CS586: Distributed Computing

Tutorial 3

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

Mutual Exclusion - Exercise 1

- Present an execution of the Bakery Algorithm where the sequence of values that are stored in each *Number[j]* ($1 \le j \le n$), is strictly increasing.

Process pi

```
1.  Choosing[i] = TRUE;
2.  Number[i] = 1 + max(Number[0], ..., Number[n-1]);
3.  Choosing[i] = FALSE;
4.  for (j = 1; j <= NUM_THREADS; j++)  do
5.        wait until (Choosing[j] == FALSE)
6.        wait until ((Number[j] == 0)  OR  ((Number[j], j) > (Number[i], i)));
7.  critical section;
8.  Number[i] = 0;
    remainder section;
```

Initially

Number[i] = 0

Choosing[i] = 0

Mutual Exclusion - Exercise 2

- Is the Bakery Algorithm correct if all *choosing[j]* variables ($1 \le j \le n$), as well as all code lines referring to them, are omitted in the code?
  - prove your claim

Process pi

```
1.  Number[i] = 1 + max(Number[0], ..., Number[n-1]);
2.  for (j = 1; j <= NUM_THREADS; j++)  do
3.        wait until ((Number[j] == 0)  OR  ((Number[j], j) > (Number[i], i)));
4.  critical section;
5.  Number[i] = 0;
    remainder section;
```

Initially

Number[i] = 0

Mutual Exclusion - Exercise 3

Process p0	
1.	await (busy == 0) ;
2.	trying = 0;
3.	if (busy == 1) goto 1;
4.	busy = 1;
5.	if (trying == 1) goto 1;
6.	critical section;
7.	busy = 0;
8.	remainder section;

Process p1	
1.	await (busy == 0) ;
2.	trying = 1;
3.	if (busy == 1) goto 1;
4.	busy = 1;
5.	if (trying == 0) goto 1;
6.	critical section;
7.	busy = 0;
8.	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 4

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 - Black-White Bakery

Process pi

```
1.  choosing[i] = TRUE;
2.  mycolor[i] = color;
3.  number[i] = 1 + max (number[j] | 1≤j≤n AND mycolor[j] == mycolor[i]);
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. if (mycolor[i] == black) then color = white;
    else color = black;
12. number[i] = 0;
    remainder section;
```

Initially

number[i] = 0

choosing[i] = 0

Mutual Exclusion - Exercise 5

- What would be the size of the entries of the number array in this version of the Black-White Bakery algorithm.

Process pi

```
1.  choosing[i] = TRUE;
2.  mycolor[i] = color;
3.  number[i] = 1 + max (number[0], … , number[n-1]);
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. if (mycolor[i] == black) then color = white;
    else color = black;
12. number[i] = 0;
    remainder section;
```

Initially

number[i] = 0

choosing[i] = 0

Mutual Exclusion - Exercise 6

- Is the Black-White Bakery Algorithm correct if the third clause of the second wait statement is removed?

Process pi

```
1.  choosing[i] = TRUE;
2.  mycolor[i] = color;
3.  number[i] = 1 + max (number[j] | 1≤j≤n AND mycolor[j] == 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)
10. critical section;
11. if (mycolor[i] == black) then color = white;
    else color = black;
12. number[i] = 0;
    remainder section;
```

Initially

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

Mutual Exclusion - Exercise 7

- 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[j] == 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

Mutual Exclusion - Exercise 8

The author of the Black-White Bakery algorithm claims the following:

- “The color bit can be easily implemented using n single-writer atomic bits, one for each process.
- To change the color, process i changes the value of the i-th bit. Black corresponds to the case where the number of bits which are set to 1 is even, while white corresponds to the case where the number of bits which are set to 1 is odd."

Provide code for this new version of the algorithm and explain why the new algorithm is correct.

The End - Questions