CS & IT ENGINEERING

Operating System

Process Synchronization

Lecture No. 3

By- Dr. Khaleel Khan sir

TOPICS TO BE COVERED

- Strict Alternation
- Peterson Algorithm
- Dekkers Algorithm

Q. Several concurrent processes are attempting to share an I / O device. In an attempt to achieve Mutual Exclusion each process is given the following structure. (Busy is a shared Boolean Variable)

CS

<code unrelated to device use> Non-CS

Repeat
until busy = false; while(busy==T); Lock variable
Busy = true;

<code to access shared > <CS>

Busy = false;

<code unrelated to device use>

Which of the following is (are) true of this approach?

I. It provides a reasonable solution to the problem of guaranteeing mutual exclusion.

II. It may consume substantial CPU time accessing the Busy variable.

III. It will fail to guarantee mutual exclusion.

A. I only

B. II only

C. III only

D. I & II

E. II & III

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();
    critical_flag = FALSE;
  }
}
```

Lock variable without 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

2) STRICT ALTERNATION

- → Busy waiting Construct
- → Software Soln
- → 2-process Soln

P0 P1

turn — 0 → P0's turn to enter CS

turn — 1 → P1's turn to enter CS

<CS>

turn (update)

The value of turn indicates, which Process turn to enter CS

Strictly on alternate basis

Processes takes turn to enter CS;

```
int turn = rand(0,1);
```

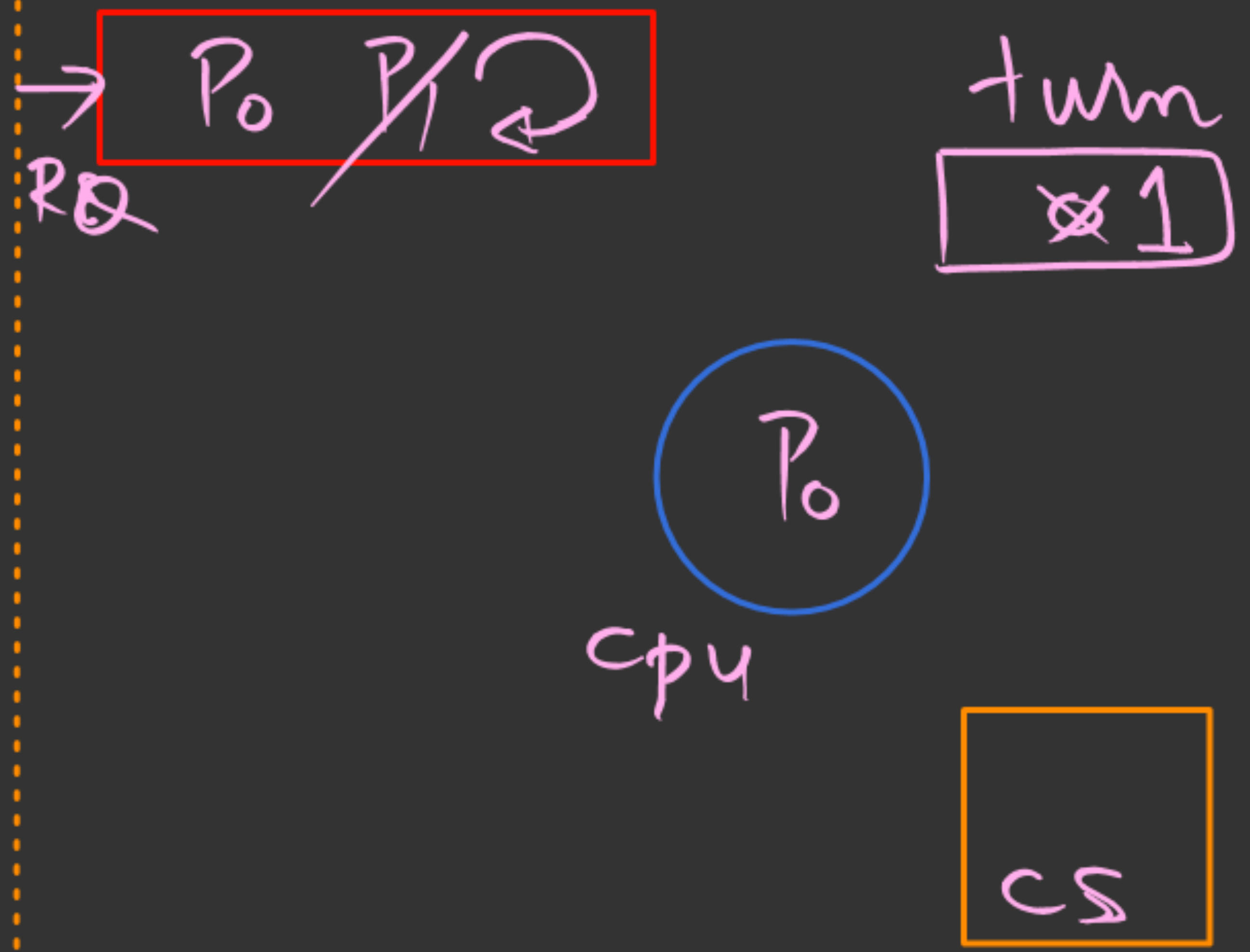
```
void Process(0)
{
  while(1)
  {
    Non_CS();
    while(turn != 0);
    <CS>
    turn = 1;
  }
}
```

```
void Process(1)
{
  while(1)
  {
    Non_CS();
    while(turn != 1);
    <CS>
    turn = 0;
  }
}
```

M.E : ✓

Progress : X

Bounded wait : ✓



Generalized Implementation of Strict Alternation

```
int turn = rand(i, j);

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

3) PETERSON SOLUTION

L·V + S·A

- → Busy waiting
- → S/W Solution
- → 2-process Solⁿ

```
#define N 2
#define TRUE 1
#define FALSE 0
int flag[N] = {FALSE}
int turn;
```

```
void Process(int i)
{
 int j = NOT(i);
 while(1)
 {
  a) Non-CS();
  b) flag[i] = T;
  c) turn = i;
  d) while (flag[j] == T && turn == i);
  e) <CS>
  f) flag[i] = F;
 }
}
```

(i) M.E : ✓
(ii) Progress : ✓
(iii) Bound wait : ✓

$flag[0] = \cancel{F}\ T$

$flag[1] = \cancel{F}\ \cancel{T}\ \cancel{F}\ T$

RQ: ~~P_0~~ ~~P_1~~

turn: $\boxed{1\ \cancel{1}\ \cancel{0}}$

CPU: P_1

<CS> P_1

t_1: P_0 $_{i=0;\ j=1}$: a; b; P_r

t_2: P_1 $_{i=1;\ j=0}$: a; b; c; d P_r

t_3: P_0 $_{i=0;\ j=1}$: c; d P_r

t_4: P_1 $_{i=1;\ j=0}$: d; <CS>

Bounded wait:

Limitations:

→ Can work with only 2-Processes;

→ waste cpu cycles (Time)

Q.2 Consider the following two-process synchronization solution.

Process 0

Entry: loop while (turn == 1);
(critical section)
Exit: turn = 1;

Process 1

Entry: loop while (turn == 0);
(critical section)
Exit: turn = 0;

Strict Alternation

The shared variable turn is initialized to zero.
Which one of the following is TRUE?

A. This is a correct two-process synchronization solution.

B. This solution violates mutual exclusion requirement.

C. This solution violates progress requirement.

D. This solution violates bounded wait requirement.

Q.

```
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;
    }
}
```

Assignment Q:

→ Peterson soln is an optimized version of Dekker's Algo;

Prove that Dekker's Algo Satisfy

(i) Mutual Exclusion
(ii) Progress
(iii) Bounded waiting

II : SYNCHRONIZATION HARWARE :

→ Each Processor Supports some Special Instructions (Architecture) that are atomic (N. Pre), which can be used to solve the Problem of critical Section,

(i) Disable Interrupt

(ii) TSL (Test and Set lock)

(iii) SWAP

$\left.\begin{array}{l} \text{(ii) TSL} \\ \text{(iii) SWAP} \end{array}\right\}$ Lock-based

1) Disable Interrupt:

P_1 P_2 P_3

Entry: Disable_Int

<CS>

Exit: Enable_Interrupts;

Restriction/Limitation:

→ Cannot be Implemented by user Processes @ U.M

→ It is Implementable by only OS Processes;

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:

```
/*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?

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


THANK YOU
GW
SOLDIERS !