CRITICAL SECTION(CS)

There are two type of processes:

⇒ Co-operative : execution of one process affects the execution of other process. these process share some resources.

⇒ Independent: execution of one process doesn't affect another.

race around condition : if two process access the same part or shared resources at same time.

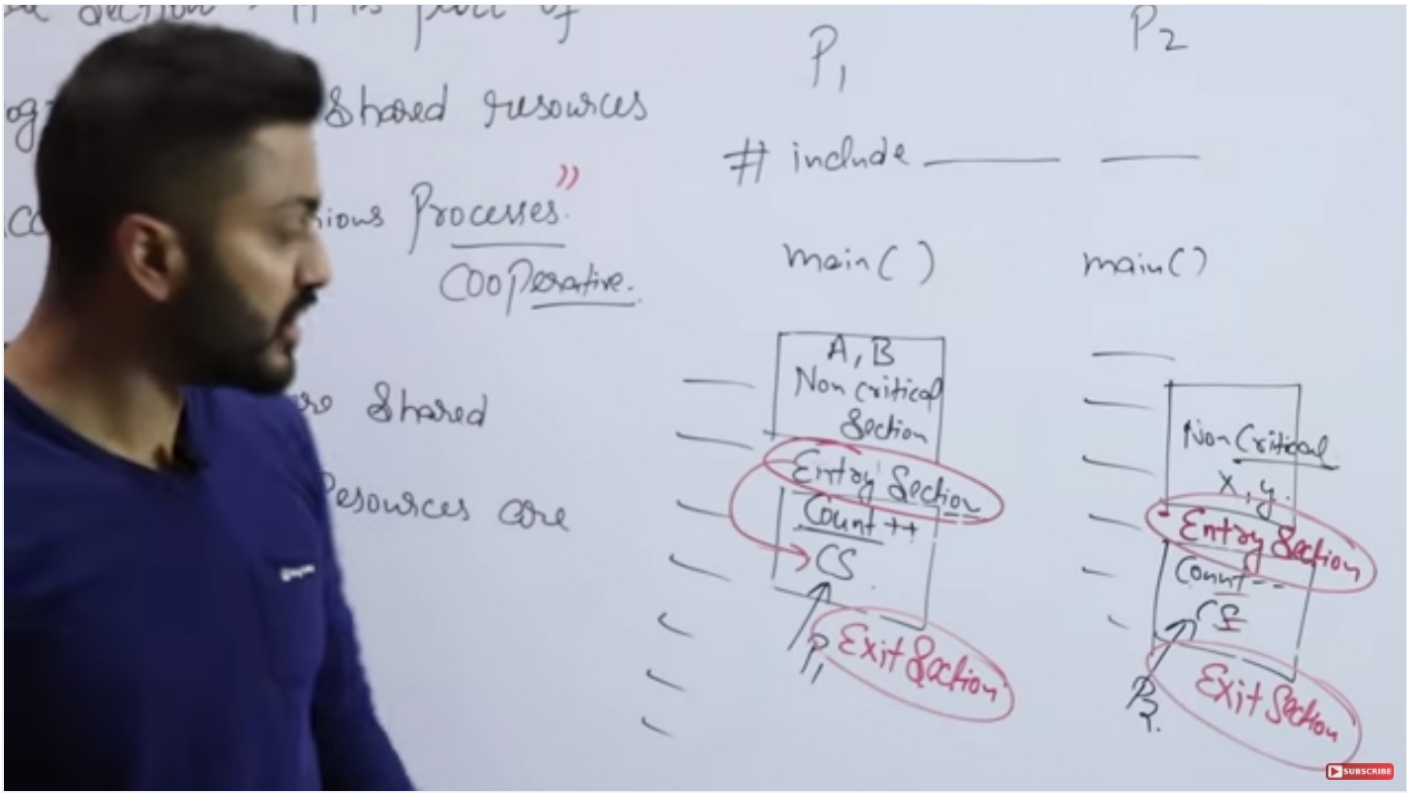
→condition arises when the process is not done in the proper sequence.

if two processes are accessing the same part and the output depend on the order of exectuion, the condition is called race around condition.

to avoid the condition we have to avoid it.

critical section:

It is the part of the program where shared resources are accessed by various processes (Co-operative processes)



we have to execute some code present in the entry section before entering the critical section and we have to execute some code before exiting the critical section.

To achieve synchronization there can be several methods/solution but these 4 conditions should be satisfied:

1. Mutual exclusion
2. progress
3. bounded wait
4. no assumption related to hardware speed

→1 and 2 are primary rules, 3 and 4 are seondary rules.

MUTUAL EXCLUSION :

If there is a process in the critical section then no other process can enter the critical section.

PROGRESS :

If a process want to enter in critical state but other process is stopping it (there might be some code of process 2) which is blocking the entrance of process 1 in CS, then there is no progress

BOUNDARY WAIT :

There should be a certain number of times a process can repeatedly access the critical state.

it should not be like process1 has use the CS infinite time whereas process2 has use the CS only once it should not be the case.

after a certain number of time process2 should also use the CS.

NO ASSUMPTION RELATED TO HARDWARE AND SPEED :

the solution should not be limitedd by the hardware and the speed of system.

like : your solution can run only on LINUX or it can only work on 64-bit systems.

critical section solutions :-

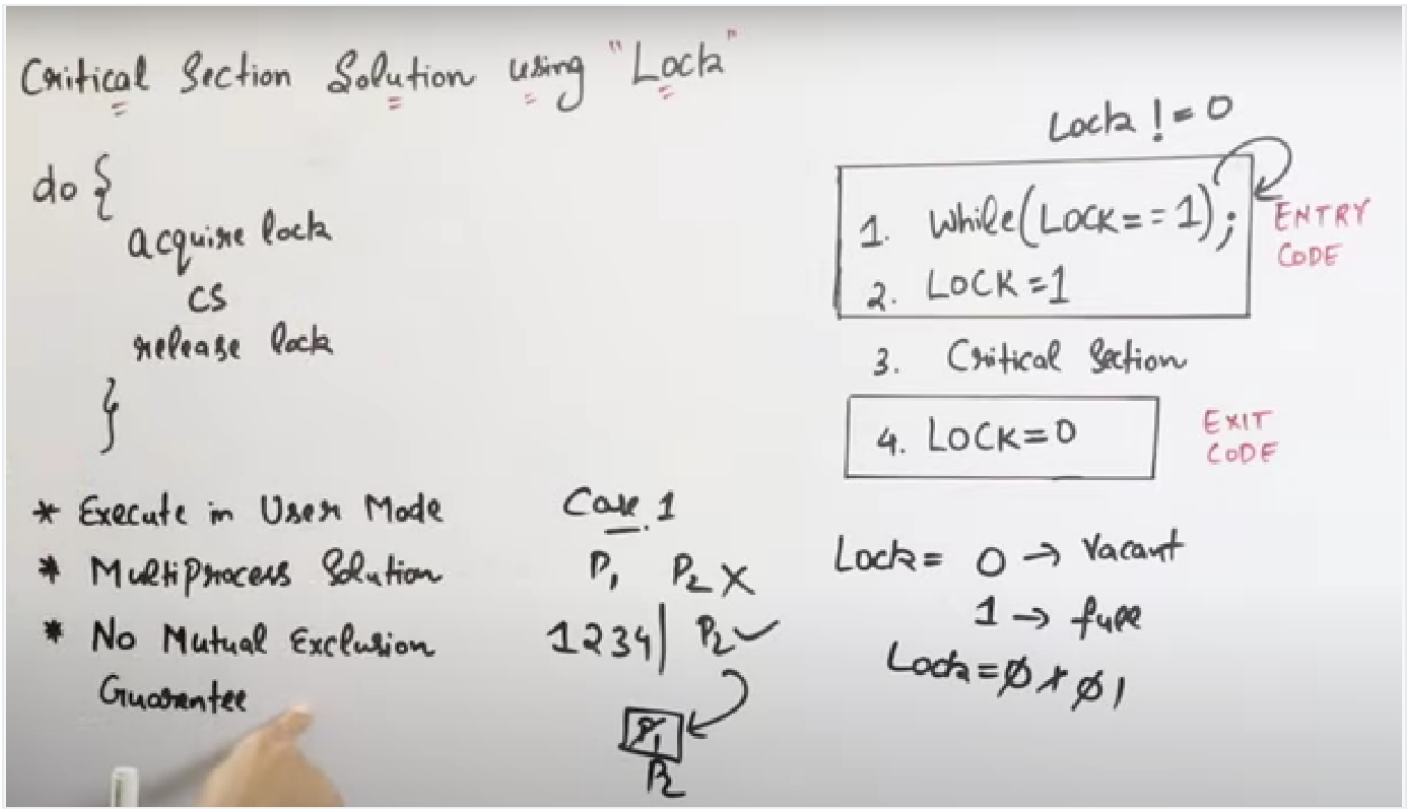
1. LOCK variable :

→ acquire lock before going to CS

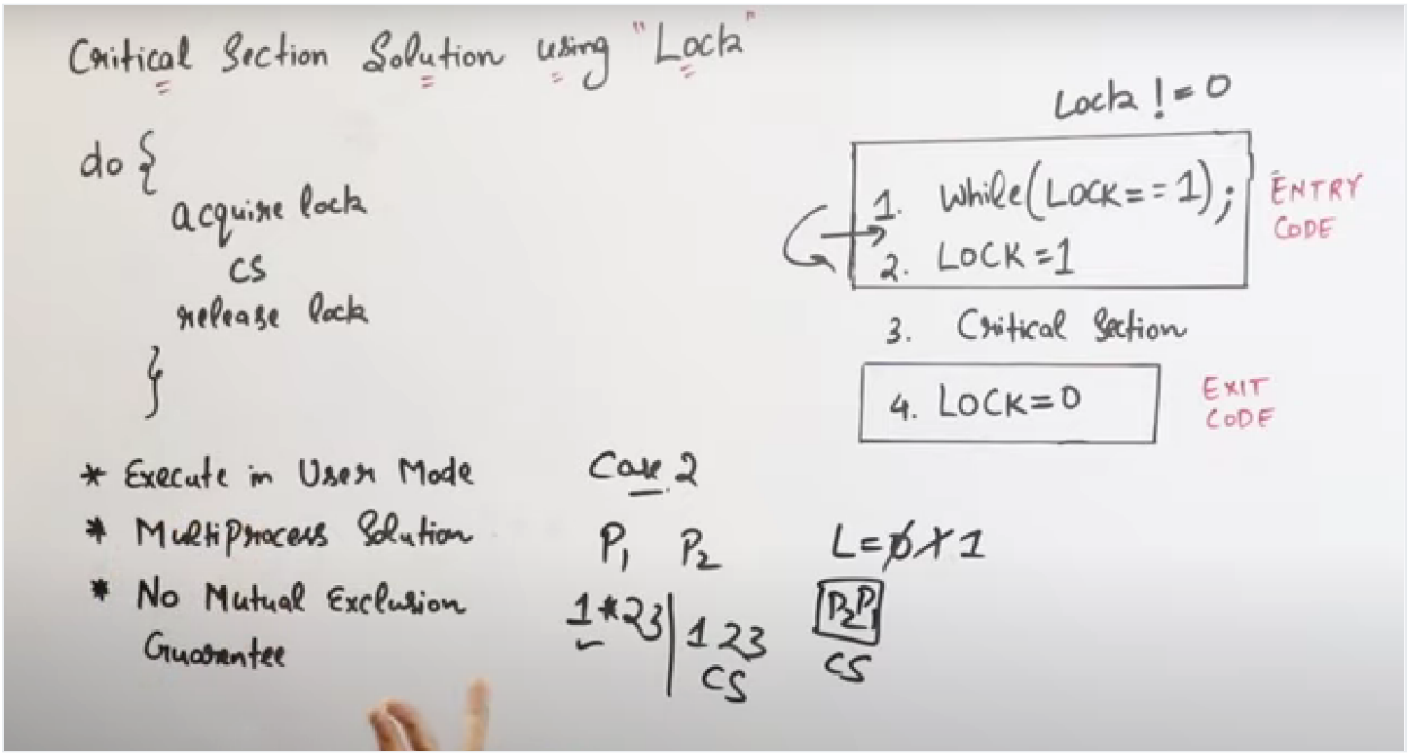
→ access the CS

→ release the lock

case 1

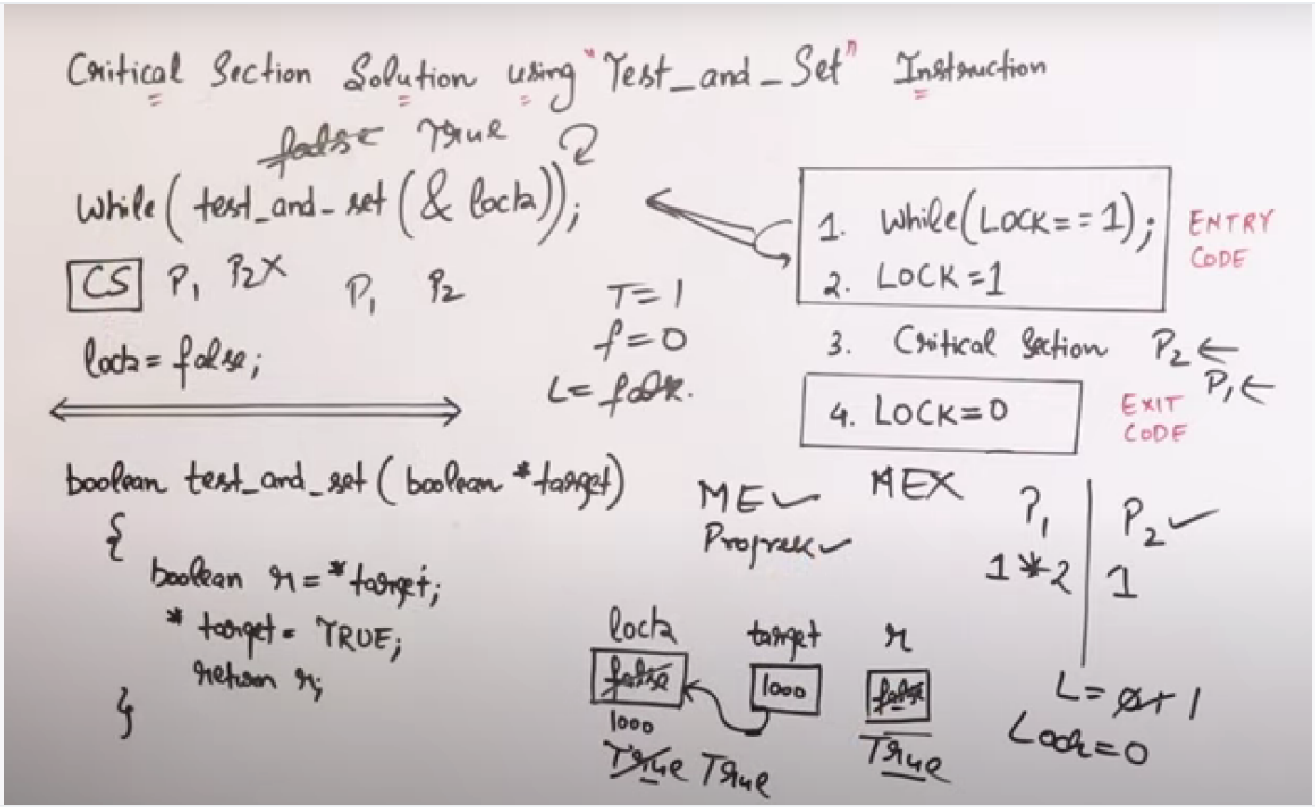


case 2 if p1 execute line 1 and prempt or stops due to some reason.



mutual exclusion was not satisfied by case 2

2. Test and set :

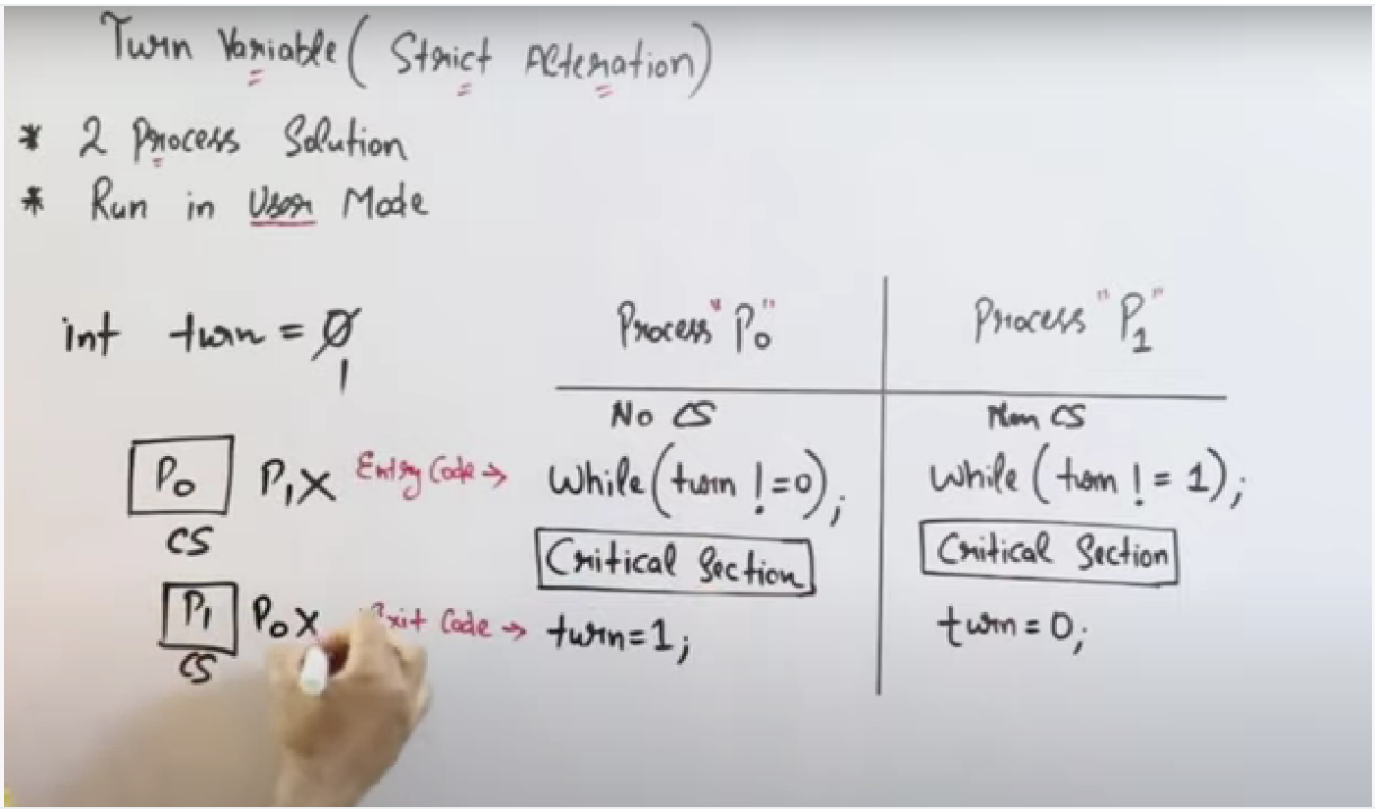


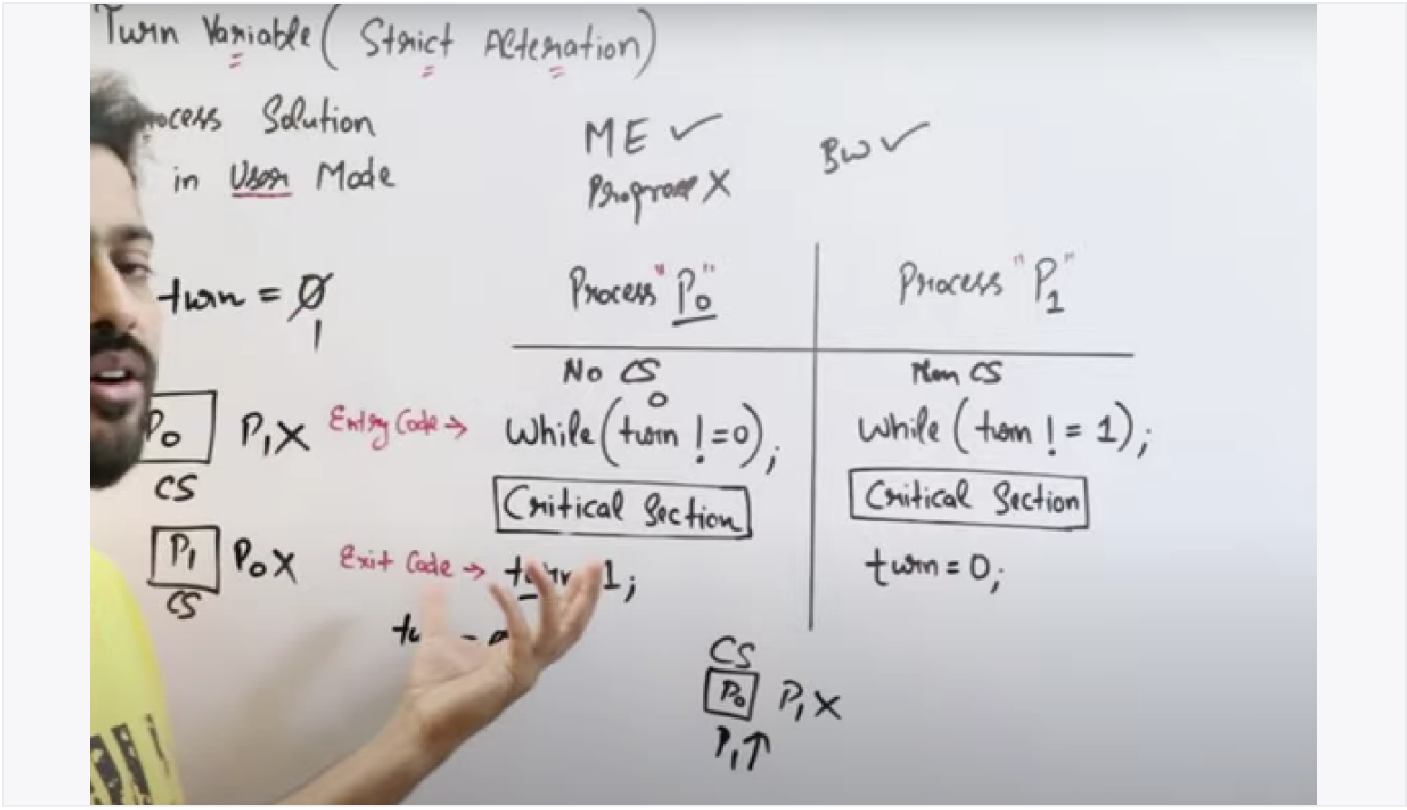
mutual exclusion and progress is satisfied.

3. TURN VARIABLE :

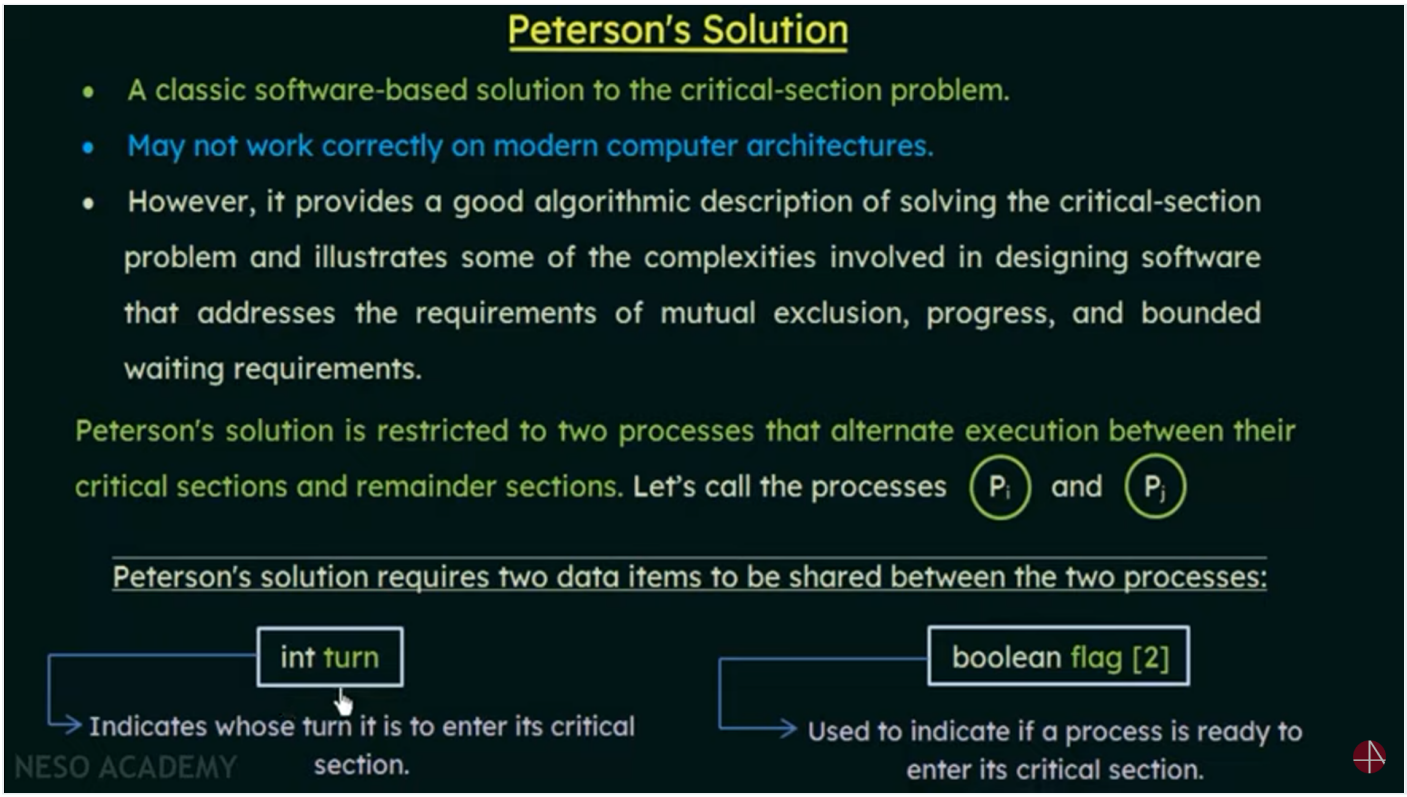
Process2 can only access critical section if process1 have already aceesed it otherwise it can't be accesed. hence satisfying the condition of mutual exclusion but not the progress, as without process1 process2 can't access CS.

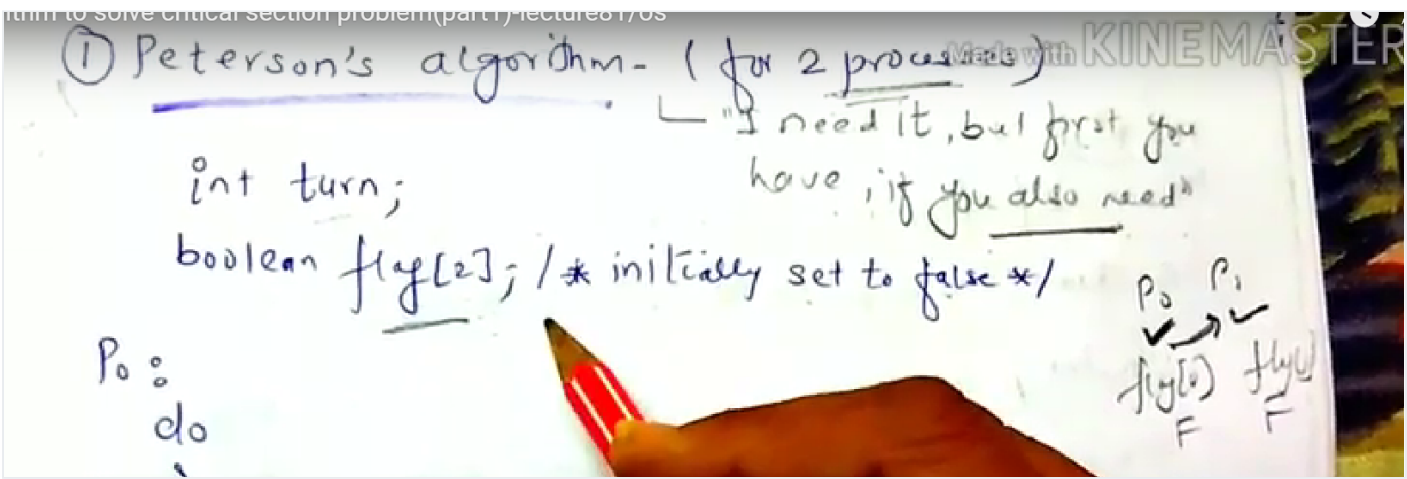
it is also called strict alteration after accessing CS once by P1, it can't again access it untill P2 doesn't access it.

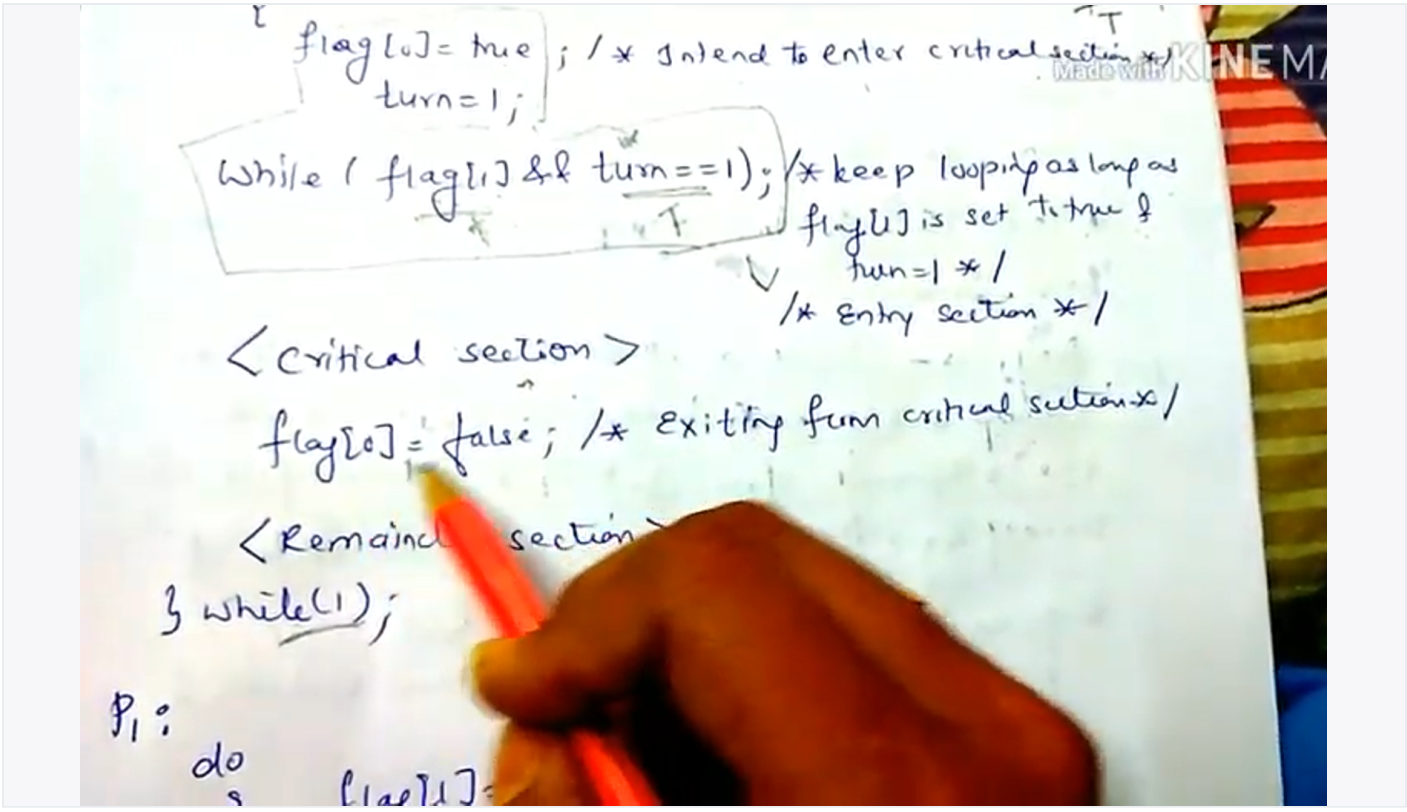


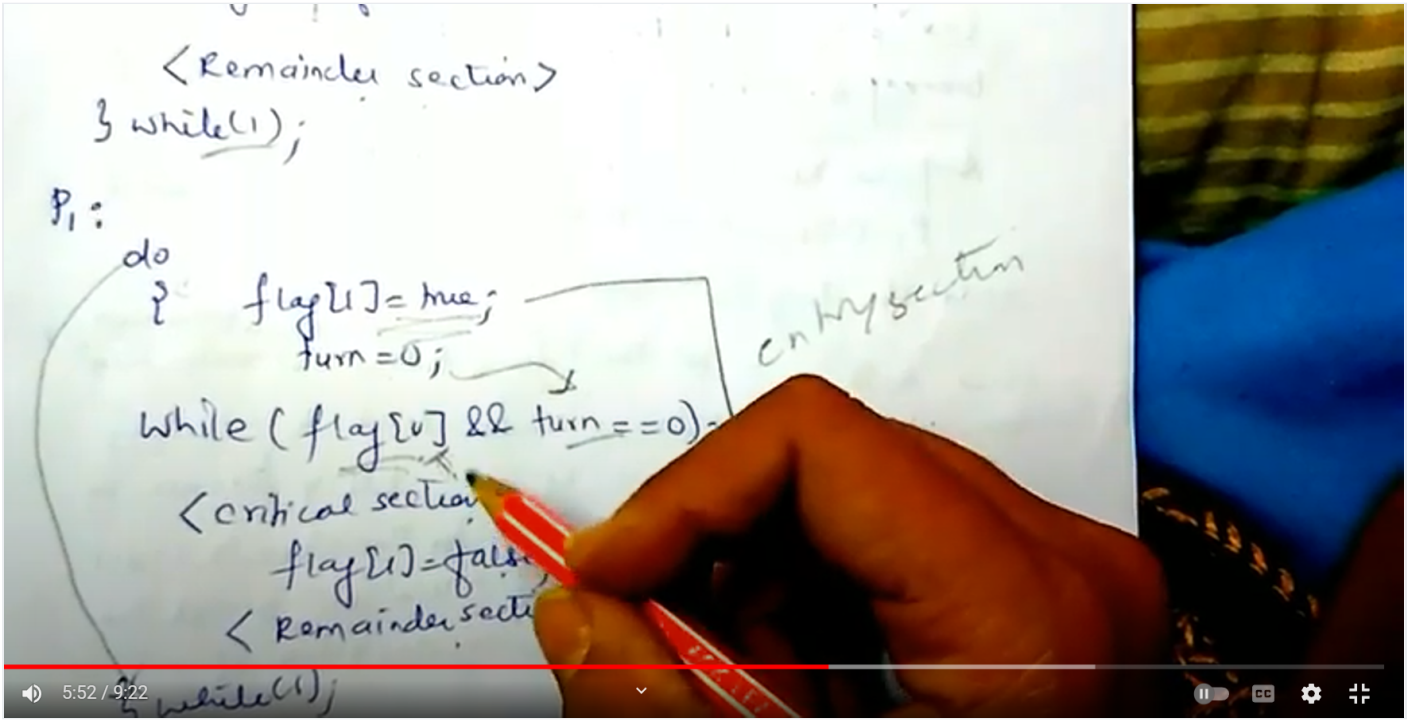


4. PETERSON'S SOLUTION









Mutual exclusion is satisfied.

Progress in satisfied

Bounded wating is satisfied.

PUSH PULL MIGRATION :

In multiprocessors, whicherver process have more processes to process will push their process to the processor who is having few number of process. same the processor with less number of processes will pull the processes.

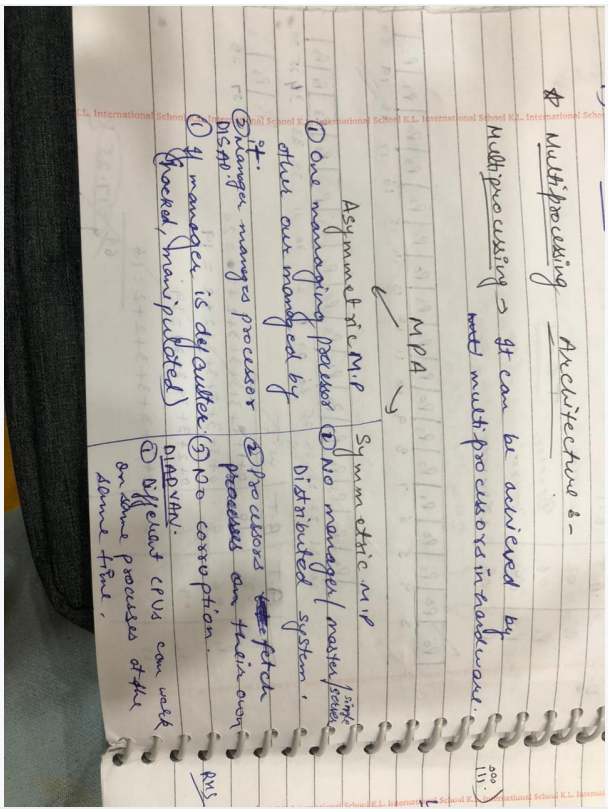
Multiprocessor architecture: multiple processors in the hardware.

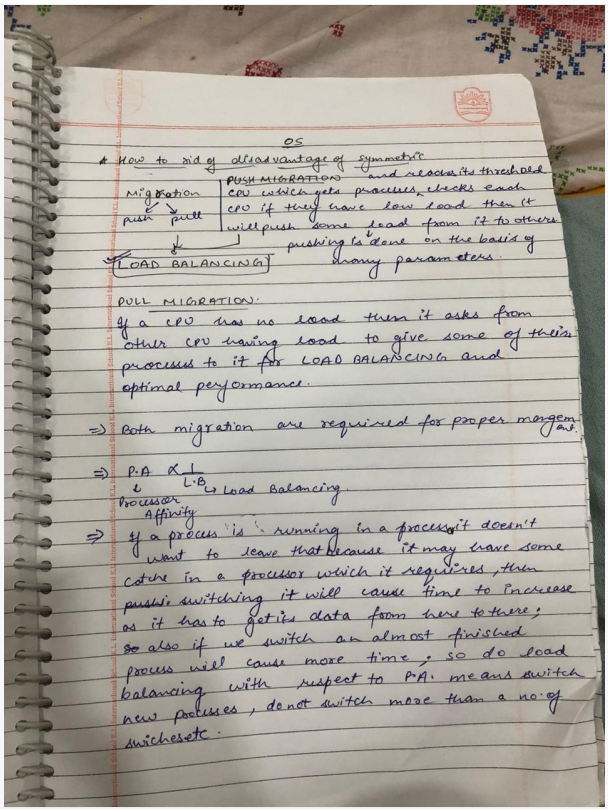
the system which have more than 2 processors.

they are of 2 types:

symmetric

assymetric multiprocessing





SHORTEST REMAINIG TIME FIRST IS THE PREMPTIVE VERSION OF SJF.

Advantages:

SRTF algorithm makes the processing of the jobs faster than SJN algorithm, given it's overhead charges are not counted.

Disadvantages:

The context switch is done a lot more times in SRTF than in SJN, and consumes CPU's valuable time for processing. This adds up to it's processing time and diminishes it's advantage of fast processing.