Unit 2 :-

- Process synchronisation (Problems case)
- CPU scheduling (Mathematical)
- Dead Locks (CPU kept idle, its time waste)

a) condns b) Resource allocation graph, deadlock preven. method

c) avoidance, detection & recovery.

- **Process synchroniz^n:**

Process of coordinating two processes, no two process should try to access same data or resource

- **Types of solution:-** —— MUTEX Locks

Software sol^n

- Peterson

Hardware solutions

- semaphore - signal to avoid

- **Software soln :**

Peterson's soln : How to solve critical section problem.

→ Two variables are used here Turn and flag.

*int turn

Turn → whose turn to enter critical section

*bool Flag **bool Flag** → to indicate if process is ready to enter critical section

Algo:-

```
do{
    flag[i] = true;    // ith process is ready to enter critical
                                                              sec^n.
    turn = j;
    while (flag[j] && turn == j);
                    critical section;
    flag[i] = false;
                remainder section
} while
```

Peterson's soln: Flag[i] = true;

i ready to enter, but Peterson gives jth process to enter which is mutual exclusion.

In this algo, i can be Producer, j can be consumer.

Initially flags are false.
When a process wants to enter CS. It sets flag to true and turn as index of other process. This means that process want to execute but it will allow other process to run first.

~~Th~~
The

- It ensures bounded waiting
- Progress
- Mutual Exclusion.

Limitation

- busy waiting

Hardware soln :

many ~~p~~ systems provide hardware support for implementing the CS code.

→ All solns are based on (LOCKING)
↓
protecting critical regions via locks

→ Uniprocessors – which are able to disable interrupts.

- currently running code would execute~~ion~~ without preemption.

~~Gtea~~

- Inefficient on multiprocessor systems.

- Modern machine provide special atomic hardware instrc^n.

Lock

HW:

(i) Test & set Lock.

(ii) compare & swap.