Process Synchronization

# Introduction

Process Synchronization was introduced to handle problems that are faced to run multiple processes by context switching.

Process is categorized into two types on the basis of synchronization and these are given below:

* Independent Process : Two processes are said to be independent if the execution of one process does not affect the execution of another process.
* Cooperative Process : Two processes are said to be cooperative if the execution of one process affects the execution of another process.

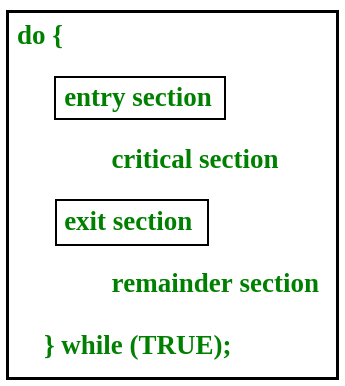
# Race Condition

**When more than one process is executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition.**

Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place. A race condition is a situation that may occur inside a critical section.

# Critical Section

A critical section is a code segment that can be accessed by only one process at a time. The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables.

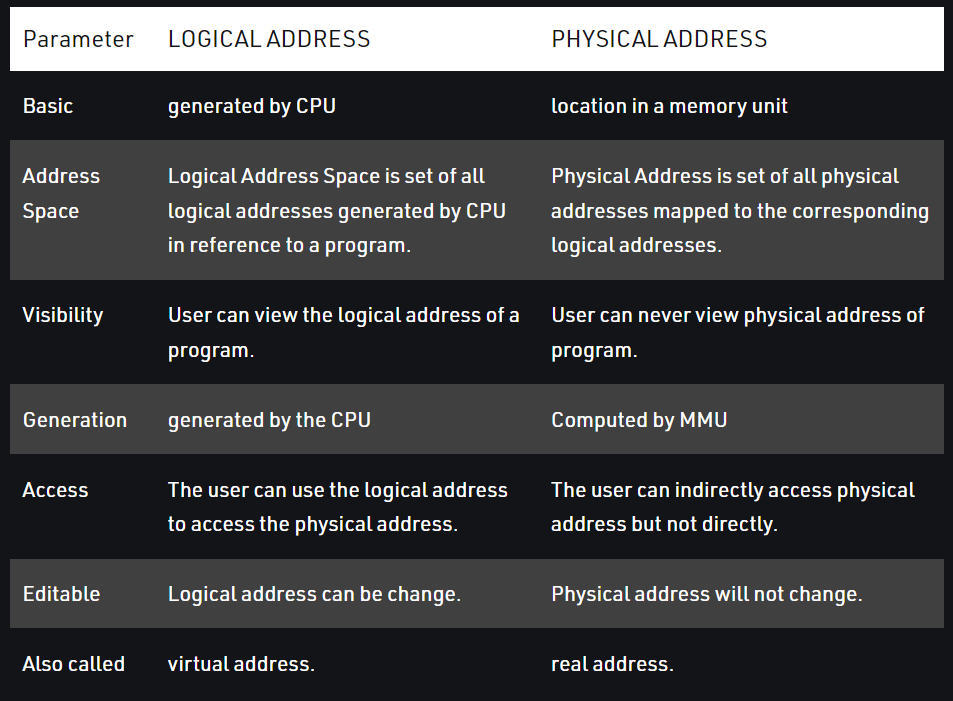


# Any solution to the critical section problem must satisfy three requirements:

* Mutual Exclusion: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
* Progress: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.
* Bounded Waiting: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

# Logical and Physical Address

* **Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
* **Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used.



# Process Synchronization using Lock Variable

This is work in the kernel mode, this is pure software base solution which doesn’t depend upon the hardware. The main two part of this algorithm is acquire the lock and release the lock.

**Entry section code:**

* while(lock == 1);
* lock = 1;

**Exit Section Code:**

* lock = 0;

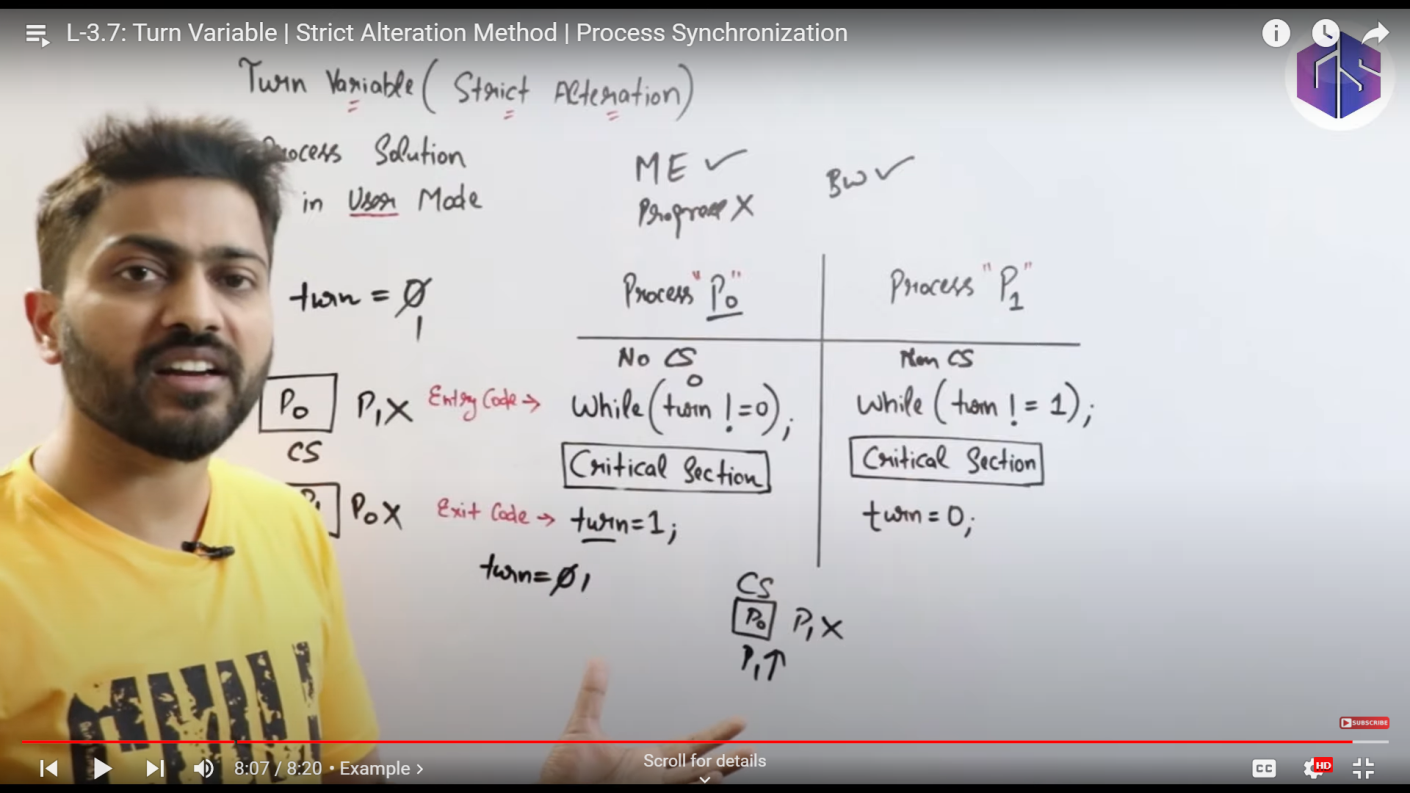
This is very easy algorithm, may be looks like it will provide ME but in the real world scenario it doesn’t work, say after while(lock == 1), content switching happen, lock value is 0, so any other process can enter into the critical section, so it doesn’t provide mutual exclusion.

# Using Set and Lock Variable

Main problem of Lock variable was context switching can happen after the 1st line of entry section, so if we can merge both of the instruction and make them atomic then we can say there is no chance of cs. Say the atomic instruction is : while(test\_and\_set(&lock)); function definition is:

* Boolean r = \*lock
* \*lock = true
* return r;

# Turn Variable / Strict Alteration



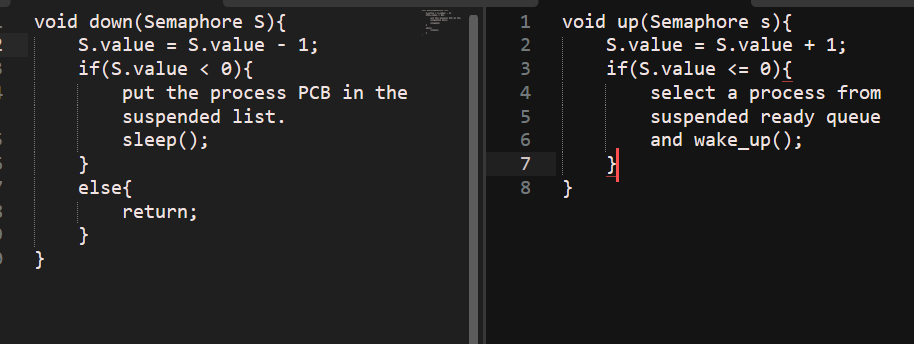
# Semaphore

Semaphore is an Integer variable which is used in mutual exclusive manner by various cooperative processes in order to achieve process synchronization.

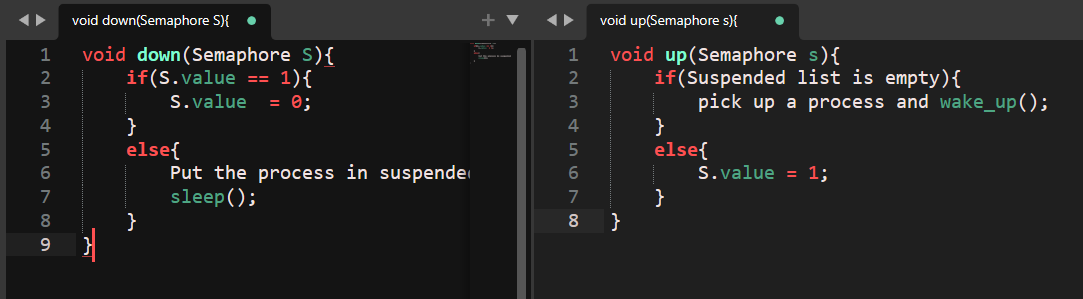
* Counting Semaphore (-inf to +inf)
* Binary Semaphore (0 & 1)

Different operations:

* P() , Down() , wait() -> Entry section
* V() , Up() , Signal() -> Exit Section
* Counting Semaphore



* If value of s is 10 : no of processes can access cs is : 10
* If value of s is -2 : no of processes in suspended list is : 2
* Binary Semaphore



# Producer Consumer Problem

