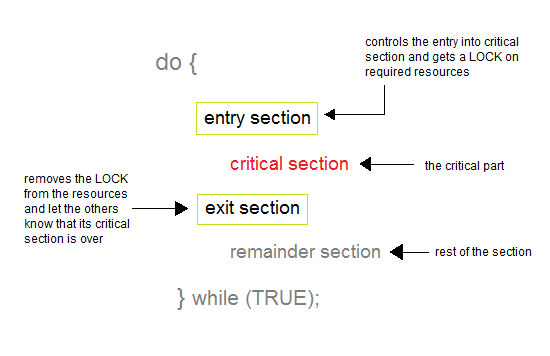
**Process Synchronization**

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

**Critical Section Problem**

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



**Solution to Critical Section Problem**

A solution to the critical section problem must satisfy the following three conditions :

1. **Mutual Exclusion**

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

1. **Progress**

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

1. **Bounded Waiting**

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

**Synchronization Hardware**

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

**Mutex Locks**

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

**Semaphores**

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called **semaphore**. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of wait and signal are :

* Wait : decrement the value of its argument S as soon as it would become non-negative.
* Signal : increment the value of its argument, S as an individual operation.

**Properties of Semaphores**

1. Simple
2. Works with many processes
3. Can have many different critical sections with different semaphores
4. Each critical section has unique access semaphores
5. Can permit multiple processes into the critical section at once, if desirable.

**Types of Semaphores**

Semaphores are mainly of two types:

1. **Binary Semaphore**

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called *Mutex*. A binary semaphore is initialized to 1 and only takes the value 0 and 1 during execution of a program.

1. **Counting Semaphores**

These are used to implement bounded concurrency.

**Limitations of Semaphores**

1. Priority Inversion is a big limitation of semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called Deadlock. We will be studying deadlocks in details in coming lessons.

**Classical Problem of Synchronization**

Following are some of the classical problem faced while process synchronaization in systems where cooperating processes are present.

**Bounded Buffer Problem**

* This problem is generalised in terms of the Producer-Consumer problem.
* Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

**The Readers Writers Problem**

* In this problem there are some processes(called readers) that only read the shared data, and never change it, and there are other processes(called writers) who may change the data in addition to reading or instead of reading it.
* There are various type of the readers-writers problem, most centred on relative priorities of readers and writers

**Dining Philosophers Problem**

* The dining philosopher's problem involves the allocation of limited resources from a group of processes in a deadlock-free and starvation-free manner.
* There are five philosophers sitting around a table, in which there are five chopsticks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

### Types of Operating Systems

Following are some of the most widely used types of Operating system.

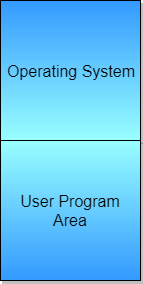
1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Realtime Operating System
8. Handheld System

#### SIMPLE BATCH SYSTEMS

* In this type of system, there is no direct interaction between user and the computer.
* The user has to submit a job (written on cards or tape) to a computer operator.
* Then computer operator places a batch of several jobs on an input device.
* Jobs are batched together by type of languages and requirement.
* Then a special program, the monitor, manages the execution of each program in the batch.
* The monitor is always in the main memory and available for execution.

Following are some disadvantages of this type of system :

1. No interaction between user and computer.
2. No mechanism to prioritise the processes.

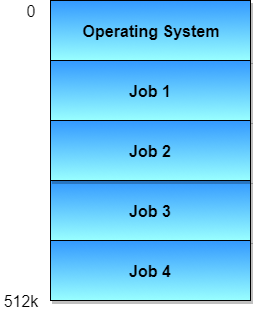


#### MULTIPROGRAMMING BATCH SYSTEMS

* In this the operating system picks up and begins to execute one of the jobs from memory.
* Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
* Jobs in the memory are always less than the number of jobs on disk(Job Pool).
* If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
* In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
* In Multiprogramming system, CPU will never be idle and keeps on processing.

**Time-Sharing Systems** are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.



#### MULTIPROCESSOR SYSTEMS

A multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

Following are some advantages of this type of system.

1. Enhanced performance
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

#### DESKTOP SYSTEMS

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither **multiuser** nor **multitasking**. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called **Desktop Systems** and include PCs running Microsoft Windows and the Apple Macintosh. Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.

**Microcomputers** were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

#### DISTRIBUTED OPERATING SYSTEMS

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

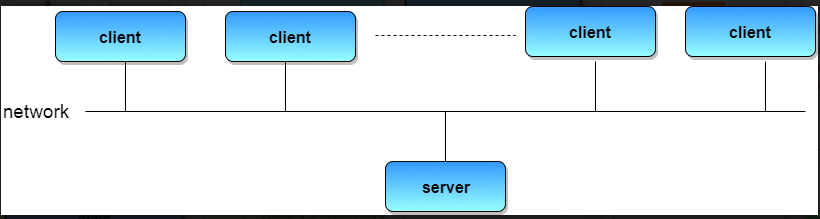
Following are some advantages of this type of system.

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

The two types of Distributed Operating Systems are: Client-Server Systems and Peer-to-Peer Systems.

##### Client-Server Systems

**Centralized systems** today act as **server systems** to satisfy requests generated by **client systems**. The general structure of a client-server system is depicted in the figure below:



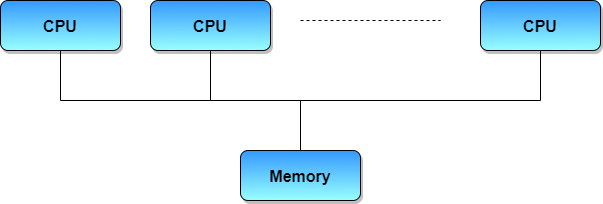
Server Systems can be broadly categorized as **compute servers** and **file servers**.

* **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
* **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

##### Peer-to-Peer Systems

The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems. When PCs were introduced in the 1970s, they were designed for **personal** use and were generally considered standalone computers. With the beginning of widespread public use of the Internet in the 1980s for electronic mail and ftp many PCs became connected to computer networks.

In contrast to the **tightly coupled** systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled systems ( or distributed systems). The general structure of a client-server system is depicted in the figure below:



#### CLUSTERED SYSTEMS

* Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
* Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
* The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
* Clustering is usually performed to provide **high availability**.
* A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.
* **Asymmetric Clustering -** In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.
* **Symmetric Clustering -** In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.
* **Parallel Clustering -** Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system use and features should expand greatly as **Storage Area Networks(SANs)**. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

#### REAL-TIME OPERATING SYSTEM

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.**

While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**.

#### HANDHELD SYSTEMS

Handheld systems include **Personal Digital Assistants(PDAs)**, such as Palm-Pilots or Cellular Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

* Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
* Currently, many handheld devices do **not use virtual memory** techniques, thus forcing program developers to work within the confines of limited physical memory.
* Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
* The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology such as **BlueTooth**, allowing remote access to e-mail and web browsing. **Cellular telephones** with connectivity to the Internet fall into this category. Their use continues to expand as network connections become more available and other options such as cameras and MP3 players, expand their utility.