Process Synchronization-

When multiple processes execute concurrently sharing system resources, then inconsistent results might be produced.

- Process Synchronization is a mechanism that deals with the synchronization of processes.
- It controls the execution of processes running concurrently to ensure that consistent results are produced.

Need of Synchronization-

Process synchronization is needed-

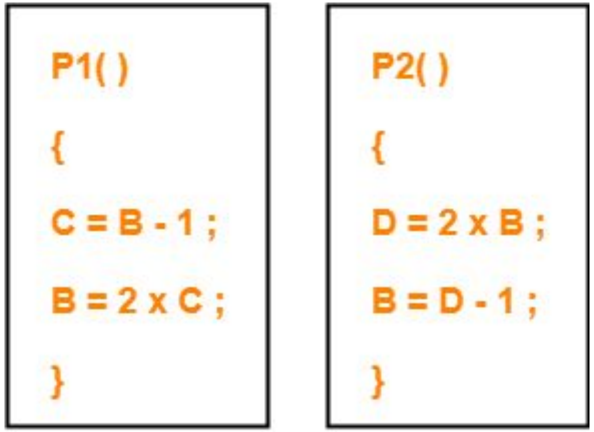
- When multiple processes execute concurrently sharing some system resources.
- To avoid the inconsistent results.

Critical Section-

Critical section is a section of the program where a process access the shared resources during its execution.

Problem-

The following two functions P1 and P2 that share a variable B with an initial value of 2 execute concurrently-



```
P1( )
{
C = B - 1 ;
B = 2 x C ;
}
```

```
P2( )
{
D = 2 x B ;
B = D - 1 ;
}
```

The number of distinct values that B can possibly take after the execution is

<u>Critical Section Problem-</u>

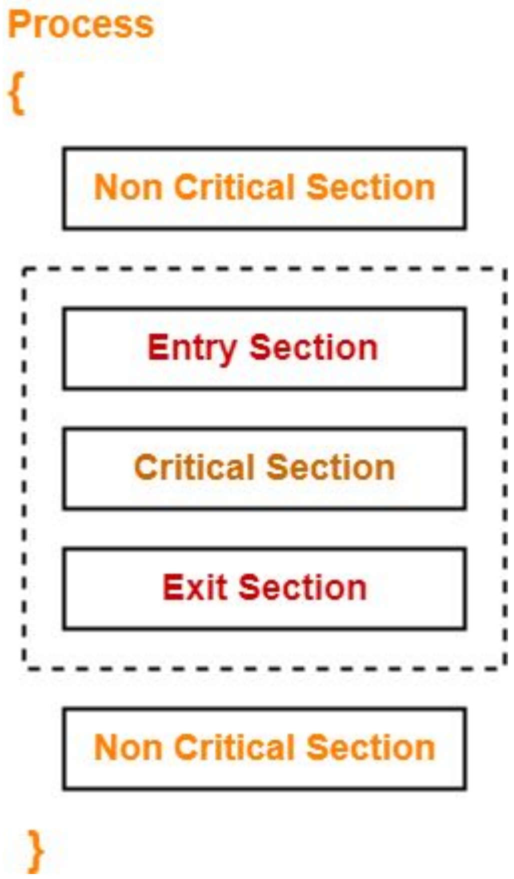
- If multiple processes access the critical section concurrently, then results produced might be inconsistent.
- This problem is called as **critical section problem**.

Synchronization Mechanisms-

Synchronization mechanisms allow the processes to access critical section in a synchronized manner to avoid the inconsistent results.

For every critical section in the program, a synchronization mechanism

- An entry section before the critical section
- An exit section after the critical section


Process
{
Non Critical Section
Entry Section
Critical Section
Exit Section
Non Critical Section
}

Entry Section-

- It acts as a gateway for a process to enter inside the critical section.
- It ensures that only one process is present inside the critical section at any time.
- It does not allow any other process to enter inside the critical section if one process is already present inside it.

Exit Section-

- It acts as an exit gate for a process to leave the critical section.
- When a process takes exit from the critical section, some changes are made so that other processes can enter inside the critical section.

Criteria For Synchronization Mechanisms-

Any synchronization mechanism proposed to handle the critical section problem should meet the following criteria-

1. Mutual Exclusion
2. Progress
3. Bounded Wait

1. Mutual Exclusion-

The mechanism must ensure-

- The processes access the critical section in a mutual exclusive manner.
- Only one process is present inside the critical section at any time.
- No other process can enter the critical section until the process already present inside it completes.

2. Progress-

The mechanism must ensure-

- An entry of a process inside the critical section is not dependent on the entry of another process inside the critical section.
- A process can freely enter inside the critical section if there is no other process present inside it.
- A process enters the critical section only if it wants to enter.
- A process is not forced to enter inside the critical section if it does not want to enter.

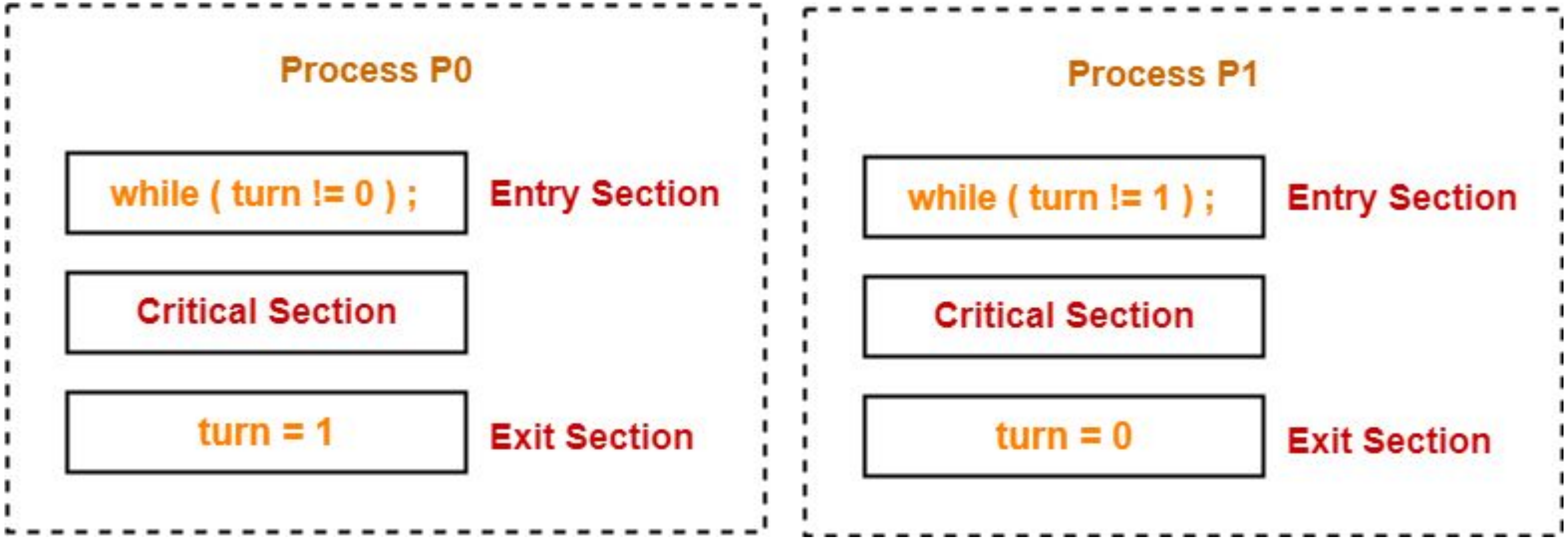
3. Bounded Wait-

The mechanism should ensure-

- The wait of a process to enter the critical section is bounded.
- A process gets to enter the critical section before its wait gets over.

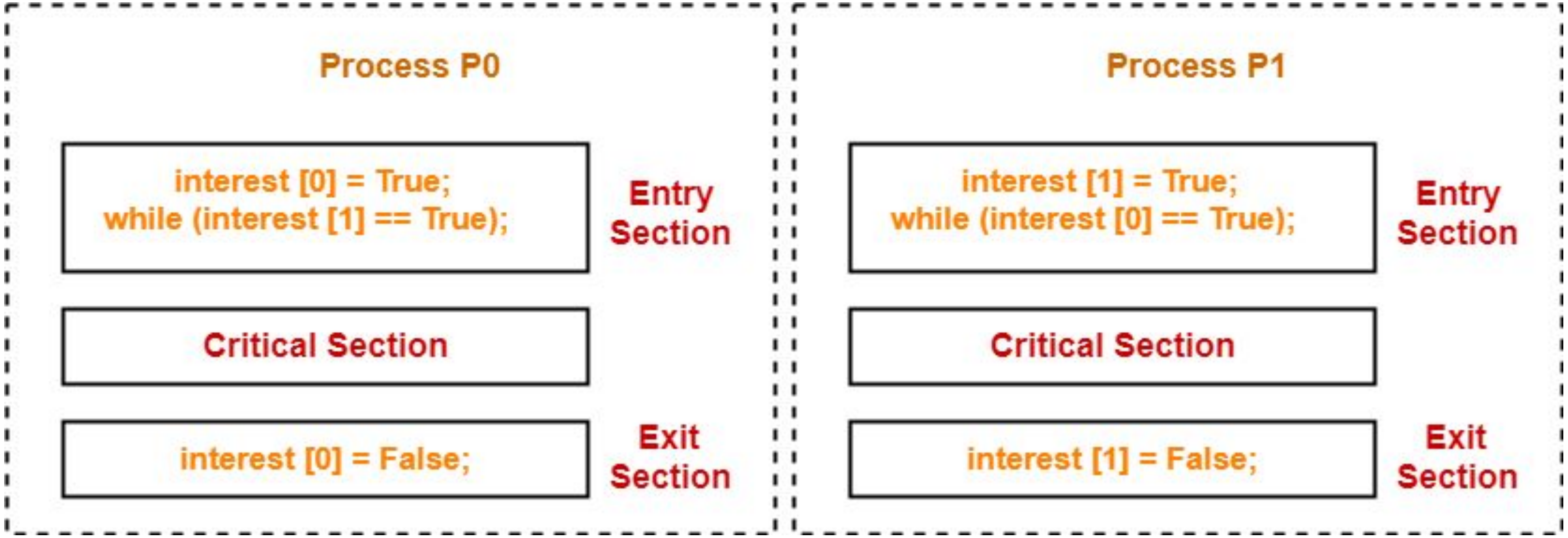
Turn Variable-

Initially, turn value is set to 0.



Interest Variable-

- Interest variable is a synchronization mechanism that provides synchronization among two processes.
- It uses an interest variable to provide the synchronization.



Characteristics-

The characteristics of this synchronization mechanism are-

- It ensures mutual exclusion.
- It suffers from deadlock

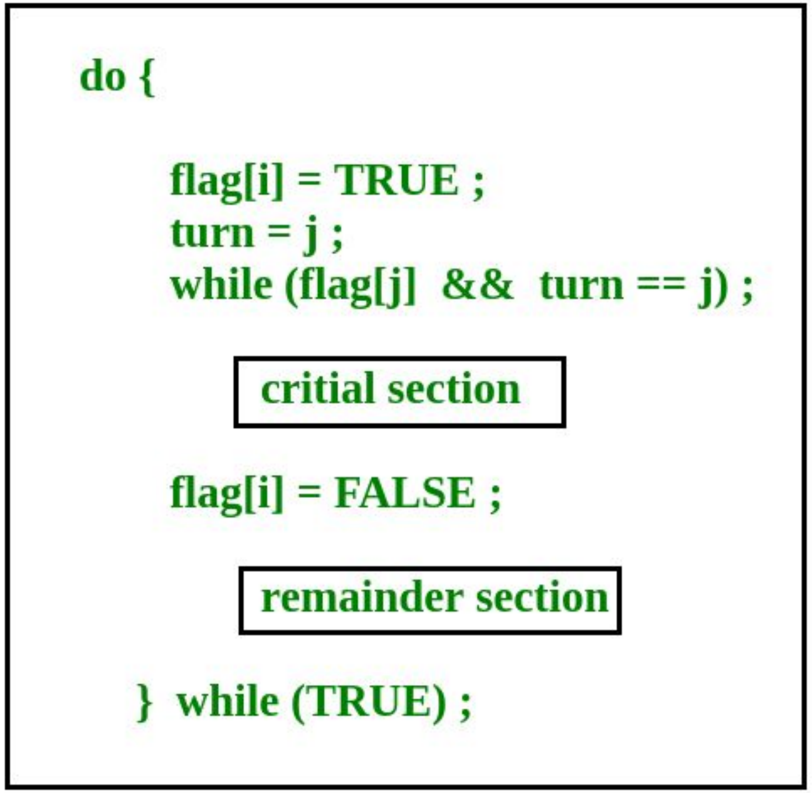
Peterson’s Solution

Peterson’s Solution is a classical software based solution to the critical section problem.

In Peterson’s solution, we have two shared variables:

- boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section
- int turn : The process whose turn is to enter the critical section.

Peterson's Solution



```
do {

        flag[i] = TRUE ;
        turn = j ;
        while (flag[j]  &&  turn == j) ;

            critial section

        flag[i] = FALSE ;

            remainder section

    }  while (TRUE) ;
```

Question

Consider the methods used by processes P_1 and P_2 for accessing their critical sections whenever needed, as given below. The initial values of shared Boolean variables S_1 and S_2 are randomly assigned

Which one of the following statements describes the properties achieved?

1. Mutual exclusion but not progress
2. Progress but not mutual exclusion
3. Neither mutual exclusion nor progress
4. Both mutual exclusion and progress

Method Used by P1

```
while (S1 == S2) ;

Critical Section

S1 = S2;
```

Method Used by P2

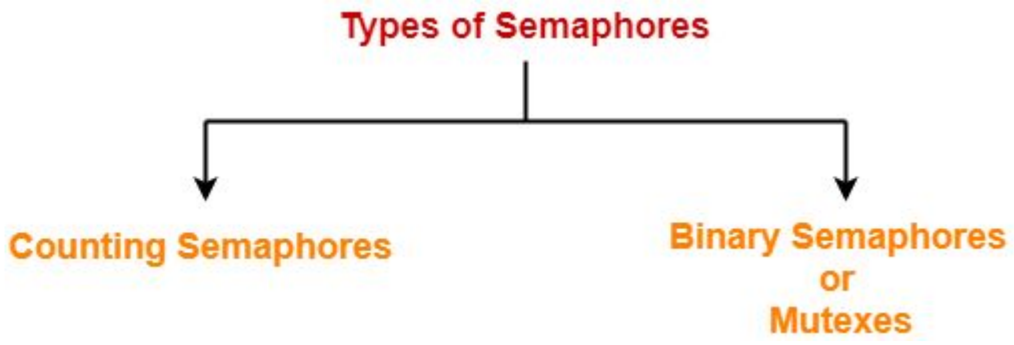
```
while (S1 != S2) ;

Critical Section

 S2 = not (S1);
```

Semaphores in OS-

- A semaphore is a simple integer variable.
- It is used to provide synchronization among multiple processes running concurrently.



Question

A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e. wait) on a counting semaphore S and invokes the V operation (i.e. signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution?

Binary semaphores

Binary semaphores are mainly used for two purposes-

- To ensure mutual exclusion.
- To implement the order in which the processes must execute.

Wait (S1) Process P1 Signal (S2)	Process P2 Signal (S1)	Wait (S2) Process P3

Two binary semaphores S1 and S2 both initialized with 0 are used.

- The execution order of the processes is P2 → P1 → P3.