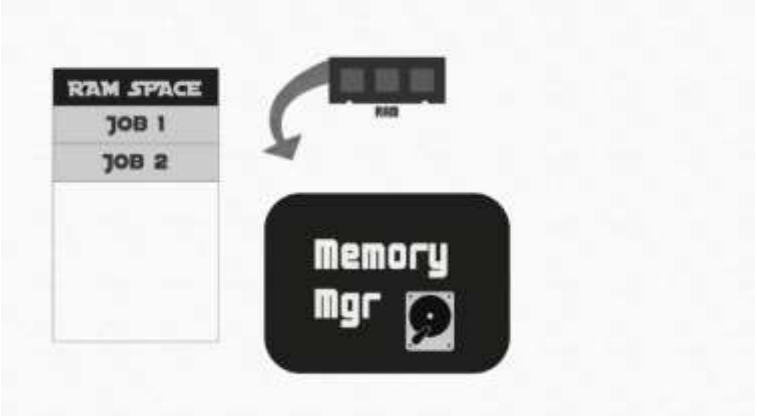
➢ It deals with memory and the moving of processes from disk to primary memory for execution and back again.

The major activities of an operating system in regard to memory-management are:

- *Keep track of which part of memory are currently being used/empty and by whom.*
- *Allocate and deallocate memory space as needed.*

▪ The operating system assigns memory to the processes as required. This can be done using best fit, first fit and worst fit algorithms.

- All the memory is tracked by the operating system i.e. it nodes what memory parts are in use by the processes.
- The operating system deallocated memory from processes as required. This may happen when a process has been terminated or if it no longer needs the memory.

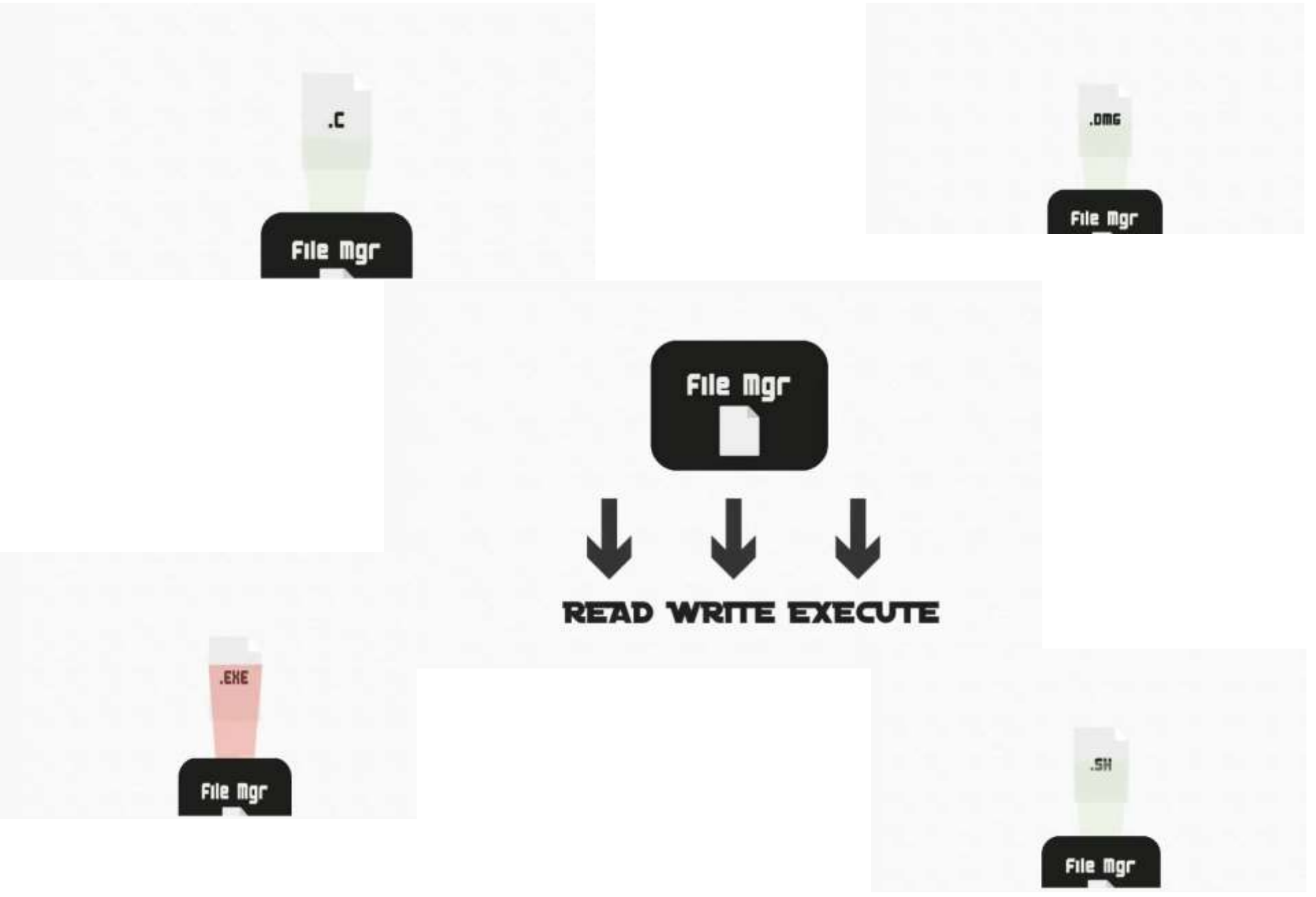


- **File Management**

  A file is a collection of related information defined by its creator. Computer can store files on the disk (secondary storage), which provide long term storage.

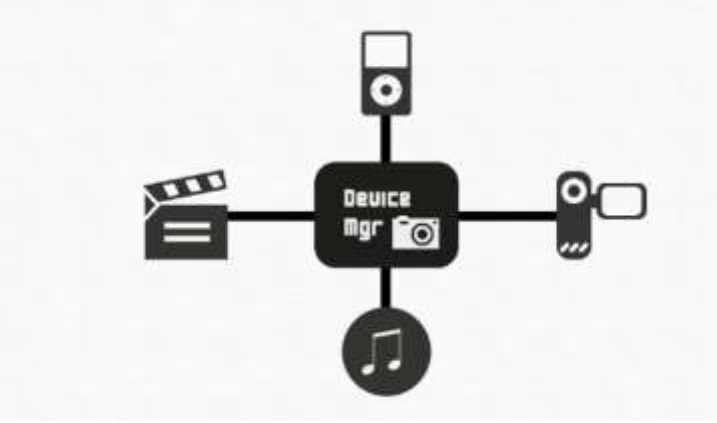
  The major activities of an operating system in regard to memory-management are:

- *The creation and deletion of files.*
- *The creation and deletion of directories.*
- *The support for manipulating files and directories.*
- *The mapping of files onto secondary storage.*
- *The backup of files on stable storage media.*


.C
File Mgr
.DMG
File Mgr
File Mgr
READ WRITE EXECUTE
.EXE
File Mgr
.SH
File Mgr

- **I/O System Management**

  One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.



Operating Systems Services

- **Program Execution**

  The system must be able to load a program into memory and to run it. The program must be able to end its execution, either normally or abnormally (indicating error).

- **I/O Operations**

  A running program may require I/O. This I/O may involve a file or an I/O device.

- **File System Manipulation**

  The output of a program may need to be written into new files or input taken from some files. The operating system provides this service.

- **Error Detection**

  An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation the operating system constantly monitors the system for detecting the errors.

- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network .Communications may be via shared memory or through message passing (packets moved by the OS)

Error detection – OS needs to be constantly aware of possible errors may occur in the CPU and memory hardware, in I/O devices, in user program

For each type of error, OS should take the appropriate action to ensure correct and consistent computing

Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Resource allocation - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them

Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.

Accounting - To keep track of which users use how much and what kinds of computer resources.

- **Networking**

  A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock. The processors communicate with one another through communication lines called network.

- **Protection System**

  Protection refers to mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.

- **Command Interpreter System**

  A command interpreter is an interface of the operating system with the user. The user gives commands which are executed by operating system (usually by turning them into system calls).

System Calls in Operating System

What is a System Call?

- A system call is **a method for a computer program** to **request a service from the kernel** of the operating system on which it is running.
- A system call is a method of interacting with the operating system via programs.
- The **Application Program Interface (API)** connects the operating system's functions to user programs.
- It acts as a link between the operating system and a process, allowing user-level programs to request operating system services.
- The kernel system can only be accessed using system calls. System calls are required for any programs that use resources.

How are system calls made?

- When a computer software needs to access the operating system's kernel, it makes a system call.
- The system call uses an API to expose the operating

system's services to user programs.

- It is the only method to access the kernel system.
- All programs or processes that require resources for execution must use system calls, as they serve as an interface between the operating system and user programs.

Why do you need system calls in Operating System?

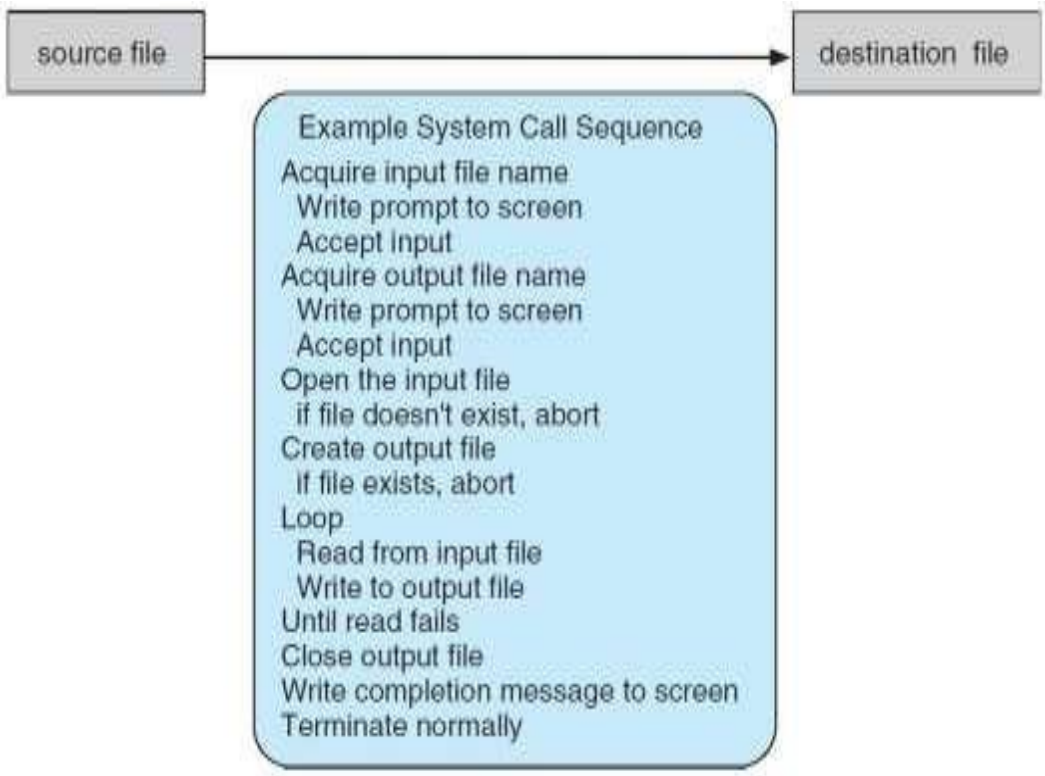
1. It is must require when a file system wants to create or delete a file.
2. Network connections require the system calls to sending and receiving data packets.
3. If you want to read or write a file, you need to system calls.
4. If you want to access hardware devices, including a printer, scanner, you need a system call.
5. System calls are used to create and manage new processes.

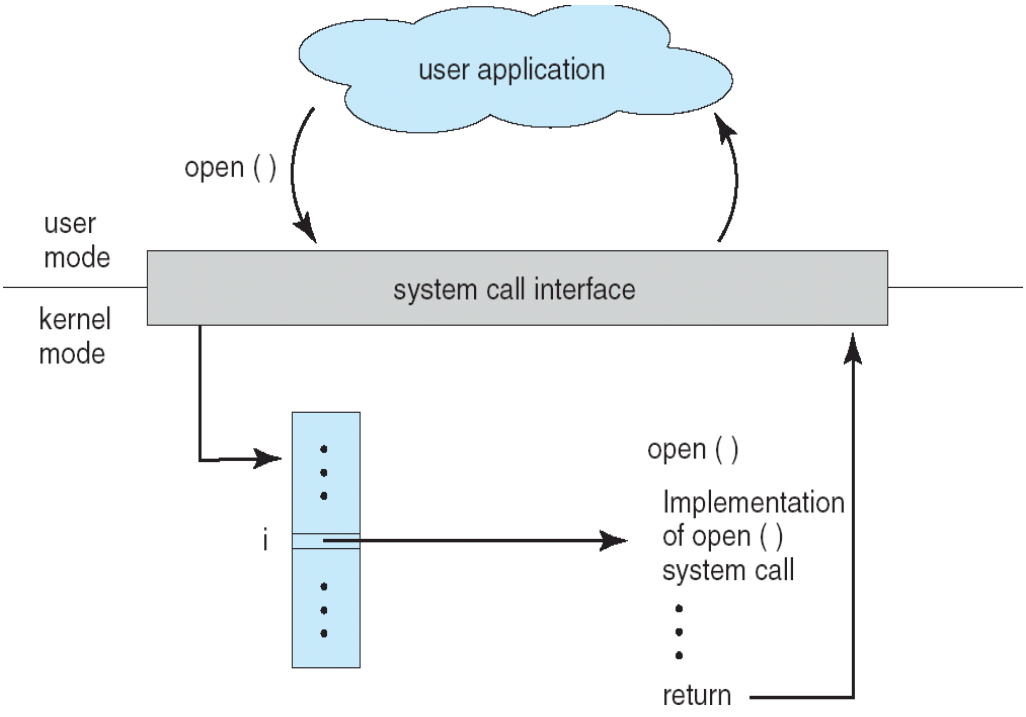
How System Calls Work?

- The Applications run in an area of memory known as **user space.**

- A system call connects to the operating system's kernel, which executes in **kernel space.**
- When an application creates a system call, it must first obtain permission from the kernel. It achieves this using an **interrupt request**, which pauses the current process and transfers control to the kernel.
- If the **request is permitted**, the kernel performs the requested action, like creating or deleting a file.
- As input, the application **receives the kernel's output**. The application resumes the procedure after the input is received.
- When the operation is finished, the kernel returns the results to the application and then moves data from kernel space to user space in memory.
- A **simple system call** may take **few nanoseconds** to provide the result, like retrieving the system date and time.
- A more **complicated system call**, such as connecting to a network device, **may take a few seconds.**
- Modern operating systems are multi-threaded, which means they can handle various system calls at the same time.

Example: System call sequence to copy the contents of one file to another file





Types of System Calls

There are commonly five types of system calls. These are as follows:

1. **Process Control**
2. **File Management**
3. **Device Management**
4. **Information Maintenance**
5. **Communication**

1.Process Control

Process control is the system call that is used to direct the processes. Some process control examples include creating, load, abort, end, execute, process, terminate the process, etc.

2.File Management

File management is a system call that is used to handle the files. Some file management examples include creating files, delete files, open, close, read, write, etc.

3.Device Management

Device management is a system call that is used to deal with devices. Some examples of device management include read, device, write, get device attributes, release device, etc.

4. Information Maintenance

Information maintenance is a system call that is used to maintain information. There are some examples of information maintenance, including getting system data, set time or date, get time or date, set system data, etc.

5.Communication

Communication is a system call that is used for communication. There are some examples of communication, including create, delete communication connections, send, receive messages, etc.

Process	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	Fork() Exit() Wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	Open() Read() Write() Close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	Ioctl() Read() Write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	Getpid() Alarm() Sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	Pipe() Shmget() Mmap()

Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorgroup()	Chmod() Umask() Chown()

Examples of Windows and Unix system calls

open()

The **open()** system call allows you to access a file on a file system. It allocates resources to the file and provides a handle that the process may refer to. Many processes can open a file at once or by a single process only. It's all based on the file system and structure.

read()

It is used to obtain data from a file on the file system. It accepts three arguments in general:

- A file descriptor.
- A buffer to store read data.
- The number of bytes to read from the file.

The file descriptor of the file to be read could be used to identify it and open it using **open()** before reading.

wait()

In some systems, a process may have to wait for another process to complete its execution before proceeding. When a parent process makes a child process, the parent process execution is suspended until the child process is finished. The **wait()** system call is used to suspend the parent process. Once the child process has completed its execution, control is returned to the parent process.

write()

It is used to write data from a user buffer to a device like a file. This system call is one way for a program to generate data. It takes three arguments in general:

- A file descriptor.
- A pointer to the buffer in which data is saved.
- The number of bytes to be written from the buffer.

fork()

Processes generate clones of themselves using the **fork()** system call. It is one of the most common ways to create processes in operating systems. When a parent process spawns a child process, execution of the parent process is

interrupted until the child process completes. Once the child process has completed its execution, control is returned to the parent process.

close()

It is used to end file system access. When this system call is invoked, it signifies that the program no longer requires the file, and the buffers are flushed, the file information is altered, and the file resources are de-allocated as a result.

exec()

When an executable file replaces an earlier executable file in an already executing process, this system function is invoked. As a new process is not built, the old process identification stays, but the new process replaces data, stack, data, head, etc

exit()

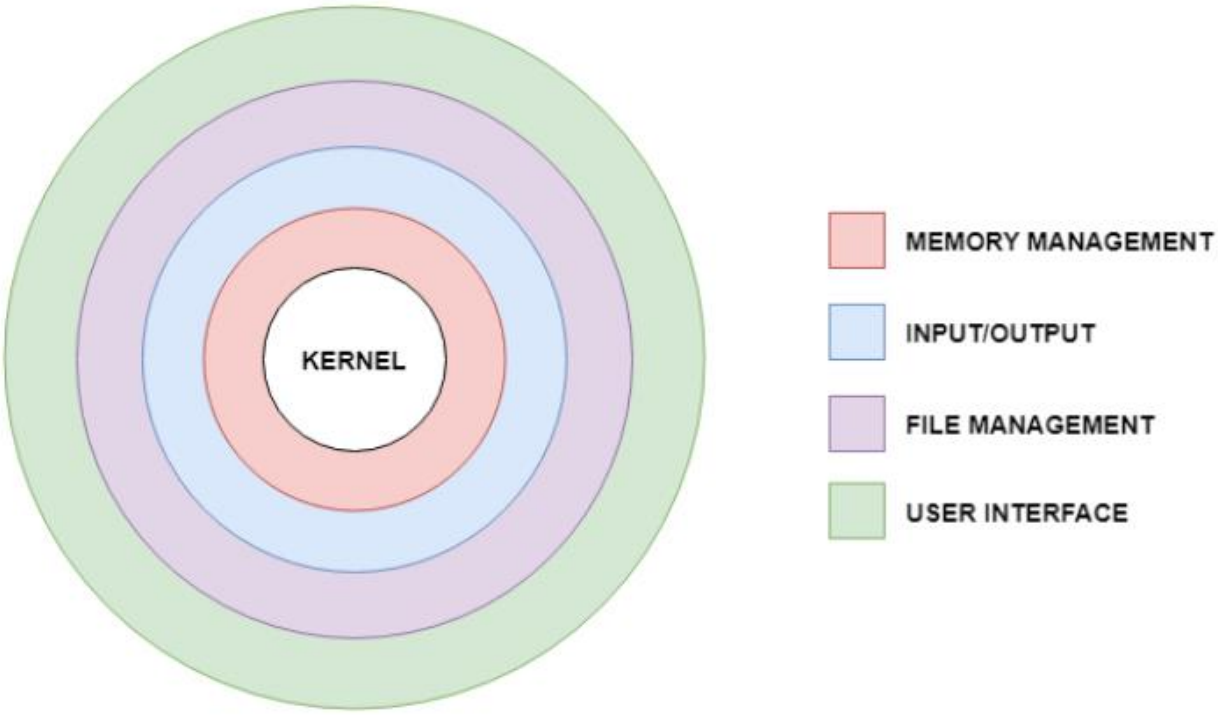
The **exit()** is a system call that is used to end program execution. This call indicates that the thread execution is complete, which is especially useful in multi-threaded environments. The operating system reclaims resources spent by the process following the use of the **exit()** system function.

OS DESIGN AND IMPLEMENTATION

An operating system is a construct that allows the user application programs to interact with the system hardware.

Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.

There are many problems that can occur while designing and implementing an operating system. These are covered in operating system design and implementation.



Layered Operating System Design

Operating System Design Goals

It is quite complicated to define all the goals and specifications of the operating system while designing it.

The design changes depending on the

- type of the operating system i.e if it is batch system, time shared system, single user system, multi user system, distributed system etc.
- Choice of hardware.

There are basically **two types of goals** while designing an operating system.

User Goals(user requirement)

The operating system should be convenient, easy to use, reliable, safe and fast according to the users. However, these specifications are not very useful as there is no set method to achieve these goals.

System Goals(designer/engineer requirement)

The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system. But there is not specific method to achieve these goals as well.

Operating System Mechanisms and Policies

There is no specific way to design an operating system as it is a highly creative task. However, there are general software principles that are applicable to all operating systems.

A subtle difference between mechanism and policy is that **mechanism** shows how to do something and **policy** shows what to do.

Policies may change over time and this would lead to changes in mechanism. So, it is better to have a **general mechanism** that would require few changes even when a policy change occurs.

For example - If the mechanism and policy are independent, then few changes are required in mechanism if policy changes. If a policy favours I/O intensive processes over CPU intensive processes, then a policy change to preference of CPU intensive processes will not change the mechanism.

Operating System Implementation

The operating system needs to be implemented after it is designed. Earlier they were written in assembly language but now higher level languages are used. The first system not

written in assembly language was the Master Control Program (MCP) for Burroughs Computers.

Advantages of Higher Level Language

There are multiple advantages to implementing an operating system using a higher level language such as:

- The code can be written faster
- It is more compact
- It is easier to understand and debug.
- It is easier to port(the operating system can be easily moved from one hardware to another)

Disadvantages of Higher Level Language

- It leads to a loss in speed and increase in storage requirements.However in modern systems only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager.
- Sometimes the routines in the system can be replaced by assembly language equivalents if required.

OPERATING SYSTEM STRUCTURES

Operating system can be implemented with the help of various structures. The structure of the OS depends mainly on **how the various common components of the operating system are interconnected and melded into the kernel**. **structures of the operating system:**

1.Simple structure

2.Layered structure

3.Microkernel Structure

4.Modular structure

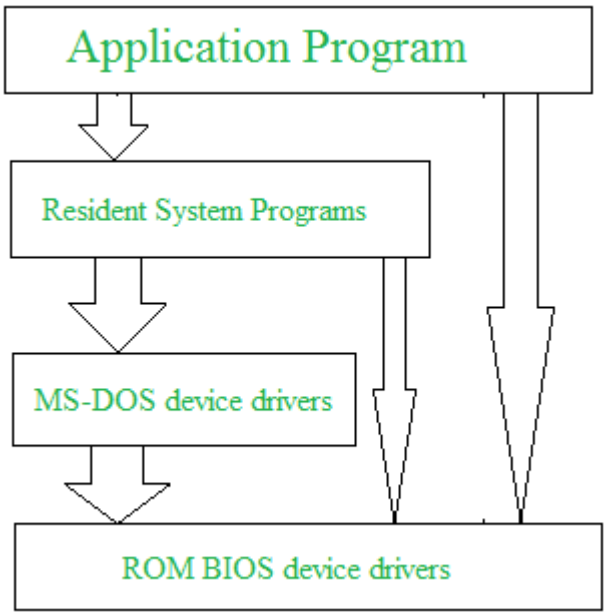
1.Simple structure:

(i)Such operating systems do not have well defined structure and are small, simple and limited systems.

(ii)The interfaces and levels of functionality are not well separated.

Example : MS-DOS

In MS-DOS, application programs are able to access the basic I/O routines. These types of operating system cause the entire system to crash if one of the user programs fails.



Structure of MS-DOS

Advantages of Simple structure:

- It delivers better application performance because of the few interfaces between the application program and the hardware.
- Easy for kernel developers to develop such an operating system.

Disadvantages of Simple structure:

- The structure is very complicated as no clear boundaries exists between modules.
- It does not enforce data hiding in the operating system.

2. Layered structure:

(i)An OS can be broken into pieces and retain much more control on system.

(ii)In this structure the OS is broken into **number of layers** (levels). The bottom layer **(layer 0) is the hardware** and the topmost **layer (layer N) is the user interface.**

Layer1 –CPU scheduling

Layer2- memory management

Layer3- operator process communication

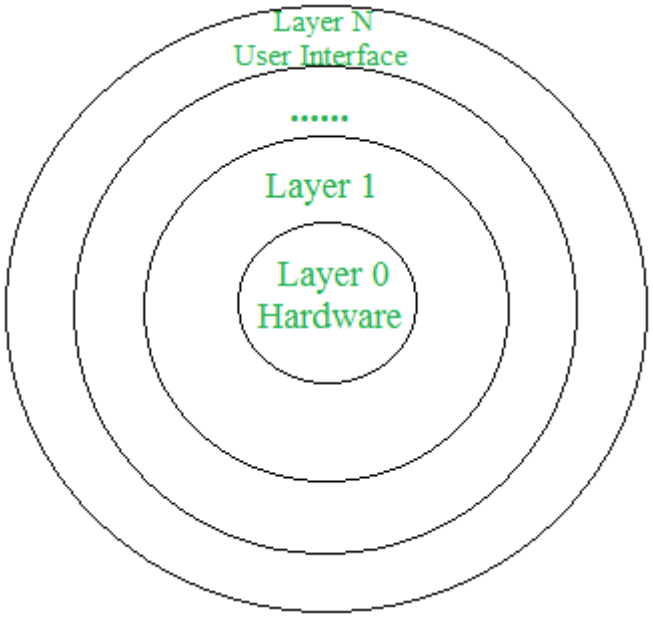
Layer4-Input Output management

(iii) each upper layer is built on the bottom layer. All the layers hide some structures, operations etc. from their upper layers.

(iv)One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.

(v) One way to achieve modularity in the operating system is the **layered approach**.

Example: UNIX



Advantages of Layered structure:

- Layering makes it easier to enhance the operating system as implementation of a layer can be changed easily without affecting the other layers.
- It is very easy to perform debugging and system verification.

Disadvantages of Layered structure:

- In this structure the application performance is degraded as compared to simple structure.
- It requires careful planning for designing the layers as higher layers use the functionalities of only the lower layers.

3. Micro-kernel Structure:

(i)This structure designs the operating system by **removing all non-essential components from the kernel** and implementing them as system and user programs.

(ii)This result in a smaller kernel called the **micro-kernel**.

(iii)All new services need to be added to user space and does not require the kernel to be modified. Thus it is **more secure** and **reliable** as if a service fails then rest of the operating system remains untouched.

Example : Mac OS

Advantages of Micro-kernel structure:

- It makes the operating system portable to various platforms.
- As microkernels are small so these can be tested effectively.

Disadvantages of Micro-kernel structure:

- Increased level of inter module communication degrades system performance.

4. Modular structure or approach:

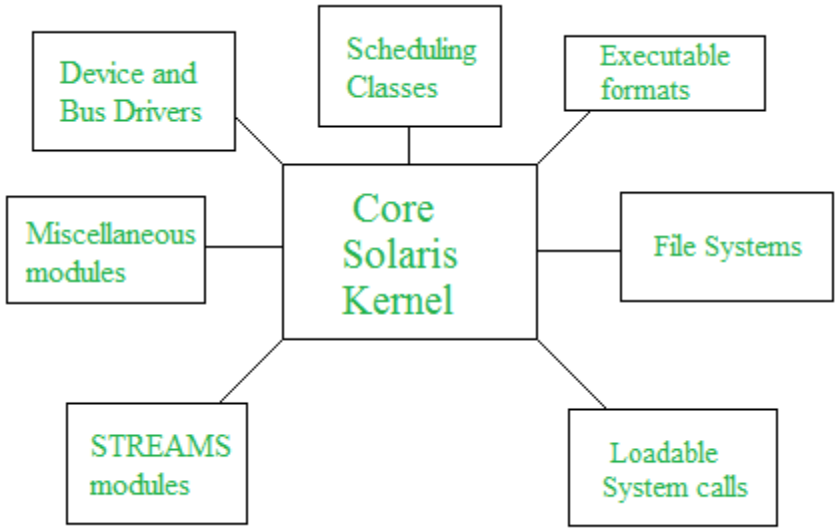
(i) It is considered as the best approach for an OS.

(ii) It involves designing of a modular kernel.

(iii)The kernel has only set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time.

(iv) It **resembles layered structure** due to the fact that each kernel has defined and protected interfaces but it is more flexible than the layered structure as a module can call any other module.

Example : **Solaris OS**



Virtual Machine (VM)

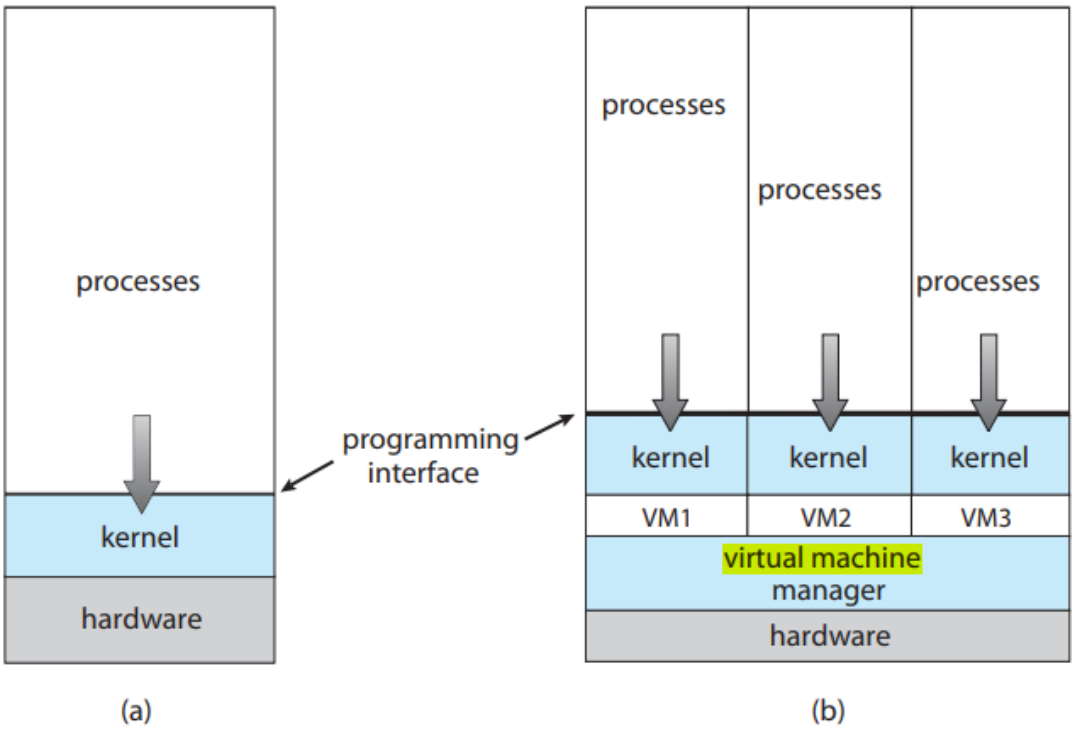
- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer into several execution environment thereby **creating the illusion** that each separate execution is running its own private computer.
- In a dual boot , we can run one OS at a time. By using Virtual machine we can run multiple OS at a time.
- A virtual machine is an execution of a computer system within our physical personal computer.
- It allows users to install and run multiple OS in a single PC thereby making illusion that each separate execution is running its own private computer.
- Here resources of computer system are shared to each OS running by virtual machine.

Characteristics of virtual machines

The characteristics of the virtual machines are as follows −

- Multiple OS systems use the same hardware and partition resources between virtual computers.
- Separate Security and configuration identity.
- Ability to move the virtual computers between the physical host computers as holistically integrated files.

The below diagram shows you the difference between the single OS with no VM and Multiple OS with VM –



(a) Nonvirtual machine. (b) Virtual machine.

VIRTUALIZATION

- A virtual machine is a type of computer application used to **create a virtual environment**, which is referred to as **virtualization**.
- Virtualization may also be used to run multiple operating systems at the same time. Through the help of a virtual machine, the user can operate software located on the computer platform.
- No separate hardware needed .It takes hardware resources from Host OS.
- It creates virtual CPU,virtual RAM,virtual storage for

each virtual machine.

- Hardware resources are shared.
- Virtual machines are completely isolated.If something fails inside virtual machine ,it doesn't affect the host machine.

IMPLEMENTATION:

- virtual machine sofware runs in kernel mode.
- virtual machine itself runs in user mode.
- Just as physical machine has 2 mode so virtual machine must have 2 modes
  - Virtual user mode
  - Virtual kernel mode.

Both of this will run in a physical user mode.

HYPERVISOR OR VIRTUAL MACHINE MONITOR:

- Hardware virtual machine software, also known as a hypervisor or virtual machine monitor.
- A computer software or firmware or hardware that creates and runs virtual machine.

Different types hypervisior.

- Type 0 Hypervisor or virtual machine monitor.
- Type 1 Hypervisor or virtual machine monitor
- Type 2 Hypervisor or virtual machine monitor

Type 0 Hypervisor or virtual machine monitor:

• Hardware-based solutions that provide support for virtual machine creation and management via firmware. These VMMs, which are commonly found in mainframe and large to midsized servers, are generally known as type 0 hypervisors.

Example: IBM LPARs and Oracle LDOMs

Type 1 Hypervisor or virtual machine monitor

A hypervisor that runs directly on the host's hardware or server to control the hardware and to manage the guest OS. It is called Bare metal hypervisor.

Example:VMwareESX/ESXi, Oracle VM server for x86,Microsoft hyperv , Citrix Xenserver.

Type 2 Hypervisor or virtual machine monitor

- A hypervisor that runs on a conventional OS just as other computer programs do.It is called Host OS hypervisor.
- General-purpose operating systems that provide standard functions as well as VMM functions, including Microsoft Windows Server with HyperV and RedHat Linux with the KVM feature. Because such systems have a feature set similar to type 1 hypervisors, they are also known as type 1.
- Applications that run on standard operating systems

but provide VMM features to guest operating systems.

Example: VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox.

ADVANTAGES:

- Easy maintenance ,availability and convenient recovery.
- Provides higher degree of protection to various resources of the system.
- Other users of the virtual machine can continue their routine opeartions without being disturbed.
- Various OS can be run concurrently to serve the different needs of various users.
- Multiple OS environment can exists simultaneously on the same machine ,isolated from each other.

DISADVANTAGES:

- The main drawback with the virtual-machine approach involves disk systems. Let us suppose that the physical machine has only three disk drives but wants to support seven virtual machines. Obviously, it cannot allocate a disk drive to each virtual machine, because virtual-machine software itself will need substantial disk space to provide virtual memory and spooling.
- When multiple virtual machines are simultaneously

running on a host computer, one virtual machine can be affected by other running virtual machines, depending on the workload.

- Virtual machines are not as efficient as a real one when accessing the hardware.