Unit I : **8 hours**

Introduction to System Software , Overview of all system software's :

Operating system , I/O manager, Assembler , Compiler, Linker , Loader.

Introductory Concepts : Operating system functions and characteristics.

Historical evolution of operating systems.

Real time systems.

Distributed systems.

An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system.

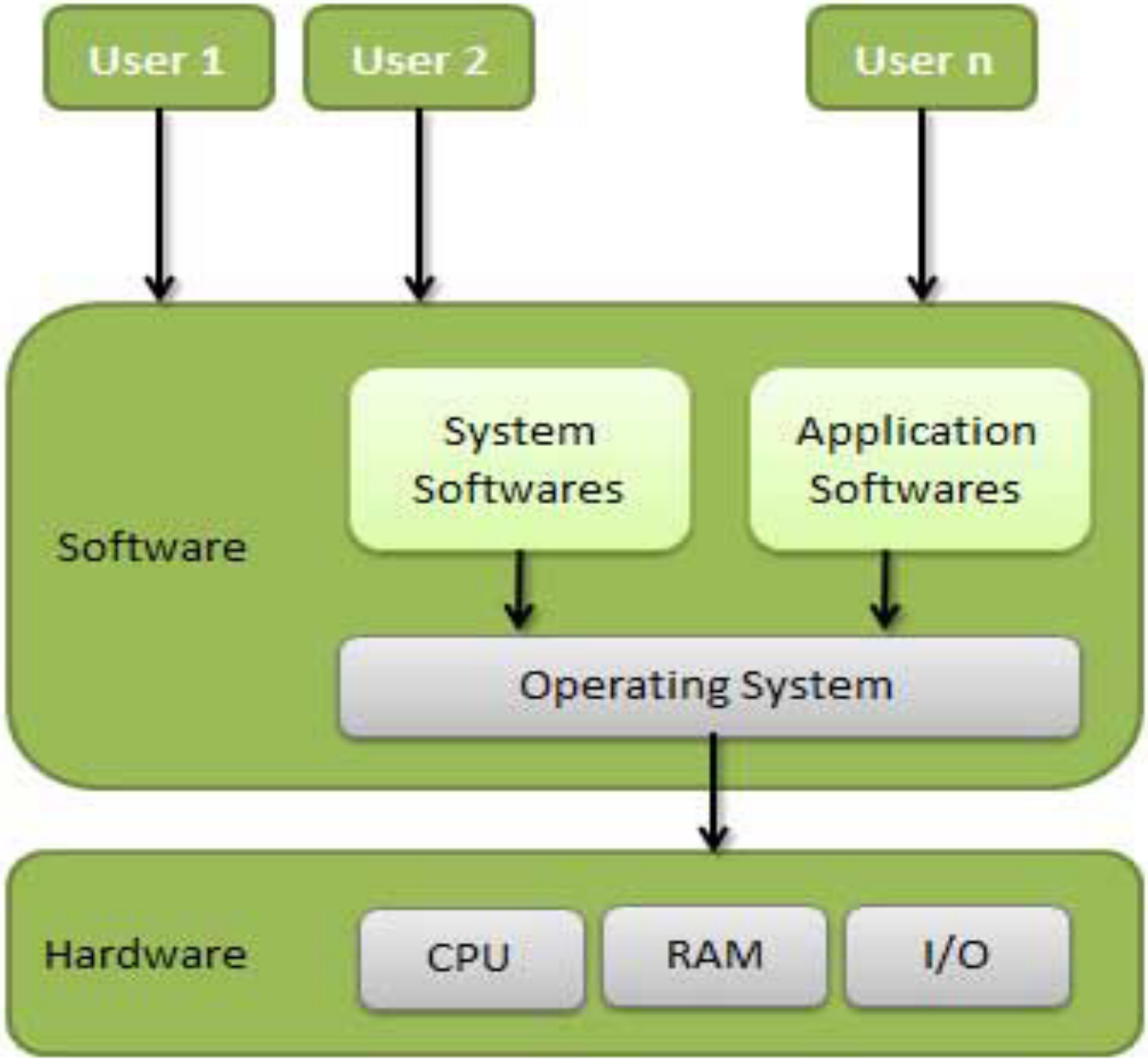
An operating System (OS) is an intermediary between users and computer hardware. It provides users an environment in which a user can execute programs conveniently and efficiently.

In technical terms, It is a software which manages hardware. An operating System controls the allocation of resources and services such as memory, processors, devices and information.

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

Following are some of important functions of an operating System.

a) Memory Management
b) Processor Management
c) Device Management
d) File Management
e) Security
f) Control over system performance
g) Job accounting
h) Error detecting aids
i) Coordination between other software and users


User 1
User 2
User n
Software
System Softwares
Application Softwares
Operating System
Hardware
CPU
RAM
I/O

Computer software can be divided into two main categories:

Application Software & System Software.

Application software consists of the programs for performing tasks particular to the machine's utilization. This software is designed to solve a particular problem for users.

e.g. spreadsheets, database systems, desktop publishing systems, program development software, and games etc.

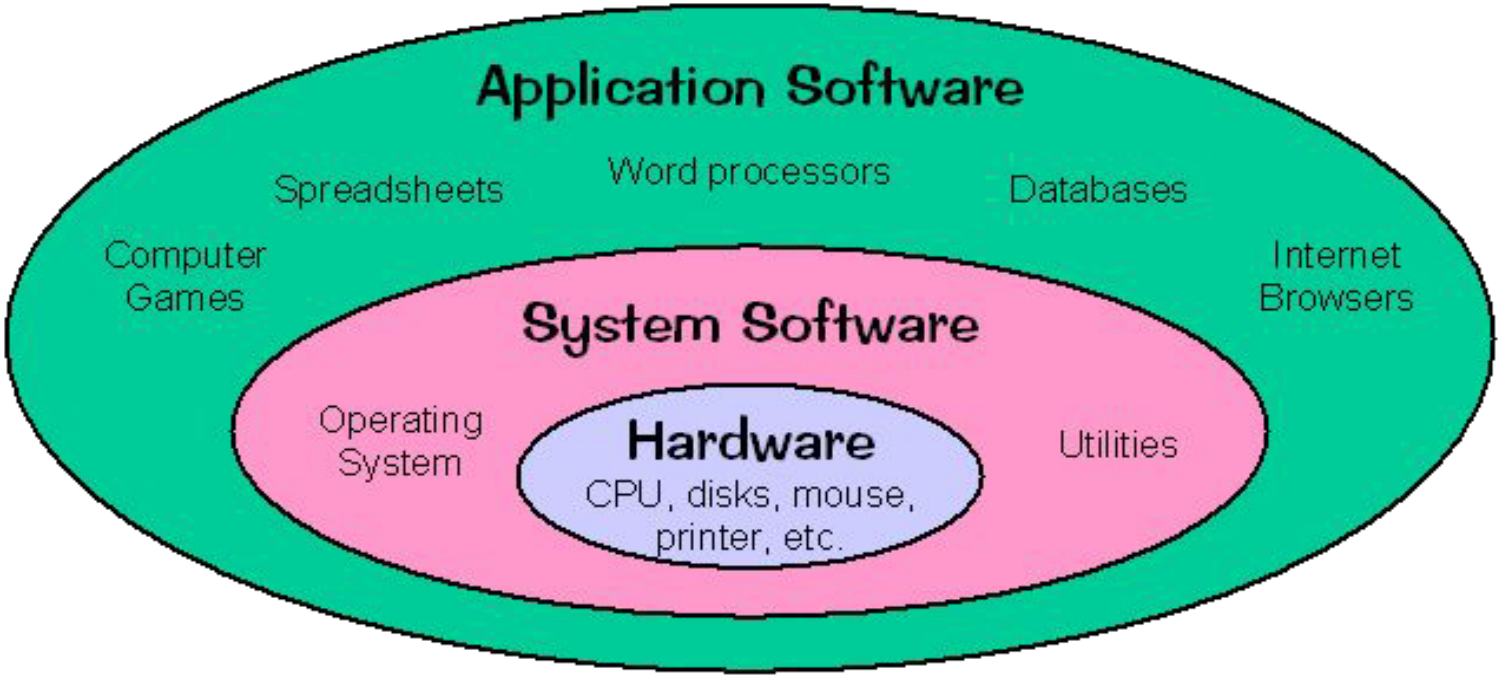
System Software is more transparent and less noticed by the typical computer user. This software provides a general programming environment in which programmers can create specific applications to suit their needs.

This environment provides new functions that are not available at the hardware level and performs tasks related to executing the application program.

System software acts as an interface between the hardware of the computer and the application software that users need to run on the computer.

The diagram below illustrates the relationship between application software and system software.

The most important type of system s/w is the OS



An operating system has three main responsibilities :

1) Perform basic tasks, such as recognizing i/p from the k/b, sending o/p to the display screen,
2) Keeping track of files and directories on the disk, and
3) Controlling peripheral devices such as disk drives and printers.

Batch		This strategy involves reading a series of jobs (called a batch) into the machine and then executing the programs for each job in the batch. This approach does not allow users to interact with programs while they operate.
Timesharing		This strategy supports multiple interactive users. Rather than preparing a job for execution ahead of time, users establish an interactive session with the computer and then provide commands, programs and data as they are needed during the session.
Personal Computing		This strategy supports a single user running multiple programs on a dedicated machine. Since only one person is using the machine, more attention is given to establishing predictable response times from the system. This strategy is quite common today because of the popularity of personal computers.
Dedicated		This strategy supports real-time and process control systems. These are the types of systems which control satellites, robots, and air-traffic control. The dedicated strategy must guarantee certain response times for particular computing tasks or the application is useless

A computer consists of various devices that provide input and output (I/O) to and from the outside world. Typical devices are keyboards, mice, audio controllers, video controllers, disk drives, networking ports, and so on. Device drivers provide the software connection between the devices and the operating system. For this reason, I/O is very important to the device driver writer.

I/O Manager

The Windows kernel-mode I/O manager manages the communication between applications and the interfaces provided by device drivers. Because devices operate at speeds that may not match the operating system, the communication between the operating system and device drivers is primarily done through **I/O Request Packets (IRPs).** These packets are similar to network packets or Windows message packets. They are passed from operating system to specific drivers and from one driver to another.

The Windows I/O system provides a layered driver model called stacks. Typically IRPs go from one driver to another in the same stack to facilitate communication.

e.g. a joystick driver would need to communicate to a USB hub, which in turn would need to communicate to a USB host controller, which would then need to communicate through a PCI bus to the rest of the computer hardware. The stack consists of joystick driver, USB hub, USB host controller, and the PCI bus.

This communication is coordinated by having each **driver in the stack** send and receive IRPs.

The I/O manager has two subcomponents: the Plug and Play Manager and Power Manager. They manage the I/O functionality for the technologies of Plug and Play and power management.

Plug and Play (PnP) is a combination of hardware technology and software techniques that enables a PC to recognize when a device is added to the system. With PnP, the system configuration can change with little or no input from the user. For example, when a USB thumb drive is plugged in, Windows can detect the thumb drive and add it to the file system automatically.

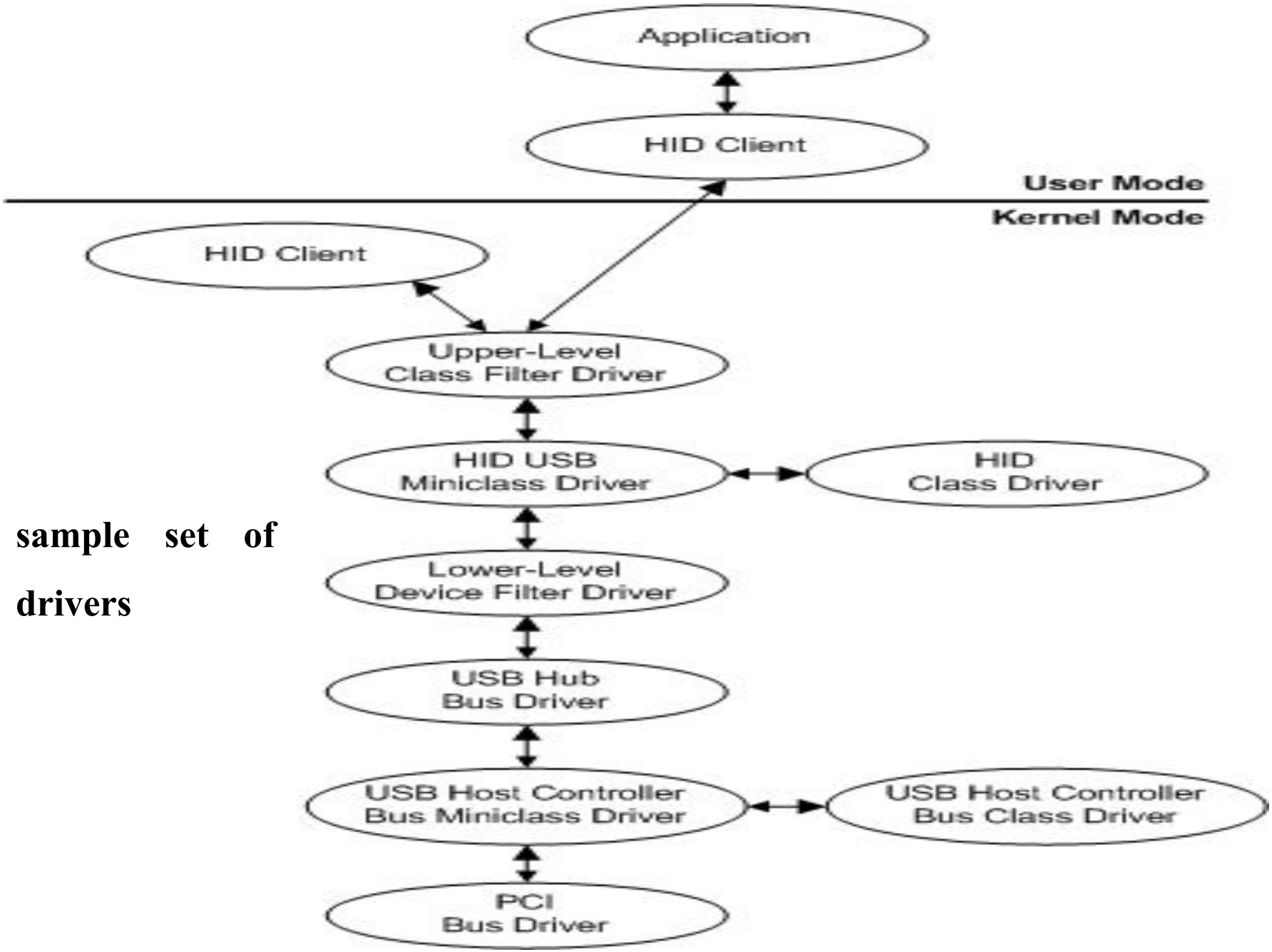
The PnP manager is actually a subsystem of the I/O manager.

A mini- driver uses an OS supplied library to abstract from the OS requirements.

Miniclass works in place of a device class driver

Class mini-driver implements device-specific details of storage class, such as tape and disk. It links against classpnp.sys. classpnp.sys implements common PNP and power support functionality.

sample set of drivers



Starting at the bottom of the previous figure, the drivers in the sample stack include:

A **PCI driver** that drives the PCI bus. This is a PnP bus driver. The PCI bus driver is provided with the system by Microsoft.

The **bus driver for the USB host controller** is implemented as a class/miniclass driver pair. The USB host controller class and miniclass drivers are provided with the system by Microsoft.

The **USB hub bus driver** that drives the USB hub. The USB hub driver is provided with the system by Microsoft.

Three drivers for the joystick device; one of them is a class / miniclass pair.

The function driver, the **main driver for the joystick device**, is the HID class driver/HID USB miniclass driver pair. (HID represents "Human Interface Device".) The HID USB miniclass driver supports the USB-specific semantics of HID devices, relying on the HID class driver DLL for general HID support.

A function driver can be specific to a particular device, or, as in the case of HID, (**Human Interface Device**) a function driver can service a group of devices.

In this example, the HID class driver/HID USB miniclass driver pair services any HID-compliant device in the system on a USB bus.

A HID class driver/HID 1394 miniclass driver pair would service any HID-compliant device on a 1394 bus.

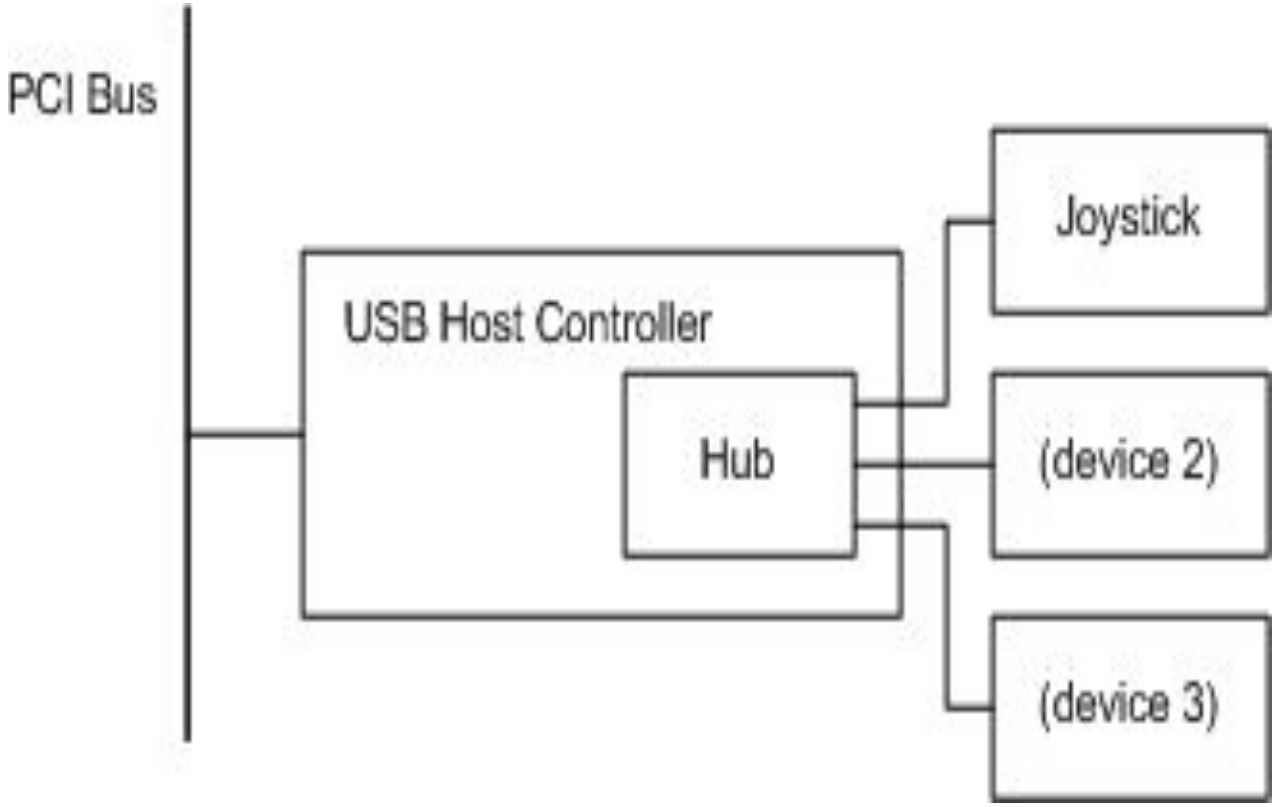
A function driver can be written by the device vendor or by Microsoft.

In this example, the function driver (the HID class/HID USB miniclass driver pair) is written by Microsoft.

There are two filter drivers for the joystick device in this example: an upper-level class filter that adds a macro button feature and a lower-level device filter that enables the joystick to emulate a mouse device.

The upper-level filter is written by someone who needs to filter the joystick I/O and the lower-level filter driver is written by the joystick vendor.

The kernel-mode and user-mode HID clients and the application are not drivers but are shown for completeness.



The figure shows a sample PnP hardware configuration for a USB joystick.

Illustration of WDM (Windows Driver Model) Driver Layers

In this figure, the USB joystick plugs into a port on a USB hub. The USB hub in this example resides on the USB Host Controller board and is plugged into the single port on the USB host controller board. The USB host controller plugs into a PCI bus. From a PnP perspective, the USB hub, the USB host controller, and the PCI bus are all bus devices because they each provide ports. The joystick is not a bus device.

KERNEL

In computing, the **kernel** is a computer program that manages I/O (input/output) requests from software, and translates them into data processing instructions for the central processing unit and other electronic components of a computer.

The kernel is a fundamental part of a modern computer's operating system.

i.e. A kernel connects the application software to the hardware of a computer.

Kernel mode, also referred to as system mode, is one of the two distinct modes of operation of the CPU (central processing unit) in Linux.

The other is **user mode**, a non-privileged mode for user programs, that is, for everything other than the kernel.

When the CPU is in kernel mode, it is assumed to be executing trusted software, and thus it can execute any instructions and reference any memory addresses (i.e., locations in memory).

The kernel (which is the core of the operating system and has complete control over everything that occurs in the system) is *trusted* software, but all other programs are considered *un-trusted* software.

Thus, all user mode software must request use of the kernel by means of a *system call* in order to perform privileged instructions, such as *process* creation or *input/output* operations.

The critical code of the kernel is usually loaded into a *protected area* of memory, which prevents it from being overwritten by other, less frequently used parts of the operating system or by applications.

The kernel performs its tasks, such as executing processes and handling interrupts, in *kernel space*, whereas everything a user normally does, such as writing text in a text editor or running programs in a GUI (graphical user interface), is done in *user space*.

This separation is made in order to prevent user data and kernel data from interfering with each other and thereby diminishing performance or causing the system to become unstable (and possibly crashing).

The OS Kernel

- The internal part of the OS is often called the ***kernel***
- Kernel Components
  - File Manager
  - Device Drivers
  - Memory Manager
  - Scheduler
  - Dispatcher

OS File Manager

- Maintains information about the files that are available on the system
- Where files are located in mass storage, their size and type and their protections, what part of mass storage is available.
- Files usually allowed to be grouped in ***directories*** or ***folders***. Allows hierarchical organization.

OS Device Drivers

- Software to communicate with peripheral devices or controllers
- Each driver is unique.
- Translates general requests into specific steps for that device.

OS Memory Manager

- Responsible for coordinating the use of the machine's main memory.
- Decides what area of memory is to be allocated for a program and its data.
- Allocates and deallocates memory for different programs and always knows what areas are free.

OS Scheduler

- Maintains a record of processes that are present, adds new processes, removes completed processes
  - memory area(s) assigned

- priority
- state of readiness to execute (ready/wait)

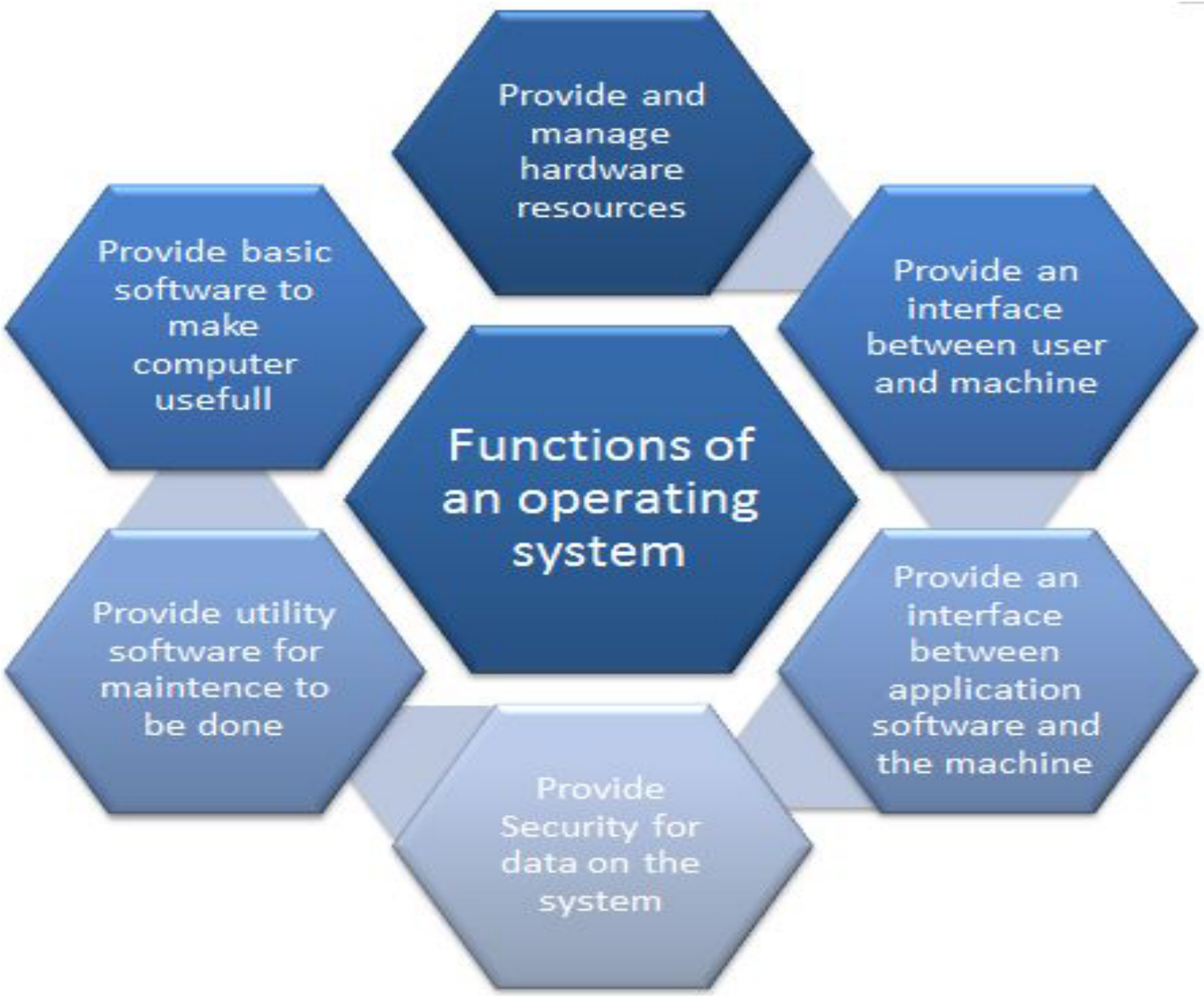
OS Dispatcher

- Ensures that processes that are ready to run are actually executed.
- Time is divided into small (50 ms) segments called a ***time slice***
- When the time slice is over, the dispatcher allows scheduler to update process state for each process, then selects the next process to run.

OS Summary

- Shell -- Interface to user.
- File Manager -- Manages Mass (bulk) Memory.
- Device Drivers -- Communicate with Peripherals.
- Memory Manager -- Manages Main Memory.
- Scheduler & Dispatcher -- Manage Processes.

Functions of OS



There are Many Functions those are Performed by the Operating System But the Main Goal of Operating System is to **Provide the Interface between the user and the hardware** i.e. Provides the Interface for Working on the System by the user.

Operating System will **Manages all the Resources** of the Computer System those are attached to the System.

e.g. Memory , Processor , all the I/O Devices etc.

The Operating System will identify at which Time the CPU will perform which Operation and in which Time the Memory is used by which Programs.

Also which Input Device will respond to which Request of the user means When the Input and Output Devices are used by the which Programs.

Storage Management

Operating System also Controls all the Storage Operations .

Means how the data or files will be stored into the computers and how the Files will be accessed by the users etc.

All the operations those are responsible for storing and accessing the files is determined by the Operating System .

Operating System also allows Creation of Files, Creation of Directories and Reading and Writing the data of Files and Directories and also copy the contents of the Files and the Directories from One Place to Another Place.

Process Management

The Operating System also treats the process management means all the Processes those are given by the user or the Process those are System ‘s own Process are handled by the Operating System .

The Operating System will create the priorities for the user and also start or stops the execution of the process.

Also makes the Child Process after dividing the Large Processes into the Small Processes.

Memory Management

Operating System also manages the memory of the Computer System means provide the memory to the process .

Also de-allocate the memory from the Process if a Process gets completed .

Extended Machine : Operating System also behaves like an Extended Machine means Operating system also Provides us Sharing of Files between Multiple Users, also Provides Some Graphical Environments and also Provides Various Languages for Communications and also Provides Many Complex Operations like using Many Hardware's and Software's.

Mastermind: Operating System also performs many functions and for those reasons we can say that Operating System is a Mastermind. It provides Booting without an Operating System and provides facility to increase the Logical Memory of the Computer System by using the Physical Memory of the Computer System and also provides various Types of Formats Like NTFS and FAT File Systems.

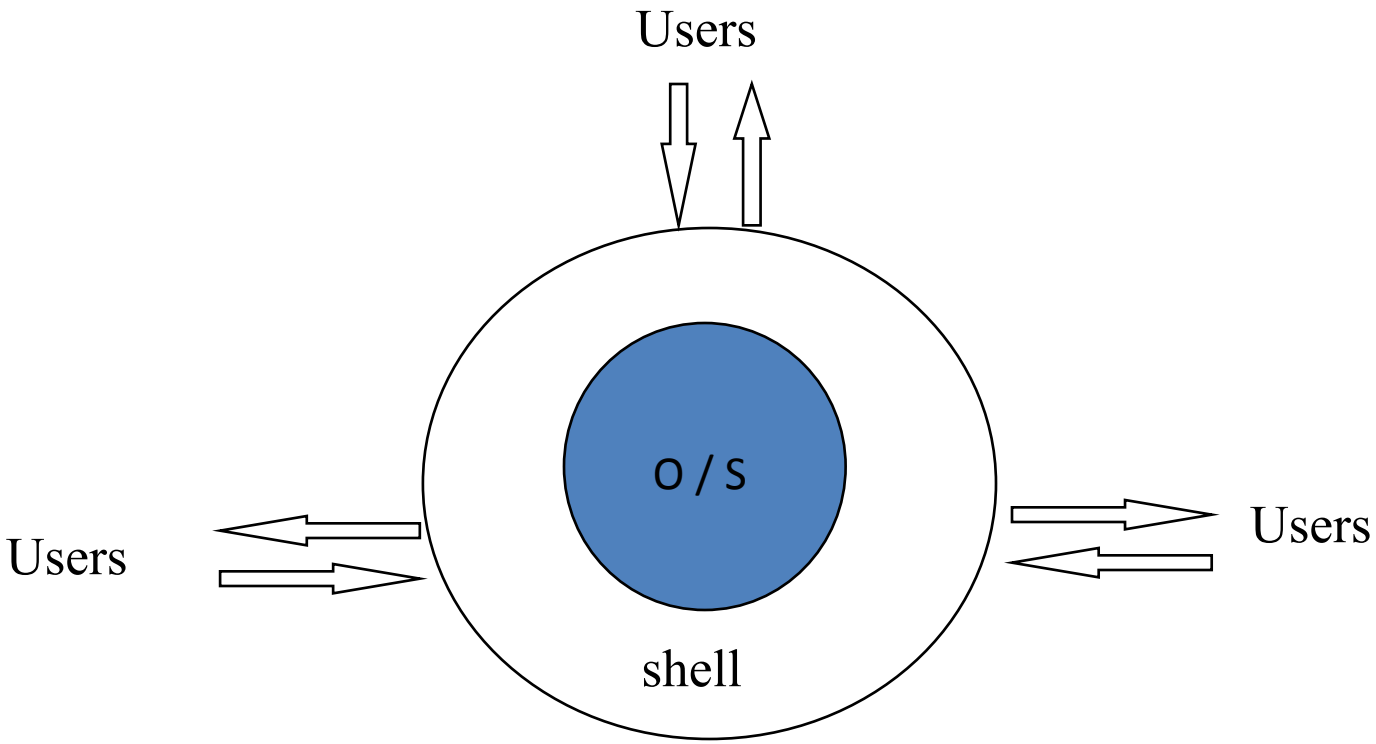
Operating System also controls the errors those have been occurred into the program and also provides recovery of the system when the system gets damaged. Means when due to some Hardware Failure , if system doesn't works properly then this recover the system and also correct the system and also provides us the Backup Facility. And Operating System also breaks the large program into the Smaller Programs those are also called as the threads. And execute those threads one by one.

Computer applications today require a single machine to perform many operations and the applications may compete for the resources of the machine.

This demands a high degree of coordination

This coordination is handled by system software known as the ***operating system***

OS Shell interface



OS for batch jobs

- Program execution required may require equipment.
- OS was a system to simplify program setup & simplify transition between jobs.
- Physical separation of users and equipment led to computer operators.

  (responsible for)
- Users left jobs with the operator and came back the next day (batch jobs).
- Users had no interaction with computer during program execution.
- Some applications may require interaction.

OS for Interactive Processing

- Allowed programs to carry on dialogue with user via remote terminals

  (workstations) Real-time processing
- Users demand timely response
- Machines too expensive to serve only one user.
- Common for several users to want interactive services at the same time.

A real-time operating system (**RTOS**) is an operating system (OS) intended to serve real-time application process data **as it comes in**, typically **without buffering delays**. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter.

OS for time-sharing

- To accommodate multiple real-time users, the OS rotates its various jobs in and out of execution via ***time-sharing.***
- Each job gets a predetermined "time slice"
- At end of time slice current job is set aside and a new one starts.
- By rapidly shuffling jobs, illusion (feeling) of several jobs executing simultaneously is created.
- Without time slicing, a computer spends most of its time waiting for peripheral devices or users.
- A collection of tasks can be completed in less time with time-sharing than when completed sequentially.

A system is said to be **Real Time** if it is required to complete it's work & deliver it's services on time. A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task.

e.g. – Flight Control System [All tasks in that system must execute on time.]

A real-time operating system **(RTOS)** is a multitasking operating system designed for real-time applications. Such applications include embedded systems, industrial robots, scientific research equipment and others.

- **Hard Real Time System**
  - Failure to meet deadlines is fatal (dangerous)
  - example : Flight Control System
- **Soft Real Time System**
  - Late completion of jobs is undesirable but not fatal.
  - System performance degrades as more & more jobs miss deadlines
  - Online Databases

Scheduling Algorithms in RTOS

- **Clock Driven Scheduling**
- **Weighted Round Robin Scheduling**
- **Priority Scheduling**

  (Greedy / List / Event Driven)
- **Clock Driven**
  - All parameters about jobs (release time/ execution time/deadline) known in advance.
  - Schedule can be computed offline or at some regular time instances.
  - Minimal runtime overhead.
  - Not suitable for many applications.

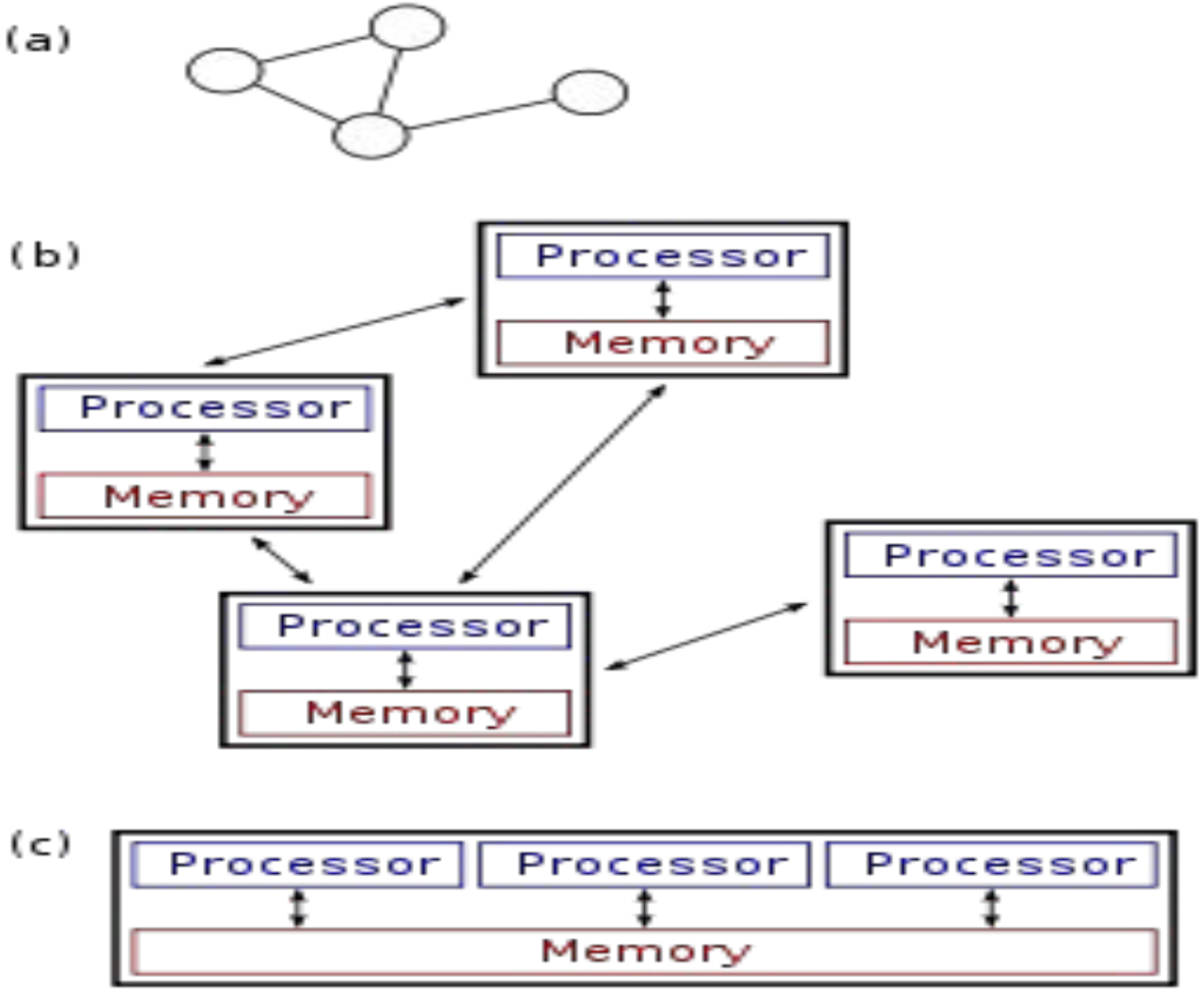
- **Weighted Round Robin**
  - Jobs scheduled in FIFO manner
  - Time quantum given to jobs is proportional to it's weight
  - Example use : High speed switching network
    - QOS guarantee.
  - Not suitable for precedence constrained jobs.
    - Job A can run only after Job B. No point in giving time quantum to Job B before Job A.

- **Priority Scheduling**

  (Greedy/List/Event Driven)

  - Processor never left idle when there are ready tasks
  - Processor allocated to processes according to priorities
  - Priorities — Static - at design time; Dynamic - at runtime

- **Earliest Deadline First (EDF)**
  - Process with earliest deadline given highest priority
- **Least Slack Time First (LSF)**
  - slack = relative deadline – execution left
- **Rate Monotonic Scheduling (RMS)**
  - For periodic tasks
  - Tasks priority inversely proportional to it's period



a)–(b) A distributed system.
(c) A parallel system.

A **distributed system is a collection of autonomous** computers linked by a computer network that appear to the users of the system as a single computer.

Definition : A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility.

By running a distributed system software the computers are enabled to:

- coordinate their activities
- share resources: hardware, software, data.

Centralised System Characteristics

- One component with non-autonomous parts.
- Component shared by users all the time.
- All resources accessible.
- Software runs in a single process.
- Single Point of control.
- Single Point of failure.

Distributed System Characteristics

- Multiple autonomous components.
- Components are not shared by all users.
- Resources may not be accessible.
- Software runs in concurrent processes on different processors.
- Multiple Points of control.
- Multiple Points of failure.

Common Characteristics

- Resource Sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance
- Transparency

Resource Sharing

- Ability to use any hardware, software or data anywhere in the system.
- Resource manager controls access, provides naming scheme and controls concurrency.
- Resource sharing model (e.g. client / server or object-based) describing how resources are provided, they are used & provider and user interact with each other.

Openness

- Openness is concerned with extensions & improvements of distributed systems.
- Detailed interfaces of components need to be published.
- New components have to be integrated with existing components.
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved.

Concurrency

- Components in distributed systems are executed in concurrent processes.
- Components access and update shared resources (e.g. variables, databases, device drivers).
- Integrity of the system may be violated if concurrent updates are not coordinated.
- Lost updates.
- Inconsistent analysis.

Scalability

- Adaption of distributed systems to
  - accommodate more users.
  - respond faster (this is the hard one)
- Usually done by adding more and/or faster processors.
- Components should not need to be changed when scale of a system increases.
- Design components to be scalable!

Fault Tolerance

- Hardware, software and networks fail!
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability.
- Fault tolerance is achieved by
  - recovery
  - redundancy

Transparency

- Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.
- Transparency has different dimensions that were identified by ANSA.
- These represent various properties that distributed systems should have.

Access Transparency

- Enables local and remote information objects to be accessed using identical operations.
- Example: File system operations in NFS.
- Example: Navigation in the Web.
- Example: SQL Queries

Location Transparency

- Enables information objects to be accessed without knowledge of their location. e.g. File system operations in NFS
- Example: Pages in the Web
- Example: Tables in distributed databases.

Concurrency Transparency

- Enables serveral processes to operate concurrently using shared information objects without interference between them. e.g. NFS , Automatic teller machine network , Database management system

Replication Transparency

- Enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs
- e.g. Distributed DBMS
- e.g. Mirroring Web Pages.

Failure Transparency

- Enables the concealment of faults
- Allows users and applications to complete their tasks despite the failure of other components.

e.g. : Database Management System

Migration Transparency

- Allows the movement of information objects within a system without affecting the operations of users or application programs

e.g. NFS , Web Pages

Performance Transparency

- Allows the system to be reconfigured to improve performance as loads vary.

e.g. Distributed make.

Scaling Transparency

- Allows the system and applications to expand in scale without change to the system structure or the application algorithms.

e.g. World-Wide-Web , Distributed Database