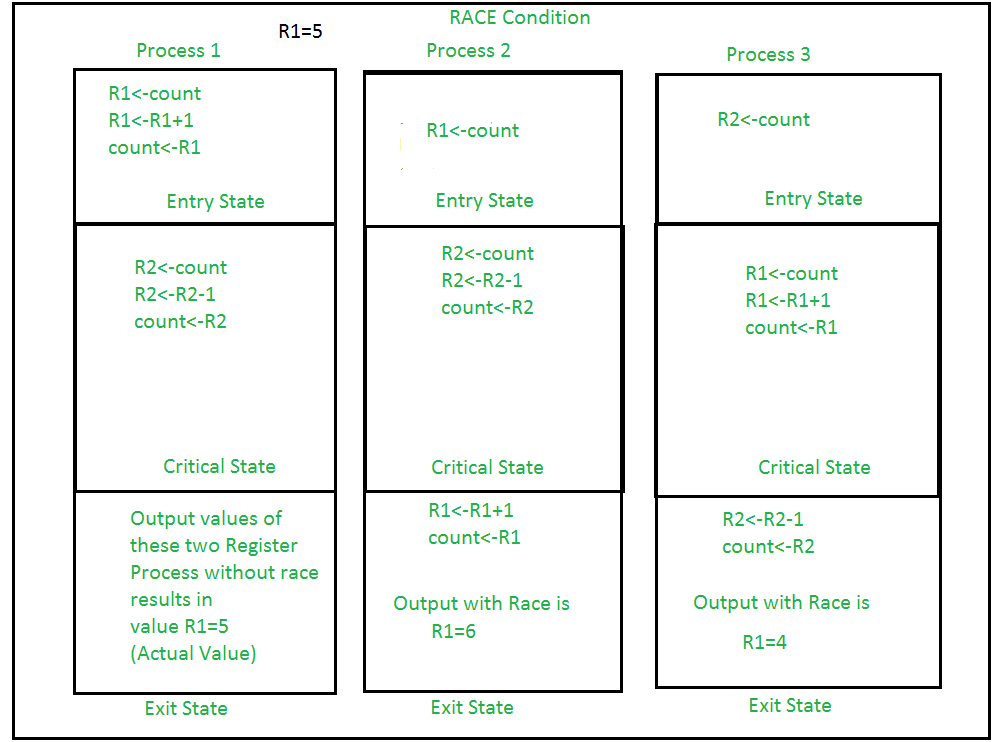
**Process Synchronization** 

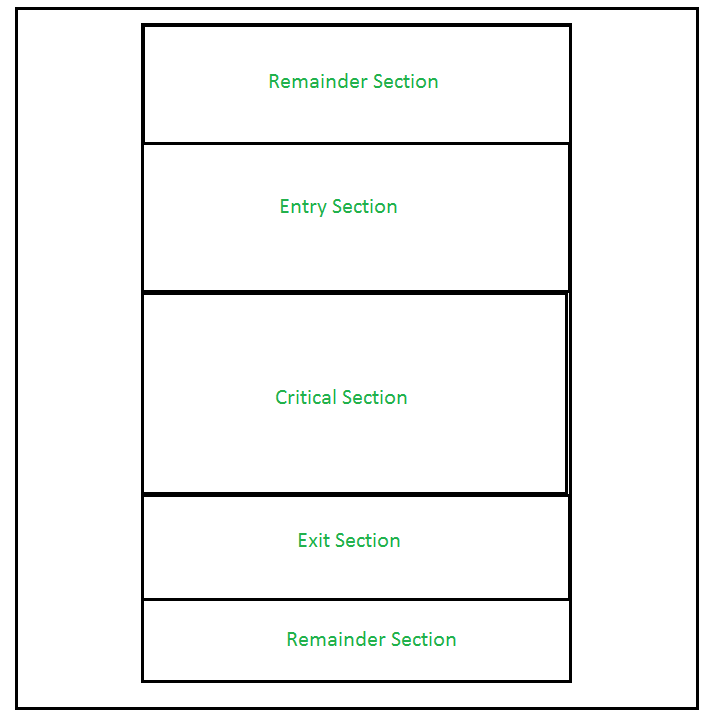
**Process Synchronization** is a technique which is used to coordinate the process that use shared Data. There are two types of Processes in an Operating Systems:-

1. **Independent Process –**  
   The process that does not affect or is affected by the other process while its execution then the process is called Independent Process. Example The process that does not share any shared variable, database, files, etc.
2. **Cooperating Process –**  
   The process that affect or is affected by the other process while execution, is called a Cooperating Process. Example The process that share file, variable, database, etc. are the Cooperating Process.

Process Synchronization is mainly used for Cooperating Process that shares the resources. Let us consider an example of **racing condition**



It is the condition where several processes tries to access the resources and modify the shared data concurrently and outcome of the process depends on the particular order of execution that leads to data inconsistency, this condition is called **Race Condition**. This condition can be avoided using the technique called Synchronization or Process Synchronization, in which we allow only one process to enter and manipulates the shared data in Critical Section.  
**Diagram of the view of CS:**



This setup can be defined in various regions like:

* **Entry Section –**  
  It is part of the process which decide the entry of a particular process in the Critical Section, out of many other processes.
* **Critical Section –**  
  It is the part in which only one process is allowed to enter and modify the shared variable. This part of the process ensures that only no other process can access the resource of shared data.
* **Exit Section –**  
  This process allows the other process that are waiting in the Entry Section, to enter into the Critical Sections. It checks that a process that after a process has finished execution in Critical Section can be removed through this Exit Section.
* **Remainder Section –**  
  The other parts of the Code other than Entry Section, Critical Section and Exit Section are known as Remainder Section.

Critical Section problems must satisfy these three requirements:

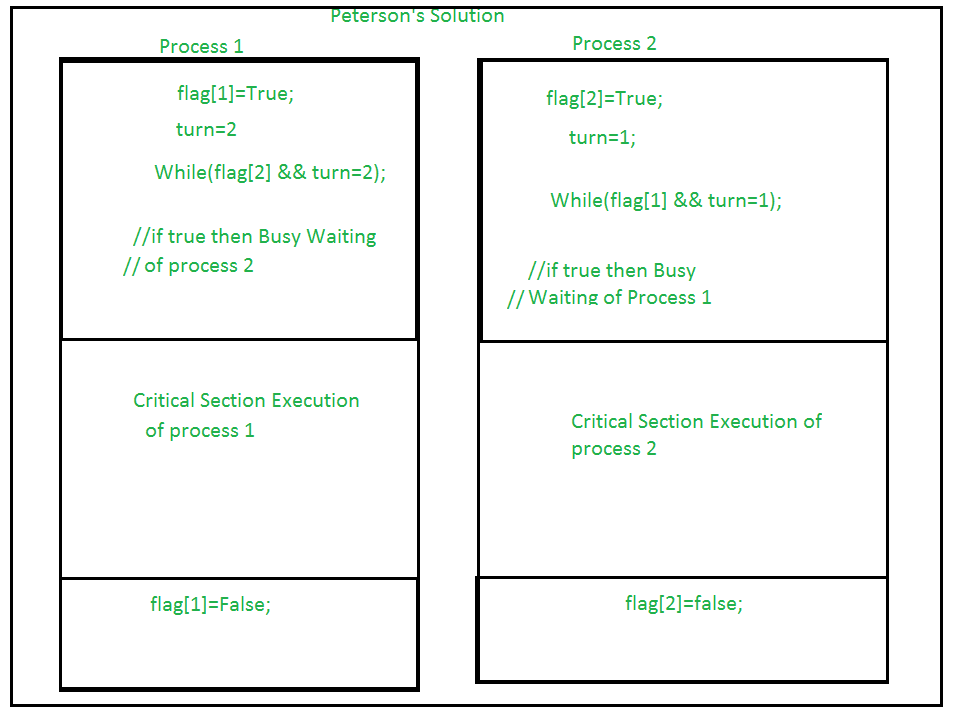
1. **Mutual Exclusion –**  
   It states that no other process is allowed to execute in the critical section if a process is executing in critical section.
2. **Progress –**  
   When no process is in the critical section, then any process from outside that request for execution can enter in the critical section without any delay. Only those process can enter that have requested and have finite time to enter the process.
3. **Bounded Waiting –**  
   An upper bound must exist on the number of times a process enters so 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.

**Process Synchronization are handled by two approaches:**

1. **Software Approach –**  
   In Software Approach, Some specific Algorithm approach is used to maintain synchronization of the data. Like in Approach One or Approach Two, for a number of two process, a temporary variable like (turn) or Boolean variable (flag) value is used to store the data. When the condition is true then the process in waiting State, known as Busy Waiting State. This does not satisfy all the Critical Section requirements.

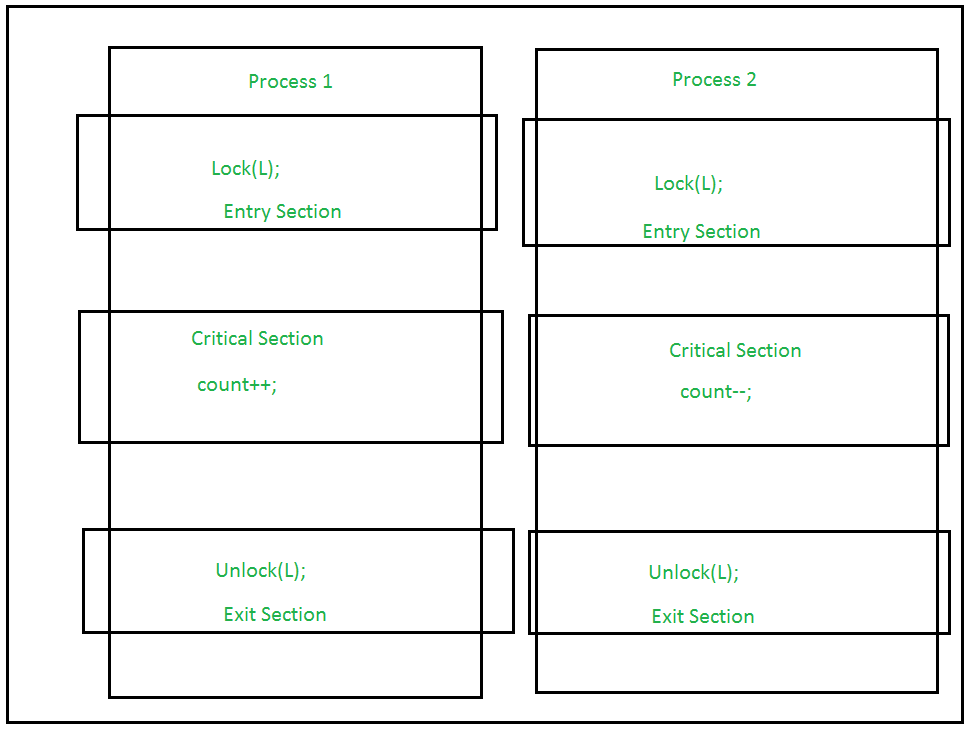
Another Software approach known as Peterson’s Solution is best for Synchronization. It uses two variables in the Entry Section so as to maintain consistency, like Flag (Boolean variable) and Turn variable (storing the process states). It satisfy all the three Critical Section requirements.

**Peterson’s Algorithm:**



1. **Hardware Approach –**  
   The Hardware Approach of synchronization can be done through Lock & Unlock technique. Locking part is done in the Entry Section, so that only one process is allowed to enter into the Critical Section, after it complete its execution, the process is moved to the Exit Section, where Unlock Operation is done so that another process in the Lock Section can repeat this process of Execution. This process is designed in such a way that all the three conditions of the Critical Sections are satisfied.

**Lock:**



**Using Interrupts –**  
These are easy to implement. When Interrupt are disabled then no other process is allowed to perform Context Switch operation that would allow only one process to enter into the Critical State.

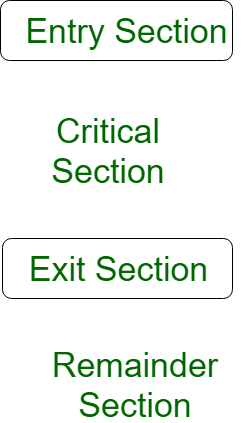
**Critical Section in Synchronization**

[**Critical Section**](http://en.wikipedia.org/wiki/Critical_section)**:**   
When more than one processes access a same code segment that segment is known as critical section. Critical section contains shared variables or resources which are needed to be synchronized to maintain consistency of data variable.  
In simple terms a critical section is group of instructions/statements or region of code that need to be executed atomically ([read this post](https://www.geeksforgeeks.org/g-fact-57/) for atomicity), such as accessing a resource (file, input or output port, global data, etc.).

In concurrent programming, if one thread tries to change the value of shared data at the same time as another thread tries to read the value (i.e. data race across threads), the result is unpredictable.

The access to such shared variable (shared memory, shared files, shared port, etc…) to be synchronized. Few programming languages have built-in support for synchronization.

It is critical to understand the importance of race condition while writing kernel mode programming (a device driver, kernel thread, etc.). Since the programmer can directly access and modifying kernel data structures.



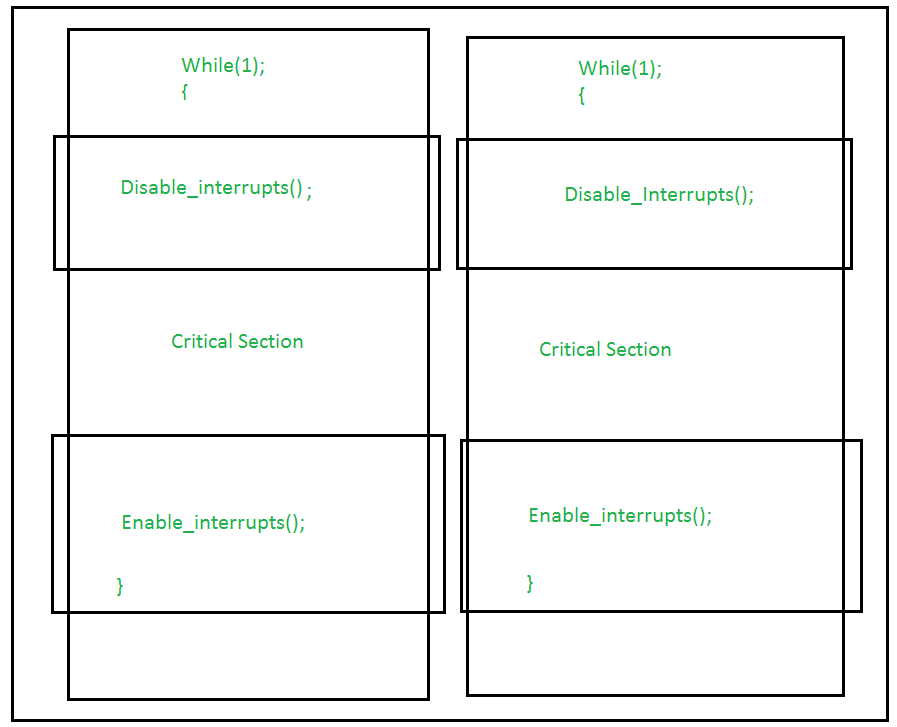
A simple solution to the critical section can be thought as shown below,

acquireLock();

Process Critical Section

releaseLock();

A thread must acquire a lock prior to executing a critical section. The lock can be acquired by only one thread. There are various ways to implement locks in the above pseudo code.



**Test\_and\_Set Operation –**  
This allows Boolean value (True/False) as a hardware Synchronization, which is atomic in nature i.e no other interrupt is allowed to access. This is mainly used in Mutual Exclusion Application. Similar type operation can be achieved through Compare and Swap function. In this process, a variable is allowed to access in Critical Section while its lock operation is ON. Till then, the other process is in Busy Waiting State. Hence Critical Section Requirements are achieved.