
CS586: Distributed Computing

Tutorial 2

Professor: Panagiota Fatourou

TA: Eleftherios Kosmas

CSD - October 2011

Mutual Exclusion

Process p0	
1.	while (TRUE) {
2.	await (in == FALSE);
3.	in = TRUE;
4.	critical section;
5.	in = FALSE;
	remainder section;
6.	}

Process p1	
1.	while (TRUE) {
2.	await (in == FALSE);
3.	in = TRUE;
4.	critical section;
5.	in = FALSE;
	remainder section;
6.	}

Initially
in = FALSE

✘ Mutual Exclusion

Mutual Exclusion

Process p0	
1.	<code>while (TRUE) {</code>
2.	<code> flag[0] = TRUE;</code>
3.	<code> await (flag[1] == FALSE);</code>
4.	<code> critical section;</code>
5.	<code> flag[0] = FALSE;</code>
	<code> remainder section;</code>
6.	<code>}</code>

Process p1	
1.	<code>while (TRUE) {</code>
2.	<code> flag[1] = TRUE;</code>
3.	<code> await (flag[0] == FALSE);</code>
4.	<code> critical section;</code>
5.	<code> flag[1] = FALSE;</code>
	<code> remainder section;</code>
6.	<code>}</code>

Initially
<code>flag[0] = FALSE</code>
<code>flag[1] = FALSE</code>

- ✓ Mutual Exclusion
- ✗ Deadlock

Mutual Exclusion

```
Process p0
1. while (TRUE) {
2.     await (turn == 0);
3.     critical section;
4.     turn = 1;
   remainder section;
5. }
```

```
Process p1
1. while (TRUE) {
2.     await (turn == 1);
3.     critical section;
4.     turn = 0;
   remainder section;
5. }
```

Initially
turn = 0

- ✓ Mutual Exclusion
- ✗ Deadlock

Mutual Exclusion - Peterson's Algorithm

Process p0

```
1. while (TRUE) {  
2.   flag[0] = TRUE;  
3.   turn = 1;  
4.   await (turn == 0 OR flag[1] == FALSE);  
5.   critical section;  
6.   flag[0] = FALSE;  
   remainder section;  
7. }
```

Process p1

```
1. while (TRUE) {  
2.   flag[1] = TRUE;  
3.   turn = 0;  
4.   await (turn == 1 OR flag[0] == FALSE);  
5.   critical section;  
6.   flag[1] = FALSE;  
   remainder section;  
7. }
```

Initially

```
flag[0] = FALSE  
flag[1] = FALSE  
turn = 0
```

Mutual Exclusion - Exercise 1

Process p0

```
1. while (TRUE) {  
2.   flag[0] = TRUE;  
3.   turn = 0;  
4.   await (turn == 0 OR flag[1] == FALSE);  
5.   critical section;  
6.   flag[0] = FALSE;  
7.   turn = 1;  
   remainder section;  
8. }
```

Process p1

```
1. while (TRUE) {  
2.   flag[1] = TRUE;  
3.   turn = 1;  
4.   await (turn == 1 OR flag[0] == FALSE);  
5.   critical section;  
6.   flag[1] = FALSE;  
7.   turn = 0;  
   remainder section;  
8. }
```

- Ensures mutual exclusion?
 - if yes, prove it
 - if not, present a counter example

Initially

```
flag[0] = FALSE  
flag[1] = FALSE  
turn = 0
```

Mutual Exclusion - Exercise 2

Process p0

```
1. while (TRUE) {
2.     flag[0] = TRUE;
3.     while (turn == 1) {
4.         await (flag[1] == FALSE);
5.         turn = 0;
6.     }
7.     critical section;
8.     flag[0] = FALSE;
   remainder section;
9. }
```

Process p1

```
1. while (TRUE) {
2.     flag[1] = TRUE;
3.     while (turn == 0) {
4.         await (flag[0] == FALSE);
5.         turn = 1;
6.     }
7.     critical section;
8.     flag[1] = FALSE;
   remainder section;
9. }
```

- Ensures mutual exclusion?
 - if yes, prove it
 - if not, present a counter example

Initially

```
flag[0] = FALSE
flag[1] = FALSE
turn = 0
```

Mutual Exclusion - Exercise 3

```
Process p0
1.  flag[0] = TRUE
2.  while (flag[1] == TRUE) {
3.      if (turn == 1) {
4.          flag[0] = FALSE;
5.          await (turn == 0);
6.          flag[0] = TRUE;
7.      }
8.  }
9.  critical section;
10. turn = 1;
11. flag[0] = FALSE;
    remainder section;
```

```
Process p1
1.  flag[1] = TRUE
2.  while (flag[0] == TRUE) {
3.      if (turn == 0) {
4.          flag[1] = FALSE;
5.          await (turn == 1);
6.          flag[1] = TRUE;
7.      }
8.  }
9.  critical section;
10. turn = 0;
11. flag[1] = FALSE;
    remainder section;
```

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

Initially
flag[i] = FALSE
turn = 0

Mutual Exclusion - Exercise 4

```
Process p0
1. while (TRUE) {
2.   want[0] = 1;
3.   await (want[1] = 0);
4.   critical section;
5.   want[0] = 0;
   remainder section;
6. }
```

```
Process p1
7. while (TRUE) {
8.   want[1] = 0;
9.   await (want[0] = 0);
10.  want[1] = 1;
11.  if (want[0] == 1) then
12.    goto line 7;
13.  critical section;
14.  want[1] = 0;
   remainder section;
15. }
```

```
Initially
want[0] = 0
want[1] = 0
```

- Ensures no deadlock and no lockout?
 - if yes, prove it
 - if not, present a counter example

The End - Questions

