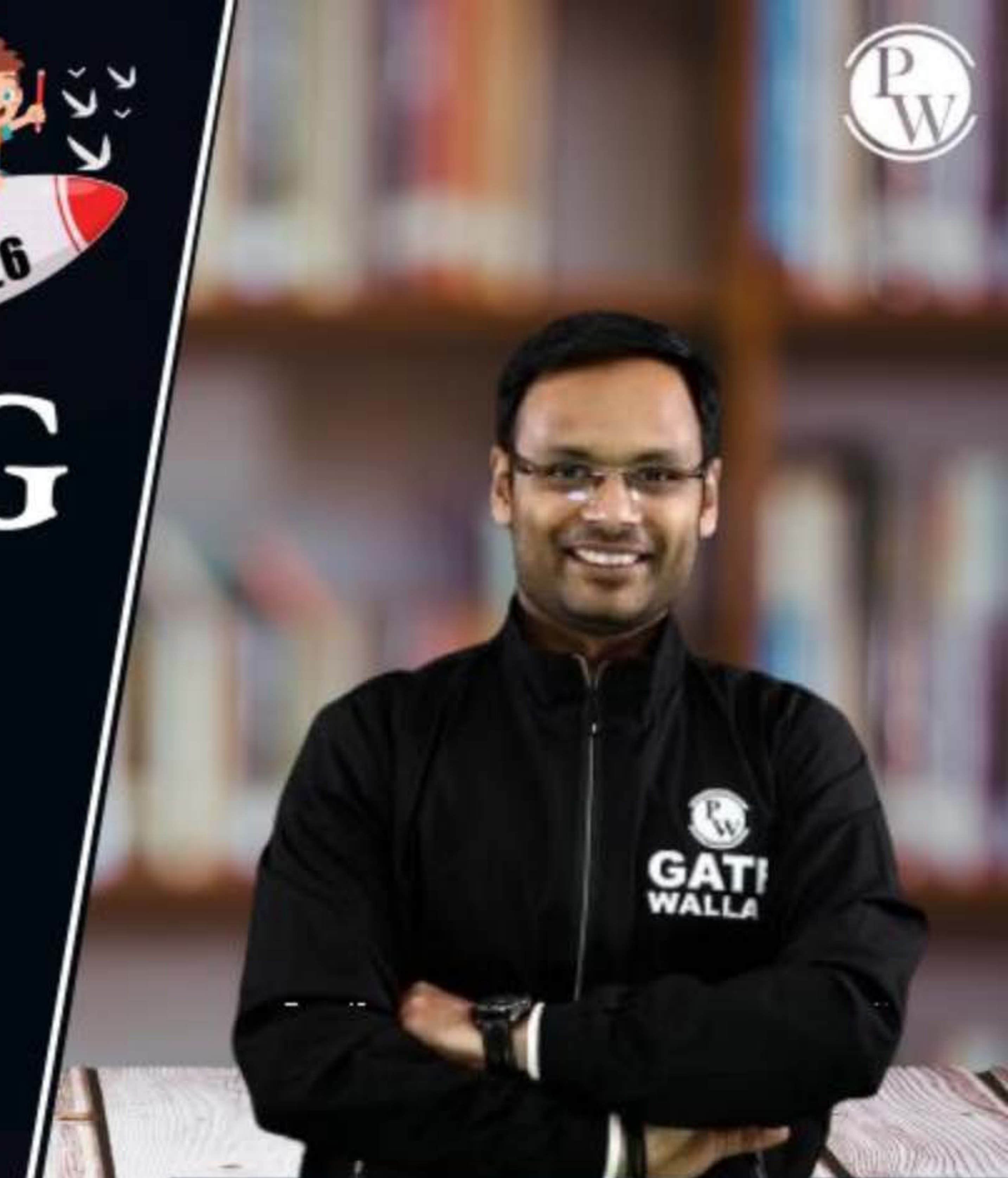
CS & IT ENGINEERING

Operating System

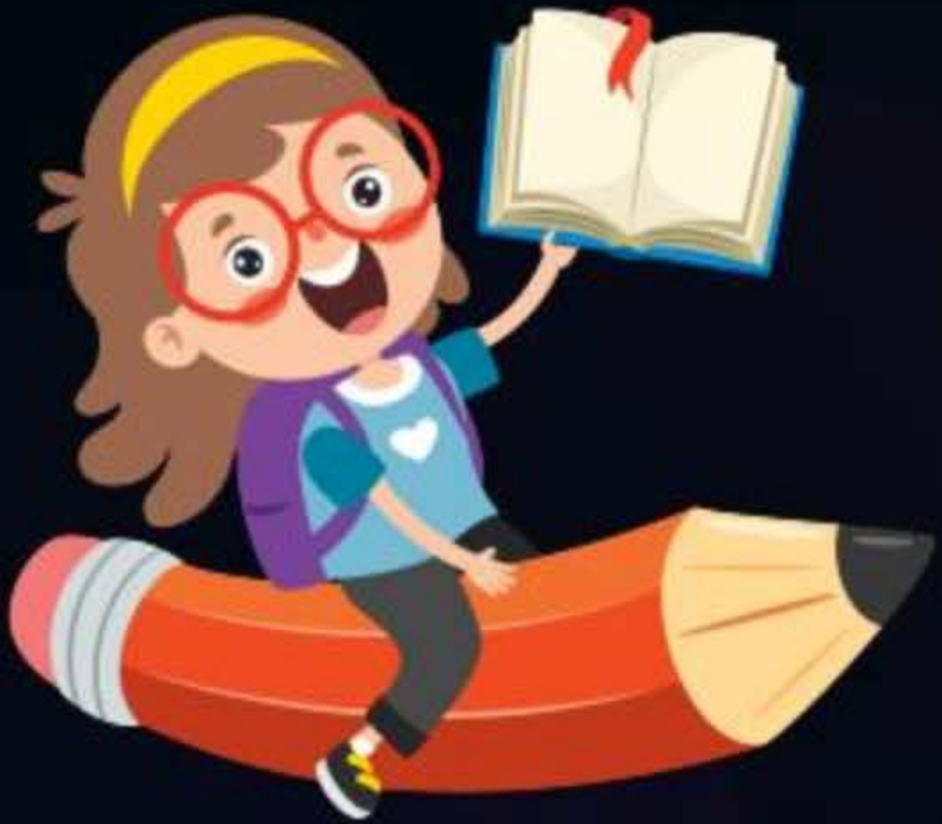
Process Synchronization



Lecture -2

By- Vishvadeep Gothi sir

Recap of Previous Lecture

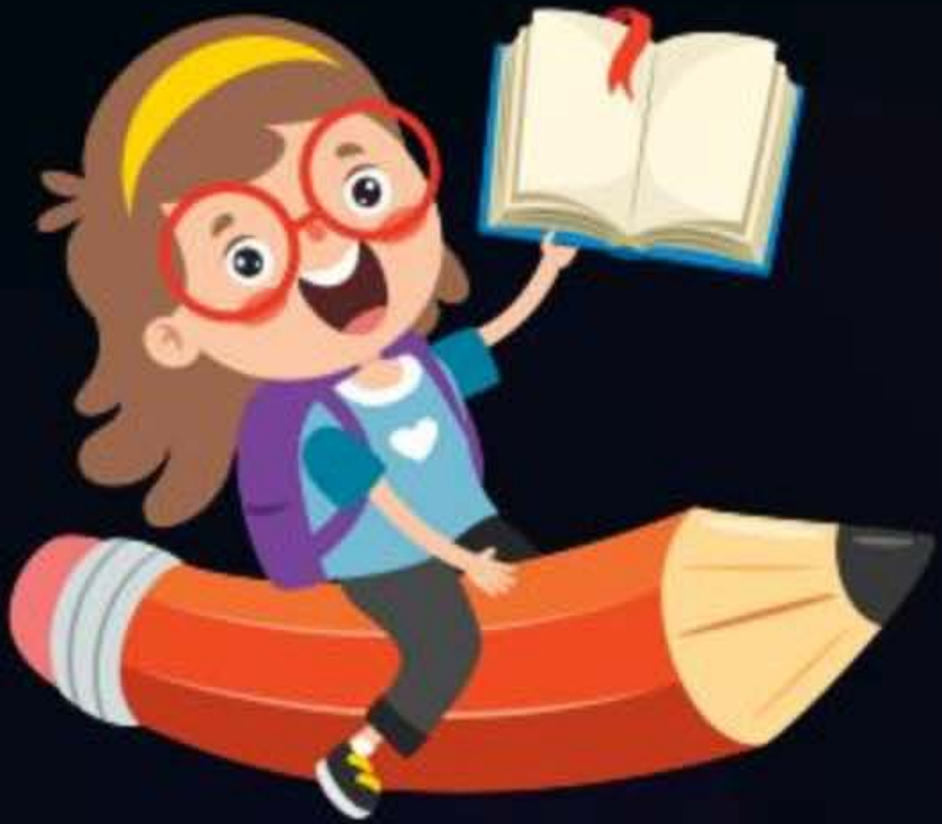


Topic Synchronization

Topic Race Condition

Topic Critical Section

Topics to be Covered



Topic Peterson's Solution

Topic Hardware Solutions of Synchronization

Topic Test-And-Set()

Topic Swap()

Topic Semaphore

Topic : Solution of Critical Section Problem

Requirements of Critical Section problem solution:

1. Mutual Exclusion
2. Progress
3. Bounded Waiting

if all fulfilled then solution is perfect

Topic : Solution of Critical Section Problem

Mutual Exclusion:

If one process is executing the critical section, then other process is not allowed to enter into critical section.

Topic : Solution of Critical Section Problem

Progress:

If no any process is in critical section and any process wants to enter into critical section, then the process must be allowed.

Topic : Solution of Critical Section Problem

Bounded Waiting:

If a process p1 is executing in critical section and other process p2 is waiting for critical section, then the waiting time of p2 must be bounded. Which means p1 must not enter in to critical section again and again by keeping p2 in waiting for long.

Topic : 2-Process Solution

$\boxed{\text{Entry section}}$

C.S.

$\boxed{\text{Exit section}}$

Solution 1

represents C.S. is open or locked

```
Boolean lock=false;
```

~~true~~ ~~false~~ True

P0

```
while(true)
{
    while(lock);
    lock=true;
        //CS
    lock=false;
    RS;
}
```

P1

```
while(true)
{
    while(lock);
    lock=true;
        //CS
    lock=false;
    RS;
}
```

× mutual Exclusion

✓ Progress

× Bounded waiting

strict alternation

Solution 2

```
int turn=0;
```

P_0

P_0 enters into cs when turn is zero.

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

P_1

P_1 enters into cs when turn is one

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

✓ mutual Exclusion

✗ progress

✓ Bounded waiting

Peterson's Solution

0	1
False	False

```
Boolean Flag[2]={False, False};
int turn;
```

→ announcement if any process wants to enter into C.S.

P0

```
while(true) {
  Flag[0]=true;
  turn=1;
  while(Flag[1] && turn==1);
     CS
  Flag[0]=False;
     RS;
}
```

P1

```
while(true){
  Flag[1]=true;
  turn=0;
  while(Flag[0] && turn==0);
     CS
  Flag[1]=False;
     RS;
}
```

✓ Mutual Exclusion
✓ Progress
✓ Bounded waiting

Question 1

H.W.

```
turn=0;

while(true)
{
    while(turn);
    turn=1;
      //CS
    turn=0;
    RS;
}
```

```
while(true)
{
    while(turn);
    lock=1;
      //CS
    lock=0;
    RS;
}
```

Question 2

H.W.

```
lock=False;

while(true)
{
    while(lock!=False);
    CS
    lock=True;
    RS;
}
```

```
while(true)
{
    while(lock!=True);
    CS
    lock=False;
    RS;
}
```

Question 3

H.W.

```
lock=False;

while(true)
{
    while(lock ==False)
    {
        lock = True;
    }
    CS
    lock=False;
    RS;
}
```

Question 4

H.W.

```
Boolean lock= True;

while(true)
{
    while(lock)
    {
        CS
        lock = False ;
    }
    lock=True;
    RS;
}
```

Topic : Synchronization Hardware

inst^{ns} in CPU

1. TestAndSet()
2. Swap()

Topic : TestAndSet()

Returns the current value flag and sets it to true.

Topic : TestAndSet()

Boolean Lock=~~False;~~ True

```
boolean TestAndSet(Boolean *trg){
boolean rv = *trg;
*trg = True;
Return  rv;
}
```

```
while(true)
{
 while(TestAndSet(&Lock));
        CS
 Lock=False;
}
```

✓ Mutual Exclusion

✓ Progress

✗ B.W.

Topic : Swap()

local (one for each process)

Boolean Key;

Boolean Lock=False;

[redacted]

↳ global

void Swap(Boolean *a, Boolean *b)

{

boolean temp = *a;

*a=*b;

*b=temp;

}

✓ M.E.

✓ Progress

× B.W.

P_0

key = ~~T~~ ~~F~~ ~~T~~ F

P_1

key = ~~T~~ T

lock = ~~False~~ ~~T~~ ~~T~~ ~~F~~ T

while(true){

Key = True;

while (key==True)

Swap(&Lock, &Key);

CS

Lock=False;

}

Topic : Synchronization Tool

1. Semaphore ⟹ int value which can be accessed using 2 functions ⟶ wait ( )
   signal ( )
2. Monitor

Topic : Synchronization Tool

- Integer value which can be accessed using following functions only
    1. wait() / P() / Degrade()
    2. signal() / V() / Upgrade()

atomic function

semaphore ⇒ by default ⇒ +ve int

→ -ve value possible if given in Question

Topic : wait() & signal()

assume semaphore S

```
wait(S)
{
   while(S<=0);
   S--;
}
```

⇓

if S is zero
then wait(S) can not
be completed successfully.

```
signal(S)
{
   S++;
}
```

Topic : Types of Semaphore

Binary Semaphore	Counting Semaphore
↓ accepts only 2 values ⇓ 0 or 1	⇓ value 0, 1, 2, 3, 4, ...

if s is a binary semaphore

wait() :-

$s = 1$

wait(s) ⇒ successful

$s = 0$

$s = 0$

wait(s) ⇒ unsuccessful

$s = 0$

signal() :-

$s = 0$

signal(s) ⇒ successful

$s = 1$

$s = 1$

signal(s) ⇒ successful

$s = 1$

Topic : Types of Semaphore

Binary Semaphore	Counting Semaphore
It is used to implement the solution of critical section problems with multiple processes	It is used to control access to a resource that has multiple instances

mutual exclusion

2 mins Summary

Topic Peterson's Solution

Topic Hardware Solutions of Synchronization

Topic Test-And-Set()

Topic Swap()

Topic Semaphore

Happy Learning

THANK - YOU