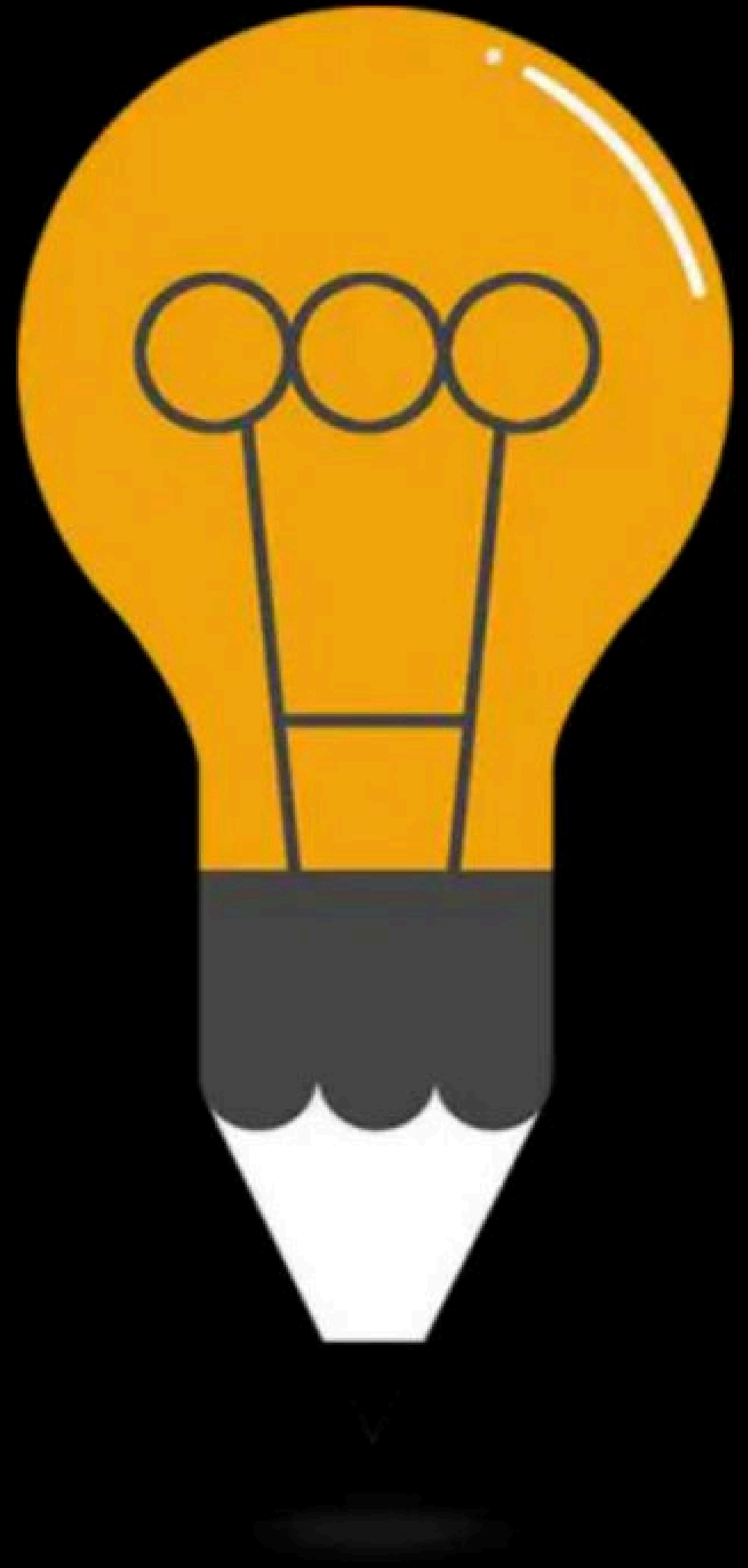
Synchronization Hardware & Semaphore

Comprehensive Course on Operating System for GATE - 2024/25

Vishvadeep Gothi • Lesson 14 • May 5, 2023



Operating System
Process Synchronization

By: Vishvadeep Gothi

Critical Section

The critical section is a code segment where the shared variables can be accessed

Solution of Critical Section Problem

Requirements of Critical Section problem solution:

1. Mutual Exclusion
2. Progress
3. Bounded Waiting

2-Process Solution

Solution 1

```
Boolean lock=false;
```

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

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

M.E. ✗

Progress ✓

B.W. ✗

Solution 2

C.S. Executn of P1 & P2 is in (Strict alternation)

```
int turn=0;
```

shared variable

P1

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

P2

```
while(true)

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

}
```

M.E. ✓

Progress ✗

B.W. ✓

Possible that P1 may starve or P2 may starve

when turn =0 P1 can enter into c.s.

when turn = 1 , P2 can enter into c.s.

checking M.E. :-

turn can be either 0 or 1.

Hence atleast one process will have while ( ) conditn true and other one will have false.

Progress :-

turn = 0   P1 can enter into CS

---

turn = 0   P2 can not enter in C.S.

Bounded waiting :-

P1 enters into c.s. | P2 waits for c.s. | & P1 try again to enter in c.s.

turn = ~~0~~ 1

Solution 3: Peterson's Solution

turn = ~~1~~ ~~1~~ ~~0~~ 1

shared

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

Flag[0] = False ~~T~~ ~~F~~ ~~T~~ ~~F~~ T

Flag[1] = False T

M.E. ✓

Progress ✓

B.W. ✓

=> No starvation

P0

entry section

```
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;
}
```

Flag[i] => indicates the intension of a process to enter into c.s. .

- false => not willing to enter into c.s.
- True => willing — " — " —

turn => indicates which process must get priority to enter into c.s. .

flag[0] = ~~F~~ T

flag[1] = ~~F~~ ~~T~~ ~~F~~ T

turn = ~~0~~ ~~1~~ 0

```
int turn = 2;

P0
while (true)
{
  while(turn != 1);
     C.S.
  turn = 2;
}

P1
while (true)
{
  while (turn != 0);
        c.s.
  turn = 2;
}
```

deadlock ✓
Progress ✗

Infinite waiting for both process

```
int turn = 1

P0
while (true)
{
  while(turn==0);
     C.S.
  turn = 0
}

P1
while (true)
{
  while (turn ==0);
     c.s.
  turn = 0
}
```

M.E. ✗
Progress ✓ → only once
B.W. ✓

Synchronization Hardware

1. TestAndSet()
2. Swap()

TestAndSet()

Returns the current value flag and sets it to true.

TestAndSet()

```
Boolean Lock=False;

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

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

Swap()

```
Boolean Key;                //Local
Boolean Lock=False;         //Shared

void Swap(Boolean *a, Boolean *b)
{
boolean temp = *a;
*a=*b;
*b=temp;
}
```

```
while(true){
 Key = True;
 while (key==True)
        Swap(&Lock, &Key);
    CS
 Lock=False;
    RS
}
```

Synchronization Tools

1. Semaphore
2. Monitor

Semaphore

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

wait() & signal()

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

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

Types of Semaphore

Binary Semaphore

Counting Semaphore

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

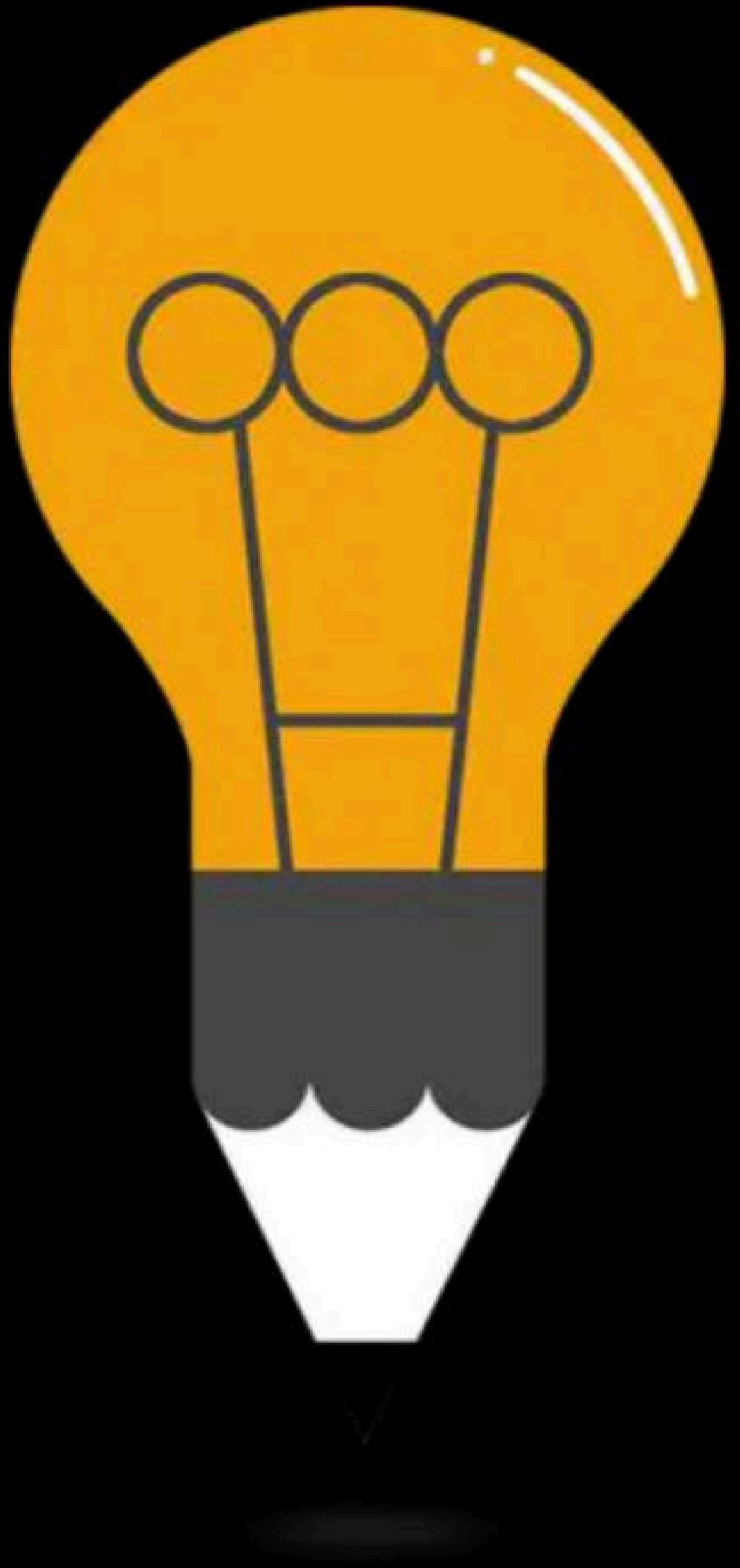
Critical Section Solution

```
S = 1

while(True)
{
  wait(S)
        C.S.
  signal(s)
}
```

Characteristics of Semaphores

- Used to provide mutual exclusion
- Used to control access to resources
- Solution using semaphore can lead to have deadlock
- Solution using semaphore can lead to have starvation
- Solution using semaphore can be busy waiting solutions
- Semaphores may lead to a priority inversion
- Semaphores are machine-independent



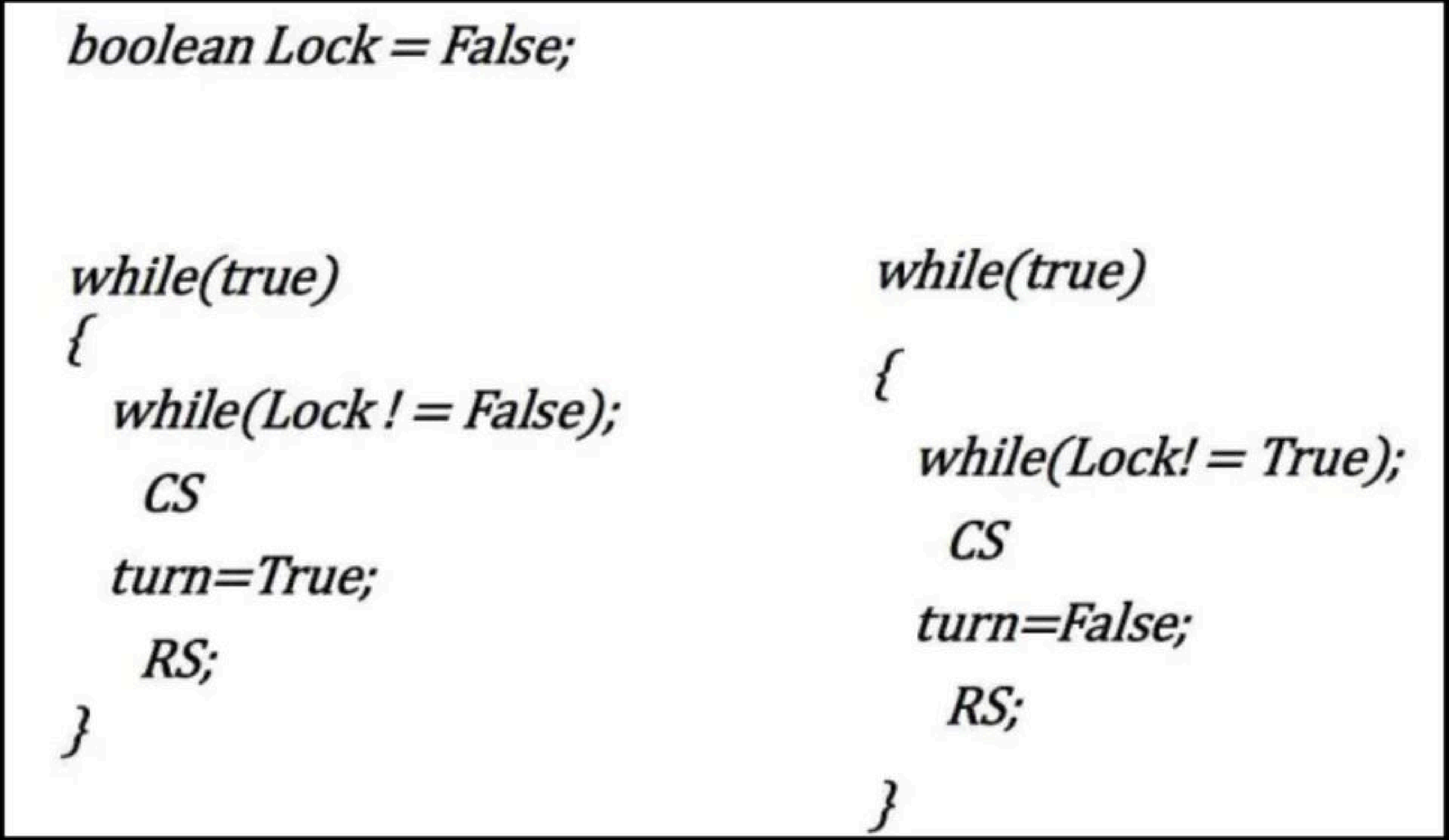
Tom.

11:00 am

DPP

By: Vishvadeep Gothi

Question 1



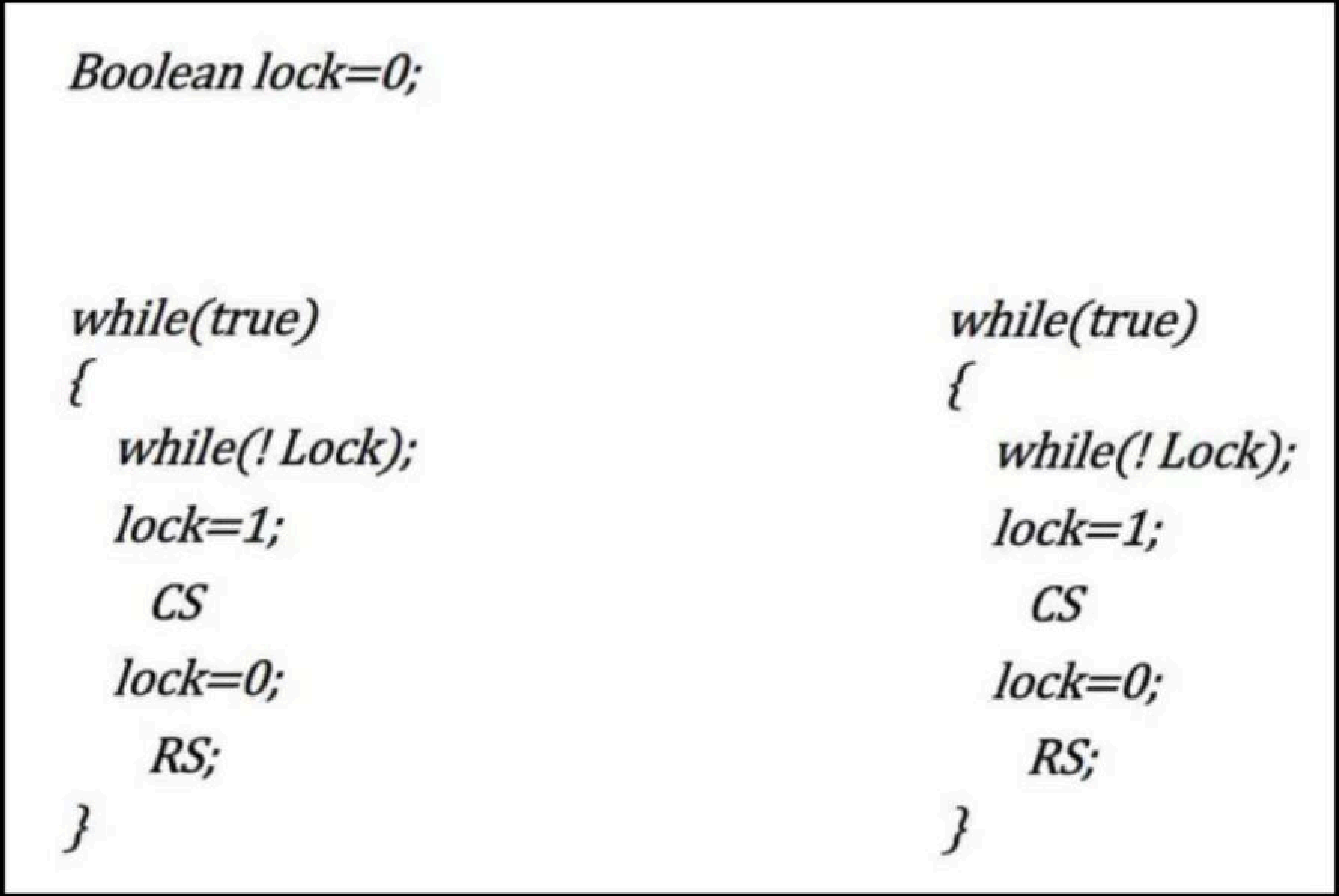
```
boolean Lock = False;



while(true)                          while(true)
{
   while(Lock ! = False);            {
     CS                                 while(Lock! = True);
   turn=True;                             CS
     RS;                                turn=False;
}                                         RS;

                                     }
```

Question 2



```
Boolean lock=0;



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

```
int turn = 0
```

P0

```
while (true)
{
  while (turn ! = 0);
      c.s.

  turn = 0;

}
```

P1

```
while (true )
{
  while (turn ! = 1);
      c.s.
  turn = 0;


}
```

Happy Learning.!