## 1. Operating system objectives and functions

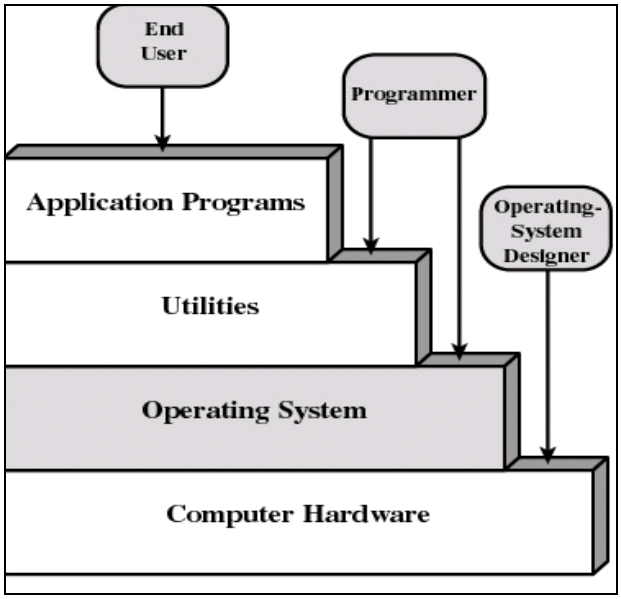
An Operating System is a software program or set of programs that mediate access between physical devices (such as a keyboard, mouse, monitor, disk drive or network connection) and application programs (such as a word processor, World-Wide Web browser or electronic mail client).

**Some characteristics of an Operating System are:**

- Whether multiple programs can run on it simultaneously: *multi-tasking*
- Whether it can take advantage of multiple processors: *multi-processing*
- Whether multiple users can run programs on it simultaneously: *multi-user*
- Whether it can reliably prevent application programs from directly accessing hardware devices: *protected*
- Whether it has built-in support for graphics.
- Whether it has built-in support for networks.

**Some popular Operating Systems are:**

- UNIX: multi-tasking, multi-processing, multi-user, protected, with built-in support for networking but not graphics.
- Windows NT: multi-tasking, multi-processing, single-user, protected, with built-in support for networking and graphics.
- Windows 95/98: multi-tasking, multi-processing, single-user, unprotected, with built-in support for networking and graphics.
- Windows 3.x: single-tasking, single-processing, single-user, unprotected, with built-in support for graphics but not networking.
- DOS: single-tasking, single-processing, single-user, unprotected with no built-in support for graphics or networking.
- NetWare: multi-tasking, multi-processing, single-user, unprotected, with built-in support for networking but not graphics.




Layered view of computer system

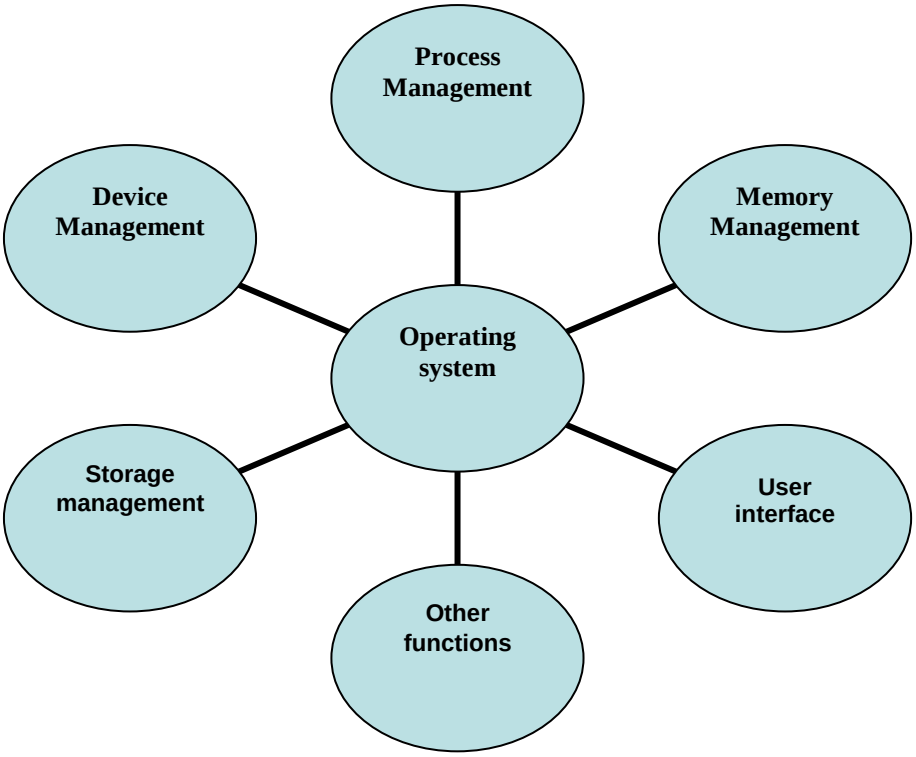
## 1.2 Objectives of Operating Systems:

Modern Operating systems generally have following three major goals. Operating systems generally accomplish these goals by running processes in low privilege and providing service calls that invoke the operating system kernel in high-privilege state.

- **To hide details of hardware by creating abstraction**
  An abstraction is software that hides lower level details and provides a set of higher-level functions. An operating system transforms the physical world of devices, instructions, memory, and time into virtual world that is the result of abstractions built by the operating system. There are several reasons for abstraction.
  *First*, the code needed to control peripheral devices is not standardized. Operating systems provide subroutines called device drivers that perform operations on behalf of programs for example, input/output operations.
  *Second*, the operating system introduces new functions as it abstracts the hardware. For instance, operating system introduces the file abstraction so that programs do not have to deal with disks.
  *Third*, the operating system transforms the computer hardware into multiple virtual computers, each belonging to a different program. Each program that is running is called a process. Each process views the hardware through the lens of abstraction.
  *Fourth*, the operating system can enforce security through abstraction.
- **To allocate resources to processes (Manage resources)**
  An operating system controls how **processes** (the active agents) may access **resources** (passive entities).
- **Provide a pleasant and effective user interface**
  The user interacts with the operating systems through the user interface and usually interested in the "look and feel" of the operating system. The most important components of the user interface are the command interpreter, the file system, on-line help, and application integration. The recent trend has been toward increasingly integrated graphical user interfaces that encompass the activities of multiple processes on networks of computers.

One can view Operating Systems from two points of views: **Resource manager** and **Extended machines**. Form Resource manager point of view Operating Systems manage the different parts of the system efficiently and from extended machines point of view Operating Systems provide a virtual machine to users that is more convenient to use. The structurally Operating Systems can be design as a monolithic system, a hierarchy of layers, a virtual machine system, an exokernel, or using the client-server model. The basic concepts of Operating Systems are processes, memory management, I/O management, the file systems, and security.

**1.3 The main functions of an OS are schematically shown below:**



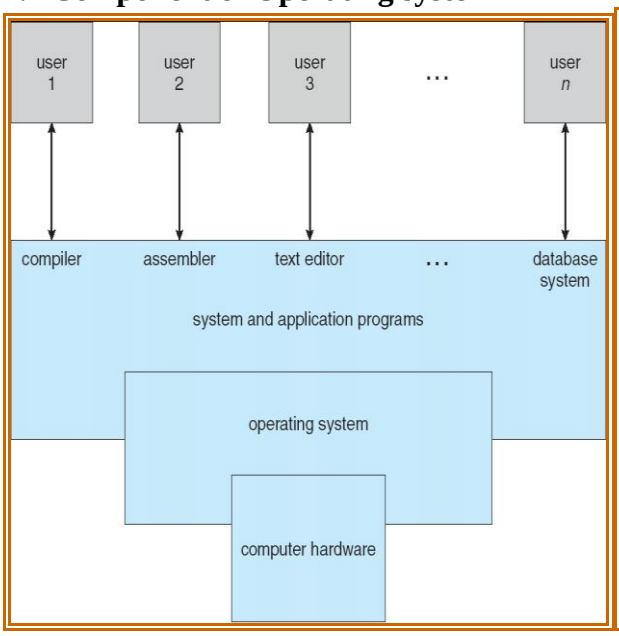

**The main functions of an OS include:**

- In a multitasking operating system where multiple programs can be running at the same time, the operating system determines which applications should run in what order and how much time should be allowed for each application before giving another application a turn.
- It manages the sharing of internal memory among multiple applications.
- It handles and monitors input and output to and from attached hardware devices, such as hard disks, printers, and dial-up ports.
- It sends messages to each application or interactive user (or to a system operator) about the status of operation and any errors that may have occurred.
- It can offload the management of what are called *batch* jobs (for example, printing) so that the initiating application is freed from this work.
- On computers that can provide parallel processing, an operating system can manage how to divide the program so that it runs on more than one processor at a time.
- Scheduling the activities of the CPU and resources to achieve efficiency and prevention of deadlock.

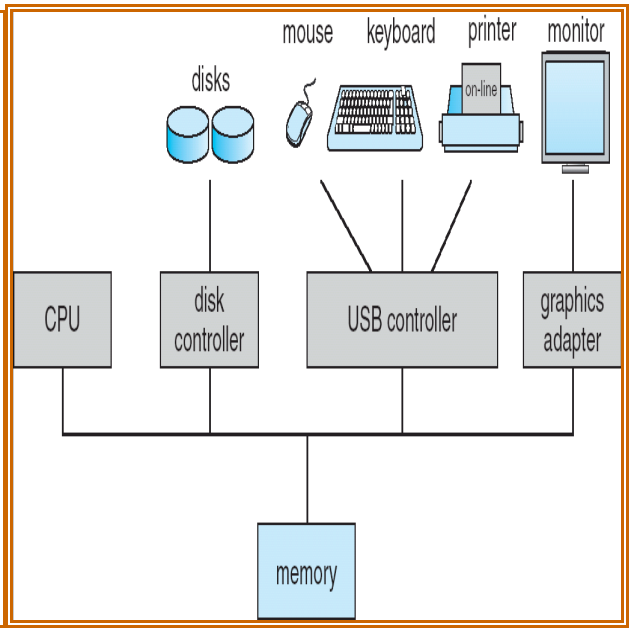
**This includes all the functions below:**

- Creating a file system
- Copying, deleting, moving files
- Multitasking programmes
- Starting the computer
- Interfacing with the hardware
- Programme intercommunication
- Networking

## 2. Component of Operating system




**With respect to User and software**




**With respect to Hardware**

## 3. Evolution of Operating Systems

**3.1 Serial Processing -** 1940's – 1950's programmer interacted directly with hardware. No operating system.

**Problems:**

**Scheduling -** users sign up for machine time. Wasted computing time

**Setup Time-** Setup included loading the compiler, source program, saving compiled program, and loading and linking. If an error occurred - start over.

### 3.2 Simple Batch Systems

Simple Batch Systems improve the utilization of computers**.**

Jobs were submitted on cards or tape to an operator who batches jobs together sequentially. The program that controls the execution of the jobs was called **monitor** - a simple version of an operating system. The interface to the monitor was accomplished through Job Control Language (JCL). For example, a JCL request could be to run the compiler for a particular programming language, then to link and load the program, then to run the user program.

The batch processing methodologies evolved to decrease these dead times, queuing up programs so that as soon as one completed, the next would start.

To support a batch processing operation, a number of card punch or paper tape writers would be used by programmers, who would use these inexpensive machines to write their programs "offline". When they completed typing them, they were submitted to the operations team, who would schedule them for running. Important programs would be run quickly, less important ones were unpredictable. When the program was finally run, the output, generally printed, would be returned to the programmer. The complete process might take days, during which the programmer might never see the computer.

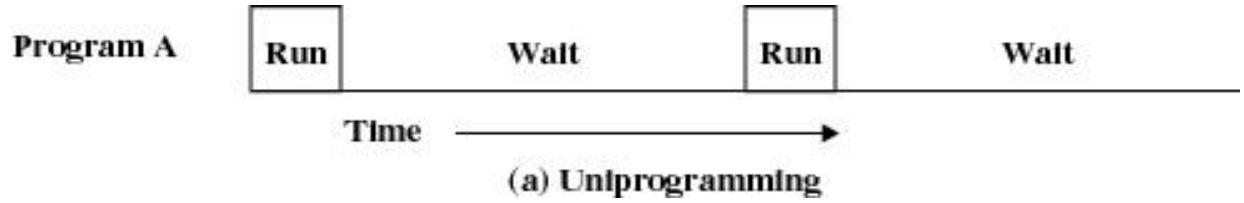
The basic physical layout of the memory of a batch job computer is shown below:

```
-------------------------------------
|                                   |
|   Monitor (permanently resident)  |
|                                   |
-------------------------------------
|                                   |
|             User Space            |
| (compilers, programs, data, etc.) |
|                                   |
-------------------------------------
```

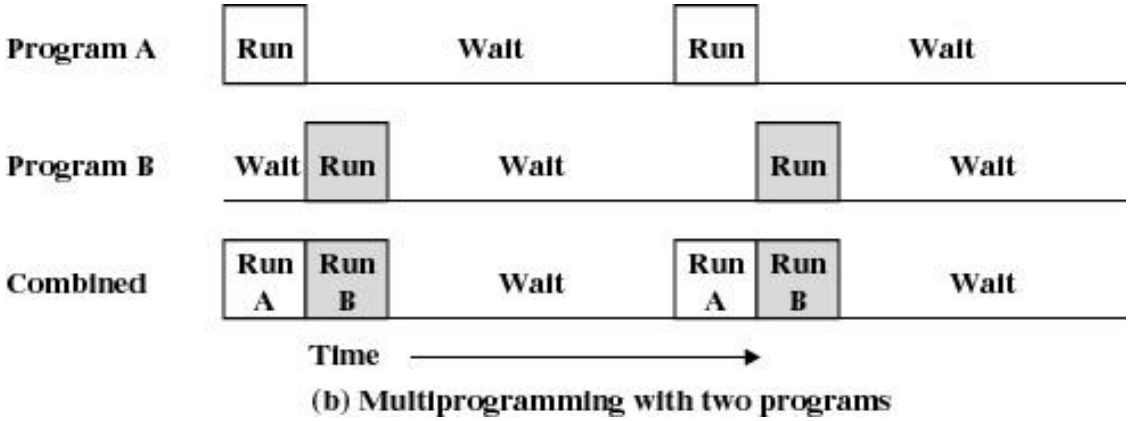
**Hardware features:**
Memory protection: do not allow the memory area containing the monitor to be altered
Timer: prevents a job from monopolizing the system

1. Advantages of batch systems
    - move much of the work of the operator to the computer
    - increased performance since it was possible for job to start as soon as the previous job finished
2. Disadvantages
    - turn-around time can be large from user standpoint
    - more difficult to debug program
    - due to lack of protection scheme, one batch job can affect pending jobs (read too many cards, etc)
    - a job could corrupt the monitor, thus affecting pending jobs
    - a job could enter an infinite loop

**3.3 Multiprogrammed Batch Systems**
More than one program resides in the main memory. While a program A uses an I/O device the processor does not stay idle, instead it runs another program B.



(a) Uniprogramming



(b) Multiprogramming with two programs

**New features**:
Memory management - to have several jobs ready to run, they must be kept in main memory
Job scheduling - the processor must decide which program to run.

### 3.4 Multiprogramming system (1960-Present day)

A group of jobs that are ready to be executed is called job pool. Since there is more than one job that can be executed, it is possible for the operating system to make a decision about which job to execute next. That decision keeps CPU utilization as high as possible.
In general, it is not possible for a single user to keep CPU or I/O devices busy at all times. Multiprogramming allows the system to increase CPU utilization by ensuring that the CPU always has a job to execute.
The CPU has a pool of jobs. When the currently executing job has to wait (if it is performing some I/O), it is removed from the CPU- Another job is selected and the CPU now executes it. This process ensures that CPU is always executing a job if there is a job to execute. In a non-multiprogrammed system, if a job had to wait for an I/O operation, CPU would also have to wait until I/O was finished.
CPU requires sophisticated data structures to implement multiprogramming. CPU must be able to decide which job is to be executed next. Some jobs will fit into memory at once. Some jobs may have to remain on disk. The operating system must be able to perform some sort of scheduling on the jobs in job pool. It determines which jobs will stay on the disk and which ones will be loaded into memory. The operating system must have some form of memory management. Memory management is used to keep track of which jobs are stored, where and how much space is available.

```
-------------------------------------
|              Monitor              |
|   (more like a operating system)  |
-------------------------------------
|         User program 1            |
-------------------------------------
|         User program 2            |
-------------------------------------
|         User program 3            |
-------------------------------------
|         User program 4            |
-------------------------------------
```

### 3.5 Time-Sharing Systems( 1970 - present day)

Multiprogramming systems : several programs use the computer system
Time-sharing systems : several (human) users use the computer system interactively.

Time-sharing developed out of the realization that while any single user was inefficient, a large group of users together were not. This was due to the pattern of interaction; in most cases users entered bursts of information followed by long pause, but a group of users working at the same time would mean that the pauses of one user would be used up by the activity of the others. Given an optimal group size, the overall process could be very efficient. Similarly, small slices of time spent waiting for disk, tape, or network input could be granted to other users.

Implementing a system able to take advantage of this would be difficult. Batch processing was really a methodological development on top of the earliest systems; computers still ran single programs for single users at any time, all that batch processing changed was the time delay between one program and the next. Developing a system that supported multiple users at the same time was a completely different concept; the "state" of each user and their programs would have to be kept in the machine, and then switched between quickly. This would take up computer cycles, and on the slow machines of the era this was a concern. However, as computers rapidly improved in speed, and especially in size of core memory in which users' states were retained, the overhead of time-sharing continually decreased, relatively.

The first involved **timesharing** or **timeslicing**. The idea of multiprogramming was extended to allow for multiple terminals to be connected to the computer, with each in-use terminal being associated with one or more jobs on the computer. The operating system is responsible for switching between the jobs, now often called **processes**, in such a way that favored user interaction. If the **context-switches** occurred quickly enough, the user had the impression that he or she had direct access to the computer.

Interactive processes are given a higher **priority** so that when IO is requested (e.g. a key is pressed), the associated process is quickly given control of the CPU so that it can process it. This is usually done through the use of an **interrupt** that causes the computer to realize that an IO event has occurred.

**Characteristics:**

- Using multiprogramming to handle multiple interactive jobs
- Processor’s time is shared among multiple users
- Multiple users simultaneously access the system through terminals

| | **Batch Multiprogramming** | **Time Sharing** |
|---|---|---|
| Principal objective | Maximize processor use | Minimize response time |
| Source of directives to operating system | Job control language commands provided with the job | Commands entered at the terminal |

### 3.6 Multiprocessing system

Multiprocessing refers to an operating situation where the simultaneous processing of programs takes place. This state of ongoing and coordinated processing is usually achieved by interconnecting two or more computer processors that make it possible to use the available resources to best advantage. Many operating systems today are equipped with a multiprocessing capability, although multiprogramming tends to be the more common approach today.

The basic platform for multiprocessing allows for more than one computer to be engaged in the used of the same programs at the same time. This means that persons working at multiple work stations can access and work with data contained within a given program. It is this level of functionality that makes it possible for users in a work environment to effectively interact via a given program.

There are essentially two different types of multiprocessing. **Symmetric multiprocessing**, more than one computer processor will share memory capacity and data path protocols. While the process may involve more than one computer station, only one copy or the operating system will be used to initiate all the orders executed by the processors involved in the connection.

The second approach to multiprocessing is known as massively **parallel processing**. Within this structure, it is possible harness and make use of large numbers of processors in order to handle tasks. Often, this type of multiprocessing will involve over two hundred processors. Within the environment of MPP, each processor works with individual operating systems and memory resources, but will connect with the other processors in the setup to divide tasks and oversee different aspects of transmissions through data paths.

Multiprocessing is a common situation with corporations that function with multiple locations and a large number of employees. The combination of resources that can result from the use of multiple computer processors make it possible to transmit data without regard to distance or location, as well as allow large numbers of users to work with a program simultaneously.

A **multiprocessor** computer is one with more than one CPU. The category of multiprocessor computers can be divided into the following sub-categories:

- **Shared memory multiprocessors** have multiple CPUs, all with access to the same memory. Communication between the the processors is easy to implement, but care must be taken so that memory accesses are synchronized.

- **Distributed memory multiprocessors** also have multiple CPUs, but each CPU has it's own associated memory. Here, memory access synchronization is not a problem, but communication between the processors is often slow and complicated.

**Related to multiprocessors are the following:**

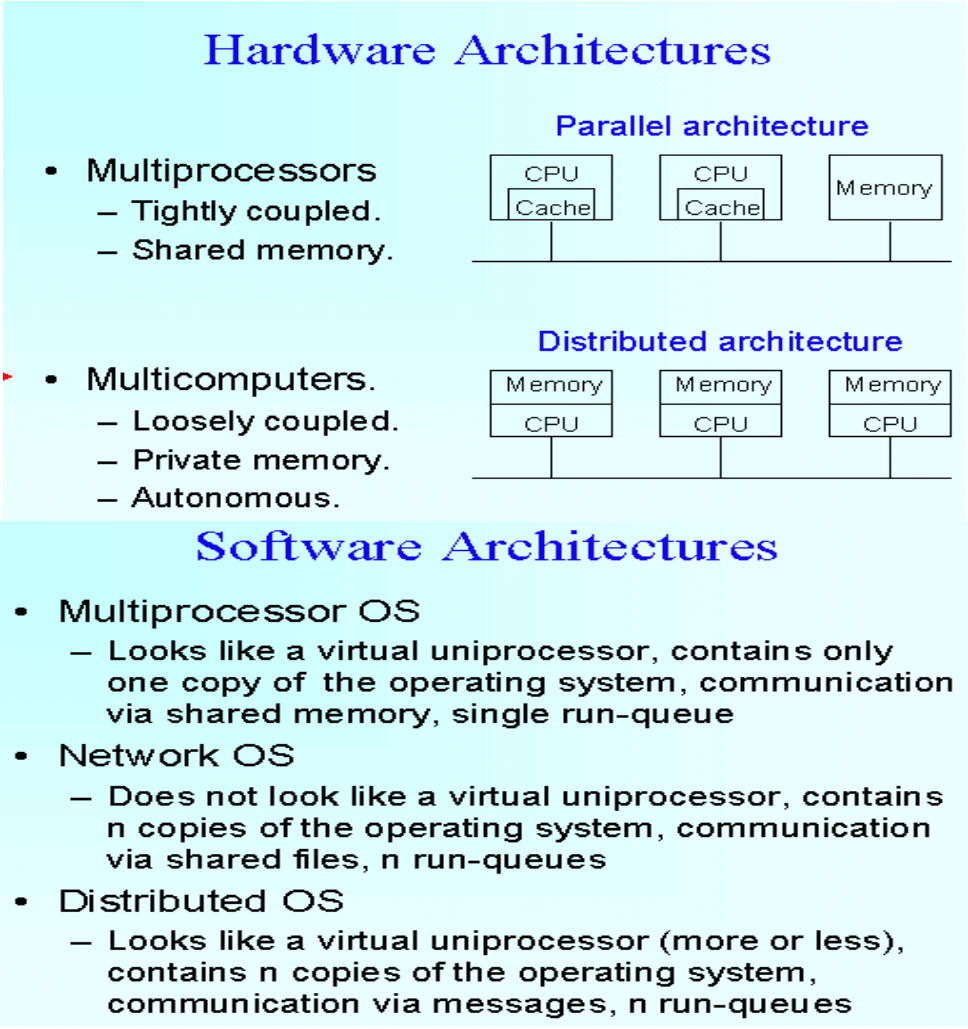
- **Networked systems** consist of multiple computers that are networked together, usually with a common operating system and shared resources. Users, however, are aware of the different computers that make up the system.

- **Distributed systems** also consist of multiple computers but differ from networked systems in that the multiple computers are transparent to the user. Often there are redundant resources and a sharing of the workload among the different computers, but this is all transparent to the user.

**3.7 Distributed system:**

A distributed operating system is one that looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs). The key concept here is transparency.

In other words, the use of multiple processors should be invisible (transparent) to the user. Another way of expressing the same idea is to say that the user views the system as a "virtual uniprocessor," not as a collection of distinct machines.



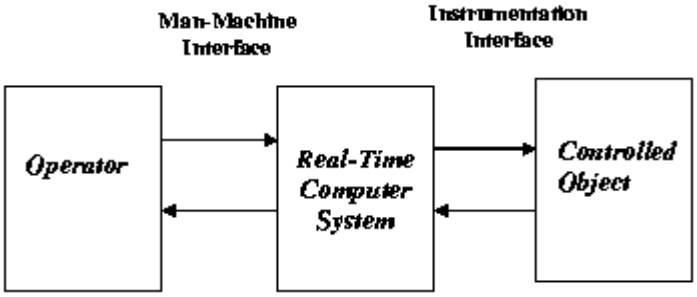

**Advantage:**

- Share resources
- Increase computation speed
- Improve functionality and data availability
- Improve reliability

**3.8 Real time system**

Real-Time systems are defense and space systems, networked multimedia systems, embedded automative electronics etc. In a real-time system the correctness of the system behavior depends not only the logical results of the computations, but also on the physical instant at which these results are produced. A real-time computer system must react to stimuli from the controlled object (or the operator) within time intervals dictated by its environment. The instant at which a

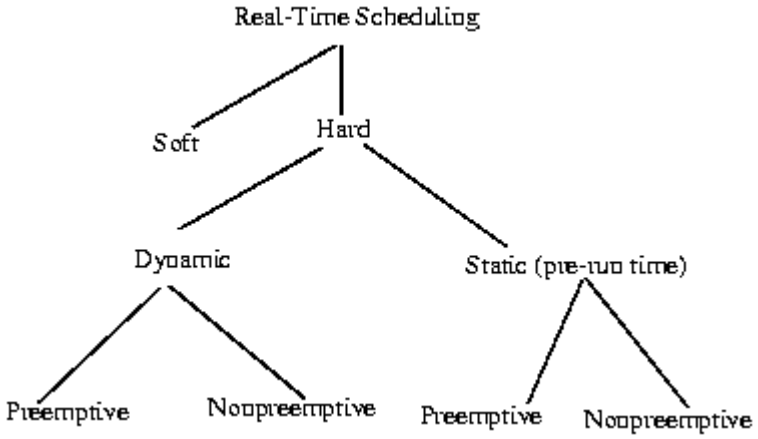
result is produced is called a deadline. If the result has utility even after the deadline has passed, the deadline is classified as soft, otherwise it is firm. If a catastrophe could result if a firm deadline is missed, the deadline is hard. Commands and Control systems, Air traffic control systems are examples for hard real-time systems. On-line transaction systems, airline reservation systems are soft real-time systems.




**Real time system**

**Real-Time Scheduling**

A hard real-time system must execute a set of concurrent real-time tasks in a such a way that all time-critical tasks meet their specified deadlines. Every task needs computational and data resources to complete the job. The scheduling problem is concerned with the allocation of the resources to satisfy the timing constraints. Figure 2 given below represents a taxonomy of real-time scheduling algorithms.




**3.9 Parallel system:**

Parallel operating systems are used to interface multiple networked computers to complete tasks in parallel. The architecture of the software is often a UNIX-based platform, which allows it to coordinate distributed loads between multiple computers in a network. Parallel operating systems are able to use software to manage all of the different resources of the computers running in parallel, such as memory, caches, storage space, and processing power. Parallel operating systems also allow a user to directly interface with all of the computers in the network.

A parallel operating system works by dividing sets of calculations into smaller parts and distributing them between the machines on a network. To facilitate communication between the processor cores and memory arrays, routing software has to either share its memory by assigning the same address space to all of the networked computers, or distribute its memory by assigning a different address space to each processing core. Sharing memory allows the operating system to run very quickly, but it is usually not as powerful. When using distributed shared memory, processors have access to both their own local memory and the memory of other processors; this distribution may slow the operating system, but it is often more flexible and efficient.

Most fields of science, including biotechnology, cosmology, theoretical physics, astrophysics, and computer science, use parallel operating systems to utilize the power of parallel computing. These types of system set-ups also help create efficiency in such industries as consulting, finance, defense, telecom and weather forecasting. In fact, parallel computing has become so robust that it has been used by cosmologists to answer questions about the origin of the universe. Scientists, researches, and industries often choose to use parallel operating systems because of its cost effectiveness as well. It costs far less money to assemble a parallel computer network than it costs to develop and build a super computer for research. Parallel systems are also completely modular, allowing inexpensive repairs and upgrades to be made.

### 4. Characteristics of Modern Operating Systems

**Microkernel architecture-**assigns only a few essential functions to the kernel, including address spaces, interprocess communication, and basic scheduling.

**Multithreading**- the process is divided into threads that can run simultaneously

**Thread**- dispatch able unit of work. It includes a processor context program counter and stack pointer and its own data storage for the stack. It executes sequentially and is interruptable

**Process**- collection of one or more threads and associated system resources.

**Symmetric multiprocessing**- standalone computer with multiple processors that share the same memory and I/O facilities connected by a communication bus. All processors can perform the same functions

**Distributed operating systems -** provide the illusion of a single main memory and single secondary memory space. Used for distributed file system

**Object-oriented design -** used for adding modular extensions to a small kernel.
Enables programmers to customize an operating system without disrupting system integrity