EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 6: September 08

Lecturer: Vijay Garg *Scribe: Jiaolong Yu*

6.1 Lower Bound on the Number of Shared Memory Location

Theorem 6.1 *(Burns and Lynch)Any mutex algorithem that uses only RW on n processes require at least n shared locations.*

Proof: Consider 2 processes, say P and Q, in competing for Critical Section. They have only one shared memory location A. Let Q run till it's about to write to A. Let P run and enter critical section. Let Q run again. It enters CS. Mutex violation. Consider 3 processes, say P, Q and R. They have 2 shared memory locations A and B. Let P and Q run till they are about to write to A. Let R run and enter critical section. let P and Q run again. One of them will enter critcal section. Mutex violation. With this method we can extend the situation to n processors and prove the theorem. ∎

Definition 6.2 *Covering state. All share variables are about to be overwritten by processes and the shared stats is consistant with no process in CS.*

6.2 Fischer's Algorithem

Turn = -1 Means the door is open.
RequestCS:

```
while(ture) {
  while(turn != -1);
  turn = i;
  wait_for_delta_time_units();
  if(turn == i) return;
}
```

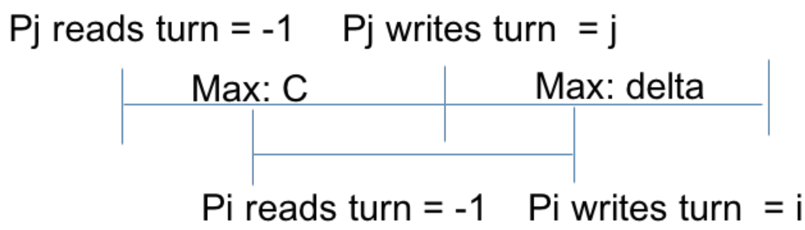
ReleaseCS:

```
turn = -1
```

This algorithm can cause mutex violation. One senario:
P_i reads turn = -1
P_j reads turn = -1
P_j sets turn = j
P_j reads turn = $jandenterCS$
P_i sets turn = i
P_i reads turn = i and enter CS

Theorem 6.3 *Assuming delta >= C, Fischer's Algorithm satisfies mutex. C: maximum time required to close the door, ie. set the turn.*

Proof: Let P_i be the processor that enters CS successfully. Assume there is another processor P_j that may enter CS.

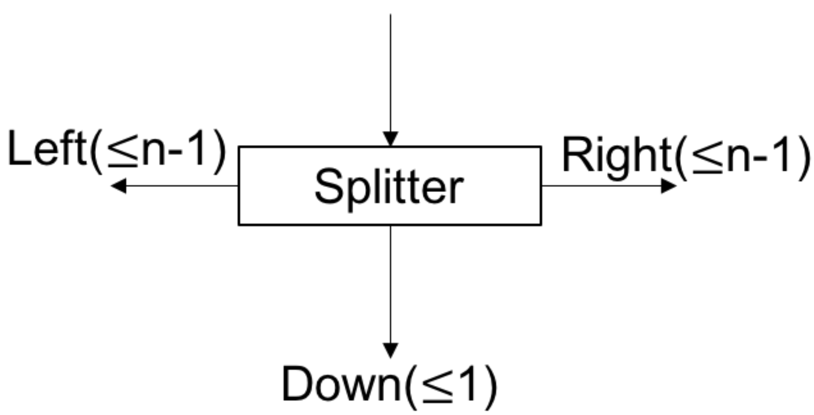


P_j must writes turn after P_i reads it to be -1. If it writes before P_i writes turn, after waiting for delta units of time which is longer than C, then P_i must have already finished writing turn, P_j will defnitely read turn = i then it will not enter CS. If P_j writes turn after P_i writes turn, then P_i will read turn = j after waiting delta units of time, then it will not enter CS. Proved that only one processor will enter CS ∎

6.3 Lamport's Fast Algorithm

This algorithm enables for processors to enter CS fast when there is no contention.
No contention – fast path Contention – slow path

6.3.1 Splitter




```
For every P_i:
  Variables:
    door: {open, closed}, initially open
    last: pid, initially -1
  last = i;
  if (door == closed) return left;
  else {
    door = closed;
    if (last == i) return down;
```

```
  else return right;
}
```

Claim 6.4 $|left| <= n - 1$

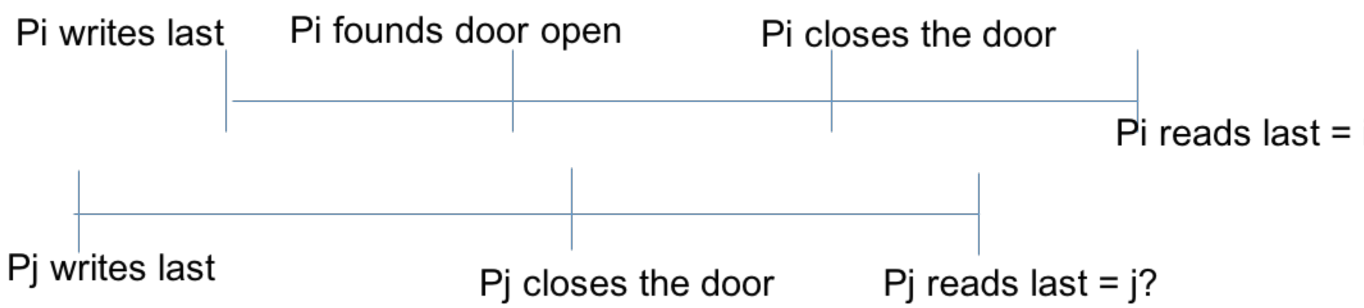
Proof: Someone need to close the door ■

Claim 6.5 $|right| <= n - 1$

Proof: Consider processor P$_i$ such that last = i then P_i is part of left or down. So as at least one process would not go right. ■

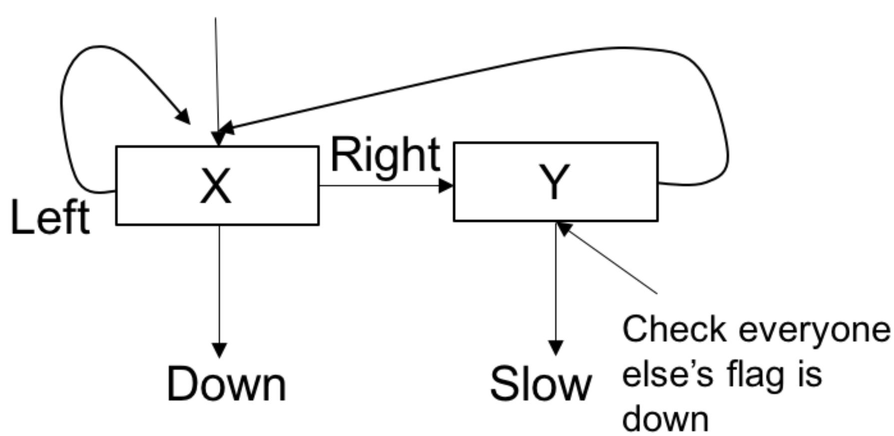
Claim 6.6 $|down| <= 1$

Proof: Let P$_i$ be the first process to go down. Assume there is another processor P_j that may go down.



As shown in the picture, if P_j does everything before P_i writes last, then P_j would be the first one to go down, which conflicts with our assumption. However, P_j must writes to last before P_i does otherwise P_i would not be able to read last = i at the end. P_j must closes the door after P_i checks the door, otherwise P_i would have found the door closed, then it cannot go down. Then we have P_i writes to last before P_i checks the door, before P_j closes the door, before P_j reads last. It means P_j must reads last = i then it can not go down. ■

6.3.2 Lamport's Fast Algorithm



RequestCS:

```
while (ture) {
  flag[i] = up;
  x = i;
  if (y != -1) { // split left
    flag[i] = down;
    waituntil( y == -1);
    continue;
  } else {
    y = i;
    if (x == i) return; // down
    else { // right
      flag[i] = down;
      waituntil(y = -1);
      continue;
    }
  }
}
```

ReleaseCS:

```
y = -1;
flag[i] = down;
```