3. Process Synchronization:-

1) Entry section
2) Critical section
3) Exit Section
4) Remainder section

Synchronization:- Working in parallel

⇒ dependent process:- It depends on other process
(or)
(Cooperative process)

⇒ Cooperative process share common resources such as Database, etc....

⇒ Whenever this happens, then there should be Sychronization.

Critical section:-
A common code which will be allowing processor to access common resource.

Entry section:-
It allows request of the process for common resourse.

Exit section:-
After completion of execution, it allows to exit.

Remainder section:-
⇒ It is not related to critical Section.

=) Three conditions to be satisfied for a solution of synchronization (Critical section)

i) M.E (Mutual Exclusion):-

If any process is there in critical section, then another process is not allowed in to critical section. everytime there should be only one process in critical section.

ii) progress:-

If any process is not interested to enter into the critical section then it should allow (not stop) other process to enter into critical section.

Because there should be atleast one process in critical section everytime.

iii) Bounded wait:-

On waiting time of any process we should keep some bound on it.

(There should not be total starvation for any process).

=> (i) and (ii) conditions must satisfyed by solution

Two process solutions:-

1) Strict alternative solution/turn variable.
2) Flag variable / interest variable.
3) peterson's solution.

→ If any common region is shared by only two process is called two process solution.

→ If more than two, these solutions are not applicable.

1) Strict alternative soln | turn variable :-

P0

```
while (turn != 0); → //if condition is false.
{
  CS
  turn = 1;
}
```

P1

```
while (turn != 1);
{
  CS
  turn = 0;
}
```

→ turn variable is '0' and '1'.

⇒ If turn = '0' ⇒ P0 is executed

⇒ If turn = '1' ⇒ P1 is executed.

Drawback:-

⇒ Alternation of process.

⇒ Mutual Exclusion is Achieved.

⇒ Progress is not achieved.

becoz Initially, turn = 0, if P0 is not interested then never and ever turn will become '1' so, P1 will never enter into C.S.

therefore, progress is not achieved.

⇒ Bounded wait is also not achieved.

⇒ but it is a soln becoz atleast '1' is satisfied.

2) Flag variable / Interest variable :=

⇒ Initially Both values are zeros.

```
Flag[0] = T;                Flag[1] = T;
while(Flag[1]);             while (Flag[0]);
{                           {
  CS                          CS
  Flag[0] = F;                Flag[1] = F;
}                           }
```

⇒ Both have interest to enter into CS.

⇒ Here, Both are [T][T] then the problem comes. There Both the process will not enter into the critical section. So, we will get the condition that is "Deadlock".

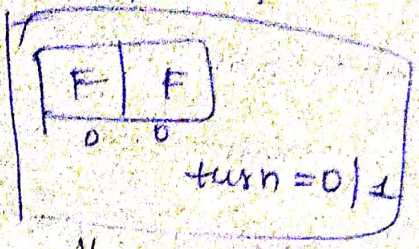
⇒ If a Deadlock is there then we can directly say that the progress is not achieved.

⇒ Mutual Exclusive is not achieved.

⇒ here, strict alternation is avoided.

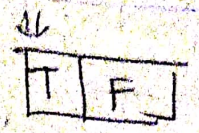
3) Peterson's Solution :-

Initially



P_0

```
Flag[0] = T;
turn = 1;
while (turn == 1 && Flag[1] == T);
{
    CS
    Flag[0] = F;
}
```

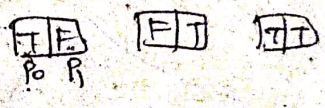


turn = 1 } P_0

⇒ It is a combination of both strict and flag.

P_1 :-

```
Flag[1] = T;
turn = 0;
while (turn == 0 && Flag[0] == T);
{
    CS
    Flag[1] = F;
}
```



i) 

⇒ only P_0 has interest.

do, now turn = 1 (initially turn = 0).

turn == 1 (true)
flag[] = T (false) } T && F ⇒ false.

So, P_0 will enter into critical section.

ii) [F | T]
P_0 P_1

⇒ P_1 has interest.

⇒ turn = 0

turn == 0 && Flag[0] = T

T && F ⇒ F

So, P_1 Enter into critical section.

iii) [T | T]

⇒ Both has interest.

⇒ first turn = 1.

⇒ second turn = 0 So, finally turn = 0.

⇒ Checking P_0 ⇒ turn == 1 && Flag[1] = T

F && F ⇒ F

⇒ Checking P_1 ⇒ turn == 0 && Flag[0] = T

T && T ⇒ T

So, here P_0 will enter into critical section.

Semaphores:-

Semaphores (S) = 1/0
mutex

```
wait(s)
{
  cs
}
signal(s)
```

```
wait(s)
{
  while(s==0);
  s=s-1;
}
```

```
signal(s)
{
  s=s+1;
}
```

⇒ Binary semaphores are called as mutex.

⇒ Initially mutex = 1

⇒ If mutex = 0, then process is already present in the critical section.

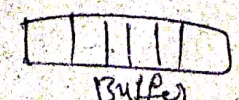
Problems: (Classical problems of Sychronization):-

① Producer – Consumer problem.

⇒ 'n' no. of producers can be there
'n' " " consumers " "
and some buffers are there.

⇒ Buffer shared b/w. producer & consumer will be kept in CS becoz at a time both producer and consumer cannot access.

⇒ If Buffer is in CS then some data can be kept.



counting semaphore: empty = n
full = 0

mutex = 1 ; } → Binary semaphor.

Producer

```
wait(empty);
wait(mutex);
produce
signal(mutex);
signal(full);
```

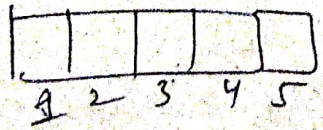
Consumer

```
wait(full);
wait(mutex);
Consumer
signal(mutex);
signal(empty);
```

⇒ For decrementation, wait is called.
⇒ If one empty buffer is filled with any other value
⇒ then it should be decremented.

⇒ To lock on buffer, mutex is used. (mutex)
So that others can't enter into C.S.

Eg:-



empty = 5; 4.
full = 0;
mutex = 1; 0 1

⇒ Mutual Exclusion is achieved.
⇒ Progress is also achieved.

(2) Readers – writers problem:-

⇒ A database can be accessed by both reader and writer

⇒ Whenever reader is there in C.S, it allows other reader but not allows other writer.

whenever writer is in C.S, it will not allows other reader and writer.

writer

```
wait(wrt);
   writing
signal(wrt);
```

wrt = 1
mutex = 1
readcount = 0

⇒ First reader has to block (or) put lock on writer not on complete C.S.

⇒ last reader has to release the lock.

⇒ This can be done by using read count.

⇒ read counter is a common variable, it can be used by multiple readers. So, it leads to inconsistency. So, mutex is used to put a lock on read counter.

Reader

```
wait(mutex);
 readcount ++;
 if(readcount == 1)
 wait(wrt);
      signal(mutex);
         reading
      wait(mutex)
      readcount --;
      if (readcount == 0)
      signal(wrt)
   signal(mutex)
```