CS586: Distributed Computing

Tutorial 4

Professor: Panagiota Fatourou
TA: Eleftherios Kosmas
CSD - October 2010

Anderson's Algorithm: The Array-Based Lock

Process pi

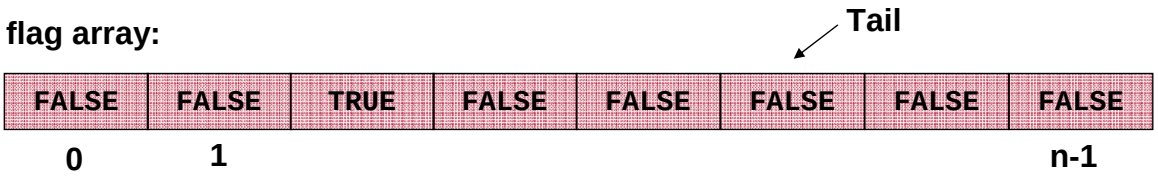
```
1.  int slot = -1;

2.  void lock(void) {
3.      slot = Fetch&Inc(&Tail);
4.      while (!Flag[slot % n]) noop;
5.  }

6.  void unlock(void) {
7.      flag[(slot + 1) % n] = TRUE;
8.      flag[slot % n] = FALSE;
9.  }
```

Initially

Tail = 0

flag[i] = false

- Anderson's Algorithm is still correct if we exchange lines 7 and 8?



CLH Lock

Process pi

```
1.  void lock(void) {
2.      MyNode.locked = TRUE;
3.      MyPred = Get&Set(&Tail, MyNode);
4.      while (MyPred->locked == TRUE) noop;
5.  }

6.  void unlock(void) {
7.      MyNode->locked = FALSE;
8.  }
```

Initially

Shared Memory

Tail : points to a NODE n with n.locked == FALSE

Global Local Memory

MyNode : points to a struct NODE

MyPred = NULL

```
typedef struct {
boolean locked;
} NODE;
```

- CLH lock is still correct if a thread reuses its own node, instead of its predecessor node?

TTAS Lock

Process pi

```
1.   void lock(MemoryByte *pB) {
2.       while (TRUE) {
3.           while (*pB == TRUE) noop;
4.           if (Test&Set(pB) == FALSE)
5.               return;
6.       }
7.   }

8.   void unlock(MemoryByte *pB) {
9.       reset(pB);
10.  }
```

Initially

*pB = FALSE

- Design an isLocked() method that tests whether a thread is holding a lock
  - but does not acquire the lock

CLH Lock

Process pi

```
1.  void lock(void) {
2.      MyNode.locked = TRUE;
3.      MyPred = Get&Set(&Tail, MyNode);
4.      while (MyPred->locked == TRUE) noop;
5.  }

6.  void unlock(void) {
7.      MyNode->locked = FALSE;
8.      MyNode = MyPred;
9.  }
```

Initially

Shared Memory

Tail : points to a NODE n with n.locked == FALSE

Global Local Memory

MyNode : points to a struct NODE

MyPred = NULL

```
typedef struct {
boolean locked;
} NODE;
```

- Design an isLocked() method that tests whether a thread is holding a lock
  - but does not acquire the lock

MCS Lock

Process pi

```
1.  void lock {
2.      MyPred = Get&Set(&Tail, MyNode);
3.      if (MyPred != NULL) {
4.          MyNode->locked = TRUE;
5.          MyPred->next = MyNode;
6.          while (MyNode->locked) noop;
7.      }
8.  }

9.  void unlock {
10.     if (MyNode->next == NULL) {
11.         if (Compare&Swap(&Tail, MyNode, NULL) == TRUE) return;
12.         while (MyNode->next == NULL) noop;
13.     }
14.     MyNode->next->locked = FALSE;
15.     MyNode->next = NULL;
16. }
```

Initially

Shared Memory

Tail = NULL

Global Local Memory

MyNode : points to a struct NODE

MyPred = NULL

```
typedef struct {
boolean locked;
struct NODE *next;
} NODE;
```

- Design an isLocked() method that tests whether a thread is holding a lock
  - but does not acquire the lock

Mutual Exclusion - Exercise 1

Process p0

```
1.  trying = 0
2.  if (busy == 1) goto 1;
3.  busy = 1;
4.  if (trying == 1) {
5.      busy = 0;
6.      goto 1;
7.  }
8.  critical section;
9.  busy = 0;
10. remainder section;
```

Process p1

```
1.  trying = 1
2.  if (busy == 1) goto 1;
3.  busy = 1;
4.  if (trying == 0) {
5.      busy = 0;
6.      goto 1;
7.  }
8.  critical section;
9.  busy = 0;
10. remainder section;
```

- Ensures mutual exclusion?
- Ensures no deadlock?
  - if yes, prove it
  - if not, present a counter example

Initially

busy = 0
trying = -1

Mutual Exclusion - Exercise 2

- Prove that the order of lines 11 and 12 in the Black-White Bakery algorithm is crucial for correctness.

Process pi

```
1.   choosing[i] = TRUE;
2.   mycolor[i] = color;
3.   number[i] = 1 + max (number[j] | 1≤j≤n AND mycolor[i] == mycolor[j]);
4.   choosing[i] = FALSE;
5.   for (j = 1; j <= NUM_THREADS; j++)  do
6.      wait until (choosing[j] == FALSE)
7.      if (mycolor[i] == mycolor[j]) then
8.            await ( number[j] == 0  OR  (number[j], j) > (number[i], i) OR
                        mycolor[j] ≠ mycolor[i] );
9.      else await ( number[j] == 0  OR  mycolor[i] ≠ color OR
                        mycolor[j] == mycolor[i] )
10.  critical section;
11.  number[i] = 0;
12.  if (mycolor[i] == black) then color = white;
     else color = black;
     remainder section;
```

Initially

number[i] = 0
choosing[i] = 0

The End - Questions