Operating System

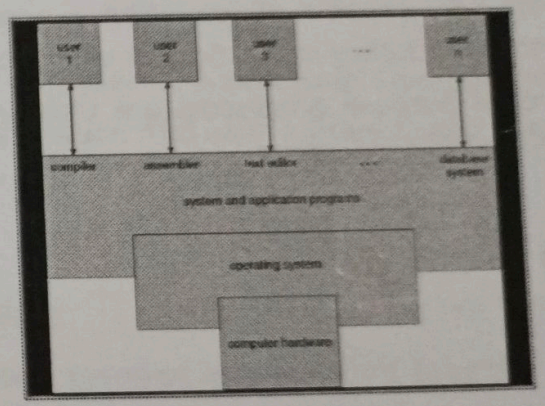
Definition

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- It is a program that manages the computer hardware.
- The purpose of an OS is to provide an environment in which a user can execute programs in a convenient and efficient manner.

Role of OS in Computer System

- A computer system consists of four components.
  - Hardware
  - Operating System
  - Application program
  - The users

Abstract View of Components of a computer system



- ***Hardware*** provides the basic computing resources for the system.
- ***Application program*** defines the ways in which these resources are used to solve users computing problem.
- The ***OS*** controls and coordinates the use of the hardware among the various application programs for the various ***users***.

Computer System Architecture

- Types of computer system based on number of general purpose processors.
  - Single processor systems
  - Multiprocessor systems
  - Clustered systems

Single processor system

- One main CPU capable of executing a general purpose instruction set including instruction for user processes.
- Other special purpose processors are also present which perform device specific task.

Multiprocessor Systems

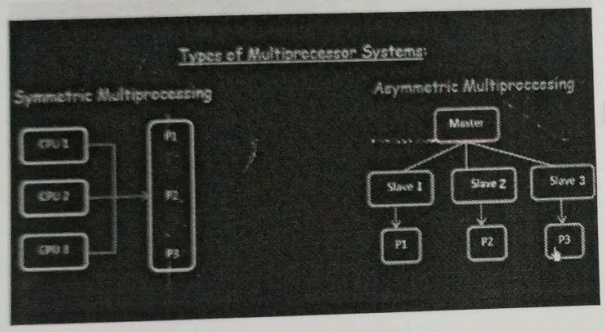
- Also known as parallel systems or tightly coupled systems.
- Has two or more processors in close communication, sharing the computer bus and sometimes the clock, memory and peripherals devices.
- Advantages
  - Increased Throughput
  - Economy of Scale
  - Increased Reliability

ecture

n number of

Types of MPS

- Symmetric Multiprocessing
- Asymmetric Multiprocessing



Clustered systems

- Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.
- They are composed of two or more individual systems coupled together.
- Provides high availability
- Can be structured asymmetrically or symmetrically

operations

Functions of OS

operations Function.

- Memory management
  - The o/s keeps track of the memory, which parts are in use and by whom.
- Process management
  - The o/s keeps track of processors and the status of processes. It decides who will have a chance to use the processor
- Device management
  - The o/s keeps track of the devices, channels, control units and decides what is an efficient way to allocate the device.
- Information management
  - O/S keeps track of the information,
  - its location, use, status etc. and decides who gets use of the resources,
  - enforce protection requirements
- Protection

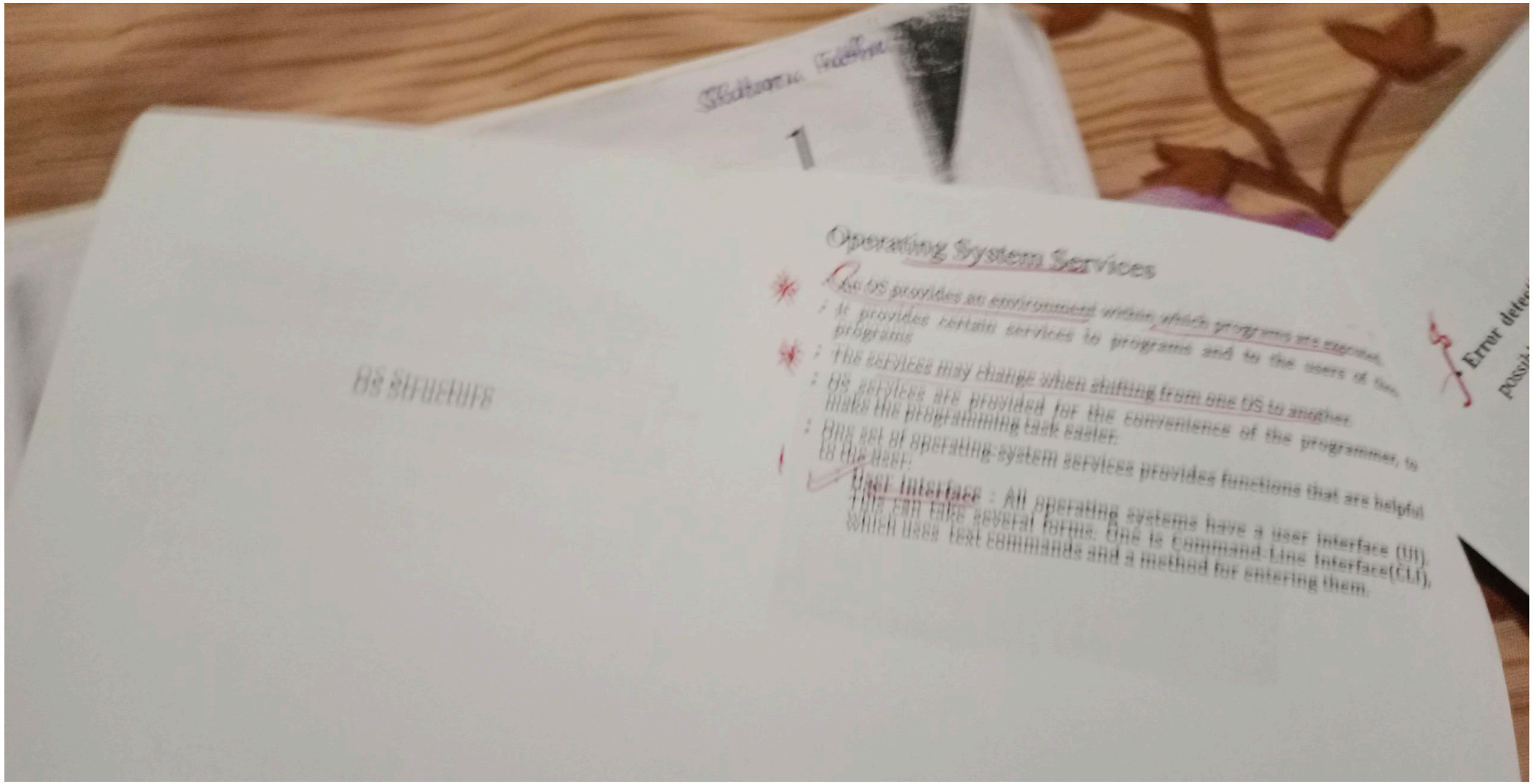
  -An o/s is to protect the user from unauthorized access of his files or data. And also it should protect itself from users
- Error Handling

  -An o/s must respond to errors by taking the appropriate actions.

* File managent

* Job Accounting

Fode:


OS structure
1
Operating System Services
Error dete

- **Batch interface**, in which commands and directives to control those commands are entered into files and those files are executed.
- **Graphical User Interface** (GUI), here the interface is a window system with a pointing device to direct I/O, choose from menus and make selections and a keyboard to enter text.

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

- I/O operations - A running program may require I/O, which may involve a file or an I/O device. For efficiency and protection, users cannot control I/O devices directly, so OS must provide a means to do I/O.
- File-system manipulation - Programs need to read and write files and directories, create and delete them by name, search for a given file, list file Information, permission management to allow or deny access to files or directories.
- 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.

re executed.
sers of those

- **Error detection** – OS needs to be constantly aware of possible errors. 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.

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing.
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them. Many types of resources are managed by the OS.
  - 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. This record keeping may be used for accounting.

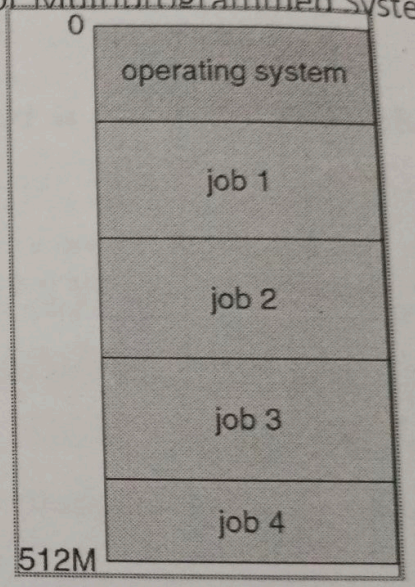
(11)

- **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - **Protection** involves ensuring that all access to system resources is controlled
  - **Security** of the system from outsiders requires user authentication, usually by means of password, to gain access to system resources.
  - Extends to defending external I/O devices from invalid access attempts including modems and network adaptors.
  - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

Operating System Structure

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Memory Layout for Multiprogrammed System



User OS interface

- There are two approaches for users to interface with the operating system.
  - Command line interface or command interpreter
  - Graphical user interface

Command Line Interface

CLI allows users to directly enter commands that are to be performed by OS.

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- The main function of CI is to get and execute a command from user. Commands at this level manipulates files- create, delete, list, print, copy, execute and so on.
- There are two ways in which these commands can be implemented.
  - Sometimes command interpreter itself contains the code to execute the command.

tem

- In the second method, most commands are implemented through system programs.
- The CI uses the command to identify a file to be loaded into memory and executed.

Graphical user interface

- User-friendly **desktop** metaphor interface
  - Provides mouse based window and menu system as an interface.
  - Mouse is moved to position its pointer on images or icons.
  - **Icons** represent files, programs, actions, etc
  - Depending on the mouse pointers location, clicking a button on the mouse can invoke a program, select a file or directory or pull down a menu that contains commands.
  - Invented at Xerox PARC

- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)
- The user interface can vary from system to system and from user to user within a system

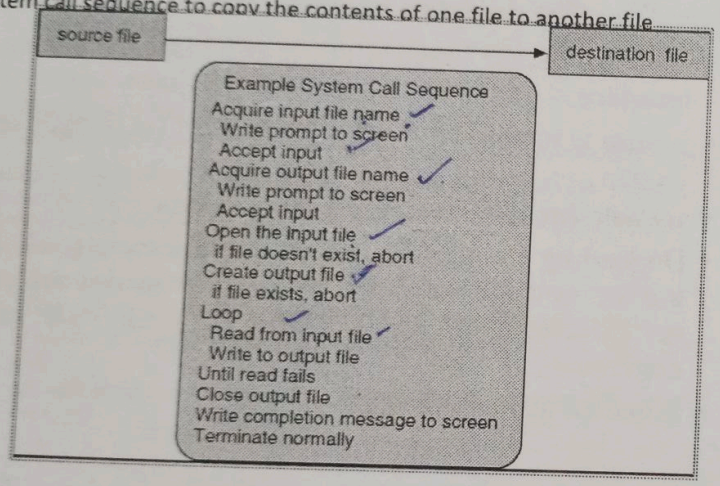
Imp

System Calls

- Programming interface to the services made available by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems, and Java API for designing programs that run on the Java virtual machine(JVM).

Example of how System Calls are used

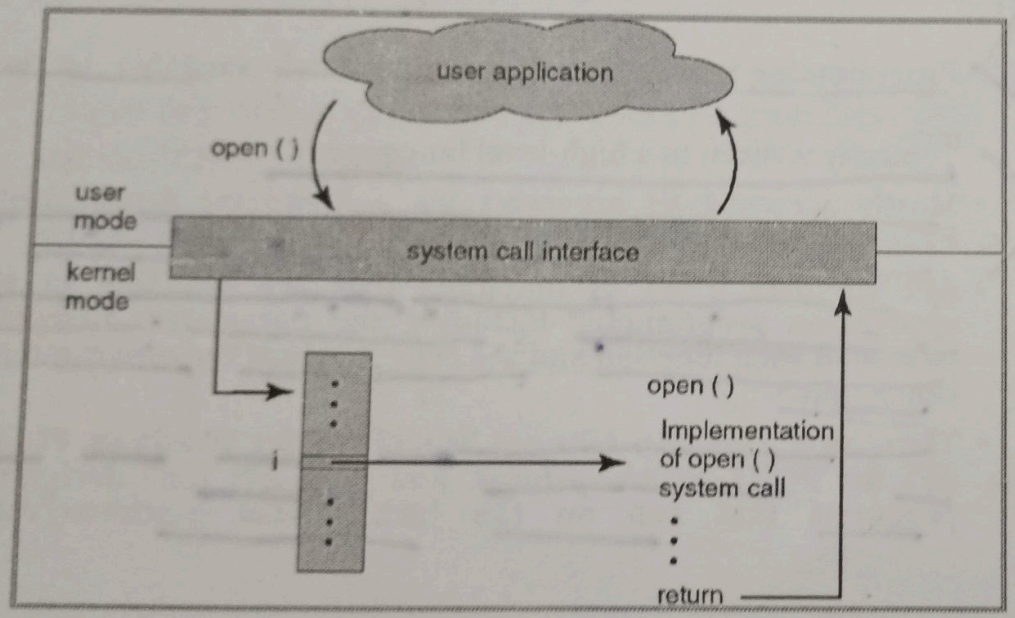
- System call sequence to copy the contents of one file to another file



System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- 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
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Figure 2.8 Types of system calls.

- **Process Control**
- A running program needs to be able to stop execution either normally (end)or abnormally(abort).
- If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, often a dump of memory is taken and an error message generated.
- The dump is written to the disk and may be examined by a debugger.
- Debugger?

- Under normal or abnormal circumstances, the OS must transfer control to the command interpreter.
- In GUI system, a popup window will alert the user to the error and ask for guidance.
- A process or job executing one program may want to ***load*** and ***execute*** another program.
- This allows the command interpreter to execute the program.
- When the newly loaded program terminates , the control returns to the existing program.
- If both programs continue concurrently, we have created a new job or process to be multiprogrammed. (***create process)***

- If we create new job or process, we should be able to control its execution.
- The control requires the ability to determine and reset the attributes of a job or process.

  (get process attribute, set process attribute)
- We may want to terminate a job or process

  (terminate process)
- Having created new jobs or processes, we may need to wait for them to finish their execution.

  (wait time, wait event, signal event)

- **File Management**
- Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*. Also, there is a need to determine the file attributes – *get* and *set* file attribute.
- Many times the OS provides an API to make these system calls.

- **Device Management**
- Process usually require several resources to execute, if the resources are available, they will be granted and control returned to the user process.
- These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.
- User programs ***request*** the device, and when finished they ***release*** the device.
- Once the device has been requested, we can *read*, *write*, and *reposition* the device.

- **Information Management**
- Some system calls exist purely for transferring information between the user program and the operating system.
- An example of this is ***time***, or ***date***.
- The OS also keeps information about all its processes and provides system calls to report this information.

- **Communication**
- There are two models of interprocess communication, the message-passing model and the shared memory model.
- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to ***create*** and gain access to regions of memory owned by other processes.
- The two processes exchange information by reading and writing in the shared data.

execute, if the
control

OS Operations

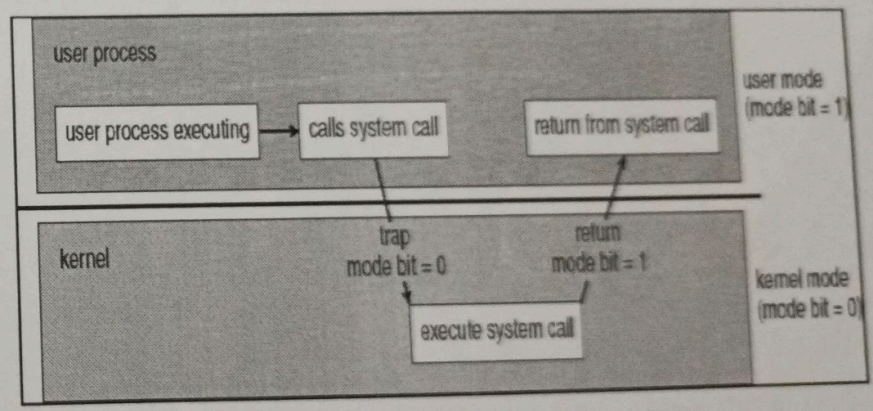
- Modern OS are interrupt driven.
- An interrupt is a hardware mechanism that enables a device to notify the CPU.
- The occurrence of an event is signaled by an interrupt either from the hardware or from the software.
- A ***trap*** or an ***exception*** is a software generated interrupt caused either by an error or by a user request.
  - Other process problems include infinite loop, processes modifying each other or the operating system

Dual mode operation

- In order to ensure proper execution of OS, we must be able to distinguish between the execution of OS code and user defined code.
- **Dual-mode** operation allows OS to protect itself and other system components. The two separate modes of operation are
  - **User mode** and **kernel mode**
- A bit called **mode bit** is added to the hardware of the computer to indicate the current mode: kernel(0),user(1).
- With mode bit we are able to distinguish between a task that is executed on behalf of OS and one that is executed on behalf of the user.

- When the computer system is executing on behalf of user application, the system is in user mode.
- When a user application requests a service from the OS, a transition from user to kernel mode occurs to fulfill the request.

Transition from user to kernel mode



- At system boot time , the hardware starts in kernel mode.
- The OS is then loaded and starts user application in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode.
- Whenever the OS gains control of the computer, it is in kernel mode.
- The system always switches to user mode before passing control to a user program.

- The protection of OS in dual mode operation is accomplished by designating 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 in user mode, the hardware does not execute the instruction but treat it as illegal and trap it to the OS.
- The instruction to switch to user mode is a privileged instruction.