Critical Section Problem: Solutions

28-Sep-2017
CS303
Autumn 2017

Simple solution (from lecture on 22-Sep-2016)

- Consider there are two processes in the system
  - Namely Process $\mathbf{P_i}$ and Process $\mathbf{P_j}$
- These are sharing a critical section (CS), where a common variable is modified
- The Processes content is as below:

P_i:
While (turn ≠ i) { }
CS
turn = j

P_j:
While (turn ≠ j) { }
CS
turn = i

- In the above, `turn` is an enumerated variable taking the value either `i` or `j`

Critical Section and Neighborhood

- Consider there are two processes in the system
  - Namely Process $\mathbf{P_i}$ and Process $\mathbf{P_j}$
- These are sharing a critical section (CS), where a common variable is modified
- The Processes content is as below:

P_i:
While (turn ≠ i) { }
CS
turn = j

P_j:
While (turn ≠ j) { }
CS
turn = i

- In the above, `turn` is an enumerated variable taking the value either `i` or `j`

Consider another version

- Use a flag array, which is initialised to *false*

  Shared Variable: boolean flag[2];

  initialised to *false*

```
P1:

while (flag[1]) {}
flag[0] = true;
CS
flag[0] = false;
```

```
P2:

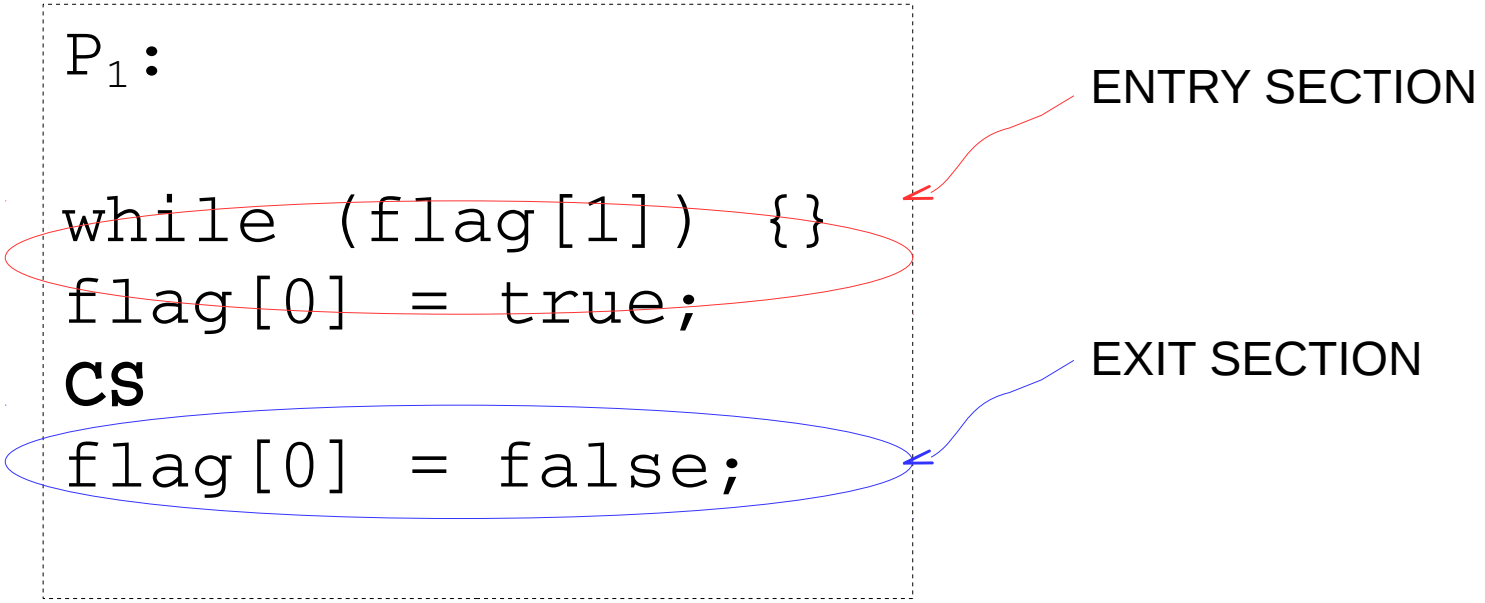
while (flag[0]) {}
flag[1] = true;
CS
flag[1] = false;
```

Mutual Exclusion is violated!

Solution 2

- Use a flag array, which is initialised to *false*

  Shared Variable: boolean flag[2];



Solution satisfying all the three criteria

- Past two algorithms failed the criteria for being a solution to critical section problem
- Consider the following:

shared variables: boolean flag[2]; //two process sol.

```
enum Turn { i, j};
Turn turn;
```

Process P_i

```
flag[i]=true; //expression of interest
turn = j; //if int. let other Pro. run
while (flag[j]& turn==j) {}
   CS
flag[i] = false;
```

Solution satisfying all the three criteria

- shared variables:

```
boolean flag[2]; //two process sol.

enum Turn { i, j};

Turn turn;
```

Process P_i

```
flag[i]=true;
turn = j;
while (flag[j]& turn==j) {}
   CS
flag[i] = false;
```

ENTRY SECTION

EXIT SECTION

Soln. With Two interacting Processes

Process P_i

```
flag[i]=true;
turn = j;
while(flag[j]& turn==j)
                      {}
   CS
flag[i] = false;
```

Process P_j

```
flag[j]=true;
turn = i;
while(flag[i]& turn==i)
                      {}
   CS
flag[j] = false;
```

This solution has: Mutual Exclusion, Progress and Bounded Waiting

Same impl. When N processes have a common Critical Section

- Good solution is given by Leslie Lamport under the name “Bakery Algorithm”
- It involves a kind of priority based access ans Shared variables as following:

```
enum
status{idle,want2enter_CS,in_CS};

enum Turn{1,2,3,...,n};

Turn turn;
```

Possible implementation with Hardware support

- The algorithms we have seen till now are purely software-based i.e. no support from HW is req.
- Two categories of solutions exist:
  - Software solutions
  - Hardware solutions (HW shall provide the required facilty for these solutions)

Hardware Solutions: solutions that are dependent on HW support

- The idea of HW solutions is to impose guaranteed atomic execution of certain calls

- Examples for such HW functions are: test_and_set, swap...
  - boolean test_and_set(boolean lock)
  - Prototype of function

```
boolean test_and_set(boolean *tVar){
boolean temp = *tVar;
*tVar= TRUE;
return temp;
}
```

Hardware-based functions

- Swap function...
  - ```
    swap(boolean x1, boolean x2){
    temp = x1;
    x1 = x2;
    x2 = temp;
    }
    ```
    ```

Hardware Solution to Critical Section Problem

```
Common data structures:
boolean interested_CS[n];//no. of processes 0,1,…,n-1
boolean lock; // lock for a given CS init. To FALSE

Local D/S: j and key // init. to false
Process P_i: //implementation
   int j; // for proc ids
   boolean key; // for accessing the lock init. to false
   interested_CS[i] = true; // initialised to false for all
   key = true;
   while(interested_CS[i]&&key) {key=test_and_reset(lock);}

      CS
   j = (i+1) mod n;
   while ((j <> i) && ~(interested_CS[j])) {j = (j+1) mod n}
   if (j==i) then lock = false;
   else interested_CS[j] = false;
```

Hardware Solution to Critical Section Problem

```
Shared Variables:
boolean interested_CS[n];//no. of processes
boolean lock; // lock for a given CS

Process P_i: //implementation
   int j; // for proc ids
   boolean key; // for accessing the lock init. to false
   interested_CS[i] = true; // initialised to false for all
   key = true;
   while(interested_CS[i]&&key) {key=test_and_reset(lock);}
      interested_CS[i] = false;
      CS
   j = (i+1) mod n;
   while ((j <> i) && ~(interested_CS[j])) {j = (j+1) mod n}
   if (j==i) then lock = false;
   else interested_CS[j] = false;
```