CS & IT ENGINEERING

Operating System

Process Synchronization

Lecture No. 6

By- Dr. Khaleel Khan Sir

- Synchronization Hardware
- TSL Instruction
- Swap Instruction

Peterson Soln

- 2-process Solution
- Busy-waiting <wastage of cpu time>
- u/m Impl.
- <L.V + S.A>

→ 3 Step Process

flag[N] = F;

turn = i/j

a) Guarantee M/E
b) " Progress
c) " Bounded Waiting

H/w

```
Process (int i)
{
    int j = NOT(i);
    while(1)
    {
      a) Non-CS()
      b) while (turn != i);
      c) turn = j;
      d) <CS>
      e) turn = j;
    }
}
```

a) M/E is not Guaranteed

Q.3

turn = 0

```
void Dekkers_Algorithm (int i)
{
    int j = ! (i);
    while (1)
    {
        (a) Non_CS();
        (b) flag[i] = TRUE;
        (c) while (flag [j] == TRUE)
        {
        if (turn == j)
        {
                flag [i] = FALSE;
                while (turn == j);
                flag [i] = TRUE;
        }
        }
        (d) <CS>
        (e) flag [i] = FALSE;
        turn = j;
    }
}
```

M/E ✓
Progress ✓
B/W ✓

Variation of → 2-Process

flag[N]; turn;

PETERSON'S SOLN

```
while (1)
{
a) Non-CS();
b) flag[i] = TRUE;
c) turn = i;
d) while (flag[j] == TRUE &&
                turn == i);
e) <CS>
f) flag[i] = FALSE;
}
```

Q.

Processes P_1 and P_2 use critical_flag in the following routine to achieve mutual exclusion. Assume that critical_flag is initialized to FALSE in the main program.

```
get_exclusive_access()
{
   if (critical_flag == FALSE)
   {
      critical_flag = TRUE;
      critical_region(); <<S>>
      critical_flag = FALSE;
   }
}
```

Category

Non-Busy-waiting

Consider the following statements.

(i) It is possible for both P_1 and P_2 to access critical region concurrently. ✓

(ii) This may lead to a deadlock. ✗

Which of the following holds?

A. (i) is false and (ii) is true

B. Both (i) and (ii) are false

C. (i) is true and (ii) is false ✓

D. Both (i) and (ii) are true

Q.

Two processes, P_1 and P_2, need to access a critical section of code. Consider the following synchronization construct used by the processes:

W1 = W2 = F;

```
/*P1*/
    while (true)
    {
            wants1 = true;
            while (wants2 == true);
            { /* Critical Section */
            wants 2 = false;
            }
    }
    /* Remainder section */
```

```
/*P2*/
    while (true)
    {
      wants2 = true;
      while (wants1 == true);
      { /* Critical Section */
            wants 2 = false;
      }
    }
    /* Remainder section */
```

Here, wants1 and wants2 are shared variables/ which are initialized to false. Which one of the following statements is TRUE about the above construct?

= MCQ

A. It does not ensure mutual exclusion.

B. It does not ensure bounded waiting.

C. It requires that processes enter the critical section in strict alternation.

D. It does not prevent deadlocks but ensures mutual exclusion.

Q. Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes

Process X

```
/* other code for process X */
while (true)
{
    varP = true;
    while (varQ == true)
    {
            /* critical section */
            varP = false;
    }
}
/* other code for process X */
```

Process Y

```
/* other code for process Y */
while (true)
{
    varQ = true;
    while (varP == true)
    {
        /* critical section */
        varQ = false;
    }
}
/* other code for process Y */
```

(Cont.....)

Q. Here, varP and varQ are shared variables and both are initialized to false. Which one of the following statements is true?

A. The proposed solution prevents deadlock but fails to guarantee mutual exclusion

B. The proposed solution guarantees mutual exclusion but fails to prevent deadlock

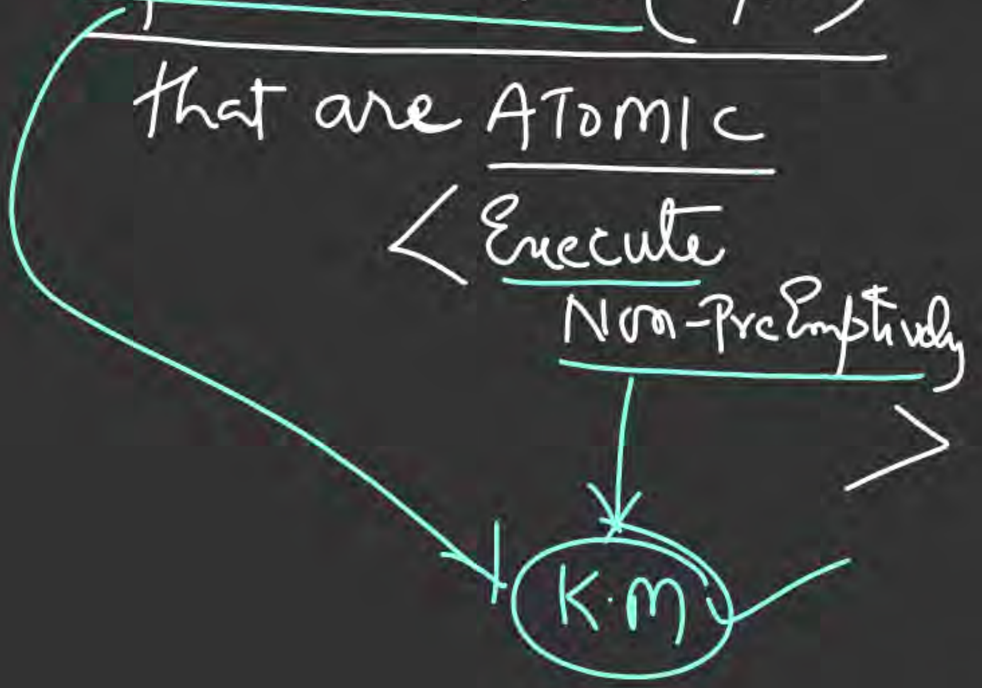
C. The proposed solution guarantees mutual exclusion and prevents deadlock

D. The proposed solution fails to prevent deadlock and fails to guarantee mutual exclusion

II: * SYNCHRONIZATION HARDWARE : Each Processor Supports some Special Instns (H/w) that are ATOMIC

a) TSL b) SWAP

→ Busy-waiting ✓
→ Can be used @ u/m as Spl. Instns
→ H/w Category ✓
→ Multi-process Soln
→ Lock-based Solutions



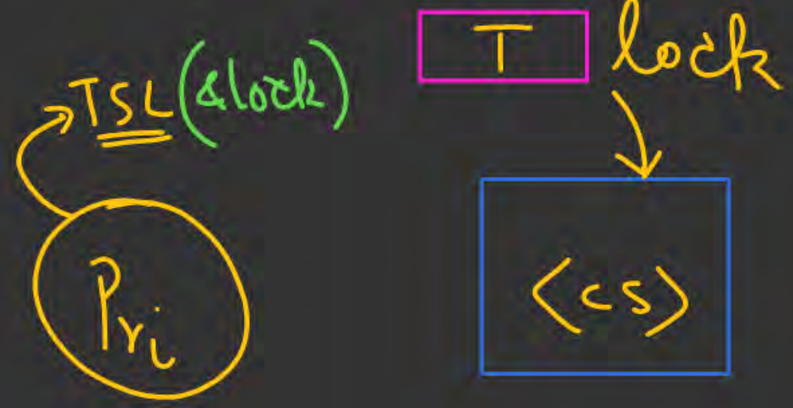
a) TSL : Test and Set Lock ;

Call: TSL(&lock);

< Process executing TSL, returns (read) the current value of lock & Sets (write) the value of lock always to TRUE >

```
Bool TSL(Bool *target)
{
    Bool rv;                    // return value
    rv = *target;               // Atomic Exec
    *target = TRUE;             // K.m
    return(rv);
}
```

→ Setting the value of lock to TRUE



Solving CS Problem thru TSL

User Program — U.M

In Galvin Text Book Synchronization

TSL: In the entry Sec (Additional Logic)

```
Bool Lock = F;

void Process(int i)
{
  while (1)
  {
    a) Non_CS();          // um
    b) while (TSL(&Lock) == T);   // using
    c) <CS>
    d) Lock = F;          // um, Exit
  }
}
```

Guarantee B/w

Sys. Call / Pv. Instn

RQ: P1 P2 P3

Lock: F T

CPU: P2

<CS>: P1

Guarantee M/E

t_1: P1 : a; TSL: F; Pre

t_2: P2 : a; TSL: T

Does not Guarantee Bounded wait

b) SWAP Instn <Atomic> → <lock-key>

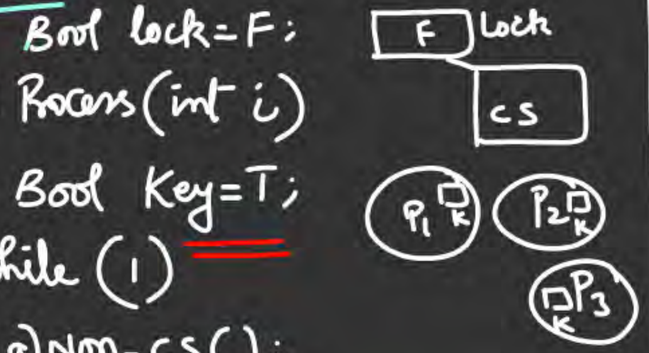
```
SWAP(Bool *a, Bool *b)
{
    Bool t;
Atomic {
    1. t = *a;
    2. *a = *b;
    3. *b = t;
}
```

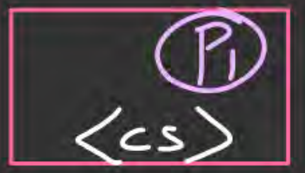
User Process

```
Bool lock=F;
Void Process(int i)
{
    Bool Key=T;
    While (1)
    {
      a) Non-CS();
      b) While (Key==T)
            SWAP(&lock, &Key);
      c) <CS>
      d) lock=F;
    }
}
```

B/W

Busy waiting





a) Guarantee M/E

b) " Progress

c) " B/W

(With Add on logic)

P_1 : a; .. SWAP ... Pre

P_2 : a; SWAP ↺

P_3 : a; SWAP ↺

Q.

The enter_CS ( ) and leave_CS () functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
void enter_CS(X)
{
while (test-and-set(X));
}
void  leave_CS(X)
{
X=0;
}
```

while (TSL(&lock)==T);
without Add m

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements:

I. The above solution to CS problem is deadlock-free. ✓

II. The solution is starvation free. ✗ Does not guarantee Bound wait

III. The processes enter CS in FIFO order. ✗

IV. More than one process can enter CS at the same time. ✗

Which of the above statements are TRUE?

A. I only ✓

B. I and II

C. II and III

D. IV only

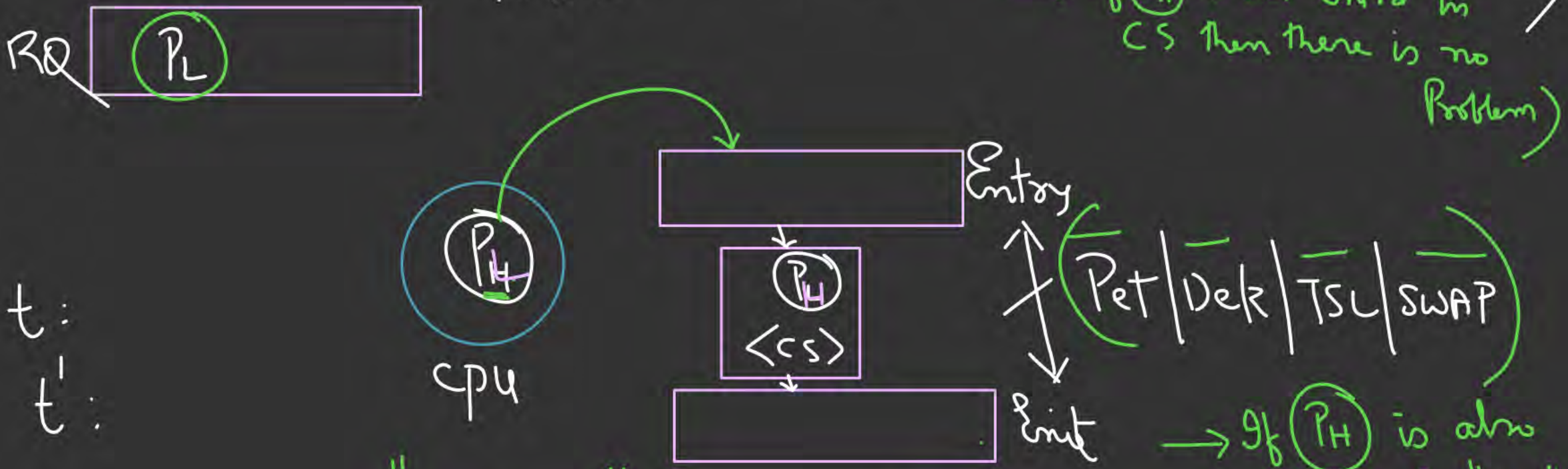
Priority - Inversion Problem :

: Pre Emptive Priority based Scheduling

< All Correct Busy-wait Soln Suffers from Priority Inv. Problem >

H > L

→ If P_H is NOT Intrd in CS then there is no Problem



→ If P_H is also intrsted in "CS"

"Deadlock"

Priority-Inheritance

Q.

Fetch_And_Add (X, i) is an atomic Read-Modify-write instruction that reads the value of memory location X, increments it by the value i and returns the old value of X, It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

2B

int i;

Entry

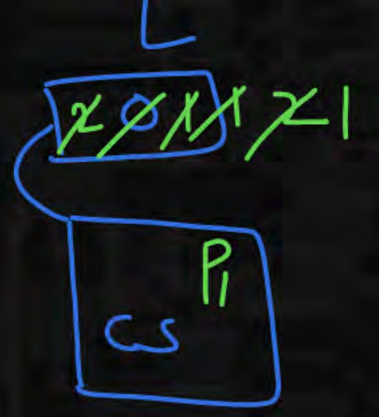
```
AcquireLock(L)
{
  While (Fetch_And_Add (L,1))
  L=1;
}
```

Exit

```
ReleaseLock(L)
{
  L=0;
}
```

Atomic

```
int F_And_A (int x, int i)
{
  int rv;
  rv = X;
  x = x+i;
  return (rv);
}
```



P_1: FA: 0

P_2: FA: 1; FA: 1

This implementation (MSQ)

A. fails as L can overflow

B. fails as L can take on a non-zero value when the lock is actually available

C. works correctly but may starve some processes

D. works correctly without starvation


THANK YOU
GW
SOLDIERS !