**Semaphore**

Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as semaphore. Semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

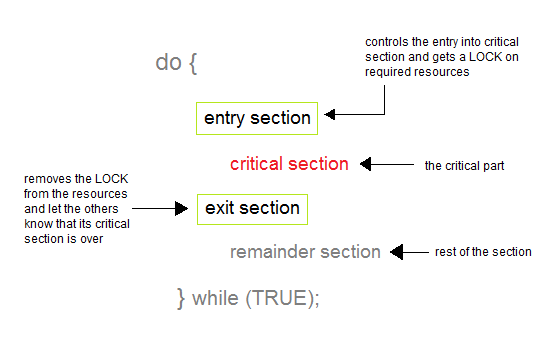
# Process Synchronization

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.

#### 2. 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.

#### 3. 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.

## Race Condition

A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

# Classical Problems of Synchronization

In this tutorial we will discuss about various classic problem of synchronization.

Semaphore can be used in other synchronization problems besides Mutual Exclusion.

Below are some of the classical problem depicting flaws of process synchronaization in systems where cooperating processes are present.

We will discuss the following three problems:

1. Bounded Buffer (Producer-Consumer) Problem
2. Dining Philosophers Problem
3. The Readers Writers Problem

## [Bounded Buffer Problem](https://www.studytonight.com/operating-system/bounded-buffer)

* This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.

Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer 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.

## [Dining Philosophers Problem](https://www.studytonight.com/operating-system/dining-philosophers-problem" \t "_blank)

* The dining philosopher's problem involves the allocation of limited resources to 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/forks 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.

## [The Readers Writers Problem](https://www.studytonight.com/operating-system/readers-writer-problem" \t "_blank)

* 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 readers-writers problem, most centred on relative priorities of readers and writers.