1 Introduction

In this lecture we will cover properties of the Bakery and Peterson algorithms, while learning proof techniques for distributed systems. Specifically we will prove that:

- The Bakery Algorithm satisfies no starvation (theorem 1).
- The two processor Peterson Algorithm satisfies mutual exclusion (theorem 3).
- The two processor Peterson Algorithm satisfies no deadlock (theorem 4).

2 Bakery Algorithm

2.1 No Starvation

Theorem 1 The Bakery Algorithm satisfies no starvation.

Proof: From Lecture 3, we know the Bakery Algorithm satisfies mutual exclusion.

Suppose the Bakery Algorithm does not satisfy starvation, and that some processors are starved. Let $P$ be the set of these starved processors. Let $p_i \in P$ be the processor with the lowest ticket, $NUM[i]$, that is starved. Eventually all processors $p_j \in P - \{p_i\}$ with equal or lower tickets $NUM[j]$ will enter and exit the critical section $CS$.

Let this be time $t$. Since all processors that choose tickets (by being in $ENTRY$) after $p_i$ will choose higher tickets, $p_i$ will have the lowest ticket at all times $t'$ after time $t$. Therefore by Lemma 1, no other processor may enter the $CS$ after time $t$.

Since the number of processors is finite (equal to $n \in \mathbb{N}$), there exists a time $t' \geq t$ when no processor enters $ENTRY$. Therefore, there is a time $t'' \geq t'$ after which no processor sets $CHOOSE[j] = 1$. Thus after time $t''$, $p_i$ should pass through line 6 and line 7. Therefore, $p_i$ enters $CS$. ■

3 Peterson’s Algorithm

Peterson’s Algorithm is a two processor algorithm. We will extend it in the future.
3.1 Initial Algorithms

Examine the following two algorithms, and let \( \text{want}_0 = \text{want}_1 = 0 \) initially:

1: procedure \( P_{\text{Initial}0} \)  
2: \(<\text{ENTRY}> \text{want}_0 := 1 \)  
3: \( \text{wait until}(\text{want}_1 = 0) \)  
4: \(<\text{EXIT}> \text{want}_0 := 0 \)  
5: end procedure

1: procedure \( P_{\text{Initial}1} \)  
2: \(<\text{ENTRY}> \text{L}: \text{want}_1 := 0 \)  
3: \( \text{wait until}(\text{want}_0 = 0) \)  
4: \( \text{want}_1 := 1 \)  
5: if \( \text{want}_0 = 1 \) then  
6: \( \text{goto L} \)  
7: end if  
8: \(<\text{EXIT}> \text{want}_1 := 0 \)  
9: end procedure

Let processor \( P_0 \) run \( P_{\text{Initial}0} \) and let processor \( P_1 \) run \( P_{\text{Initial}1} \). Let time \( t_i < t_j \) when \( i < j \) for \( i, j \in \mathbb{N} \) such that time \( t_i \) occurs chronologically first. There exists an execution where \( P_1 \) is starved:

<table>
<thead>
<tr>
<th>Time</th>
<th>( P_0 )</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 )</td>
<td>2: ( \text{want}_0 := 1 )</td>
<td>2: ( \text{want}_1 := 0 )</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>&lt;CS&gt;</td>
<td>read ( \text{want}_0 = 1 )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>4: ( \text{want}_0 := 0 )</td>
<td>3: ( \text{read want}_0 = 1 )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>&lt;REM&gt;</td>
<td>&lt;REM&gt;</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>2: ( \text{want}_0 := 1 )</td>
<td>3: ( \text{read want}_0 = 1 )</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>&lt;CS&gt;</td>
<td>&lt;CS&gt;</td>
</tr>
<tr>
<td>( t_{6+} )</td>
<td>repeat ( t_2...t_5 )</td>
<td>repeat ( t_2...t_5 )</td>
</tr>
</tbody>
</table>

Furthermore, if both \( P_0 \) and \( P_1 \) used \( P_{\text{Initial}0} \) we end up with deadlock:

<table>
<thead>
<tr>
<th>Time</th>
<th>( P_0 )</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 )</td>
<td>2: ( \text{want}_0 := 1 )</td>
<td>2: ( \text{want}_1 := 1 )</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>3: ( \text{wait until}(\text{want}_1 = 0) )</td>
<td>3: ( \text{wait until}(\text{want}_0 = 0) )</td>
</tr>
</tbody>
</table>

Finally, if both \( P_0 \) and \( P_1 \) used \( P_{\text{Initial}1} \) we also end up with deadlock:

<table>
<thead>
<tr>
<th>Time</th>
<th>( P_0 )</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 )</td>
<td>2: ( \text{want}_0 := 0 )</td>
<td>2: ( \text{want}_1 := 0 )</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>4: ( \text{want}_0 := 1 )</td>
<td>4: ( \text{want}_1 := 1 )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>6: ( \text{goto L} )</td>
<td>6: ( \text{goto L} )</td>
</tr>
</tbody>
</table>
3.2 The Solution Algorithm

The following is the correct Peterson algorithm. It uses a priority bit to switch between the processors:

```
1: procedure Peterson0
2:   <ENTRY> want0 := 0
3:   wait until(want1 = 0) or (priority = 0)
4:   want0 := 1
5:   if priority = 1 then
6:       if want1 = 1 then
7:           goto 2
8:       end if
9:   else
10:      wait until(want1 = 0)
11: end if
12: <CS>
13: <EXIT> priority := 1
14: want0 := 0
<REM>
15: end procedure
```

```
1: procedure Peterson1
2:   <ENTRY> want1 := 0
3:   wait until(want0 = 0) or (priority = 1)
4:   want1 := 1
5:   if priority = 0 then
6:       if want0 = 1 then
7:           goto 2
8:   end if
9: else
10:      wait until(want0 = 0)
11: end if
12: <CS>
13: <EXIT> priority := 0
14: want1 := 0
<REM>
15: end procedure
```

3.3 Properties

**Lemma 2** If $P_i$ is in lines 5 - 12 then want$_i$ = 1. If $P_i$ is in <REM> then want$_i$ = 0.

**Proof:** We only need to look at $P_i$’s code to determine the value of want$_i$. Note that want$_i$ is set to 1 in line 4 and is not modified in line 2 - 12. Also $P_i$ sets want$_i$ to 0 in line 14. So want$_i$ = 0 in <REM>.

**Theorem 3** Peterson1 on processor $P_1$ and Peterson2 on processor $P_2$ provides mutual exclusion between $P_1$ and $P_2$.

**Proof:** Assume without loss of generality that $P_0$ has priority. Suppose for contradiction that $P_0$ and $P_1$ are in <CS> together at time $t$.

Let $w_0$ be the last time that $P_0$ is in line 4 before time $t$. Let $w_1$ be the last time that $P_1$ is in line 4 before time $t$.

Without loss of generality let $w_0 < w_1$. Then by lemma 2, want$_0$ = 1 from time $w_0$ to time $t$. But, $p_1$ reads want$_0$ = 0 in line 6 or line 10 before entering <CS> at time $r_1$, where $w_1 < r_1 < t$.

This implies $w_0 < r_1 < t$, and we have reached a contradiction. Specifically, we claimed that want$_0$ = 1 between $w_0$ and $t$, so want$_0$ = 1 and want$_0$ = 0 at time $r_1$. Thus, Peterson1 on processor $P_1$ and Peterson2 on processor $P_2$ provides mutual exclusion between $P_1$ and $P_2$. ■
Theorem 4  Peterson1 on processor $P_1$ and Peterson2 on processor $P_2$ provides no deadlock between $P_1$ and $P_2$.

Proof: Suppose for contradiction that at all times $t_i$ after time $t$, some processor is in $<\text{ENTRY}>$, but no processor is in $<\text{CS}>$.

There are two cases:

1. Both $P_0$ and $P_1$ are in entry forever. Starting at time $s \geq t$ both $P_0$ and $P_1$ are in the $<\text{ENTRY}>$. Without loss of generality assume priority $= 0$ at time $s$.

   Note that priority never changes after this since priority is changed only on exit from $<\text{CS}>$. After time $s' \geq s$ each processor will either:

   (1) loop in line 3.
   (2) cycle through lines 2 - 7.
   (3) loop in line 10.

   Since priority $= 0$, $P_0$ passes through lines 2 - 5 and loops in line 10 with want$_0 = 1$. Thus cases (1) and (2) are not possible for $P_0$, and want$_0 = 1$ at time $s'$. Since priority $= 0$, $P_1$ was the last processor to exit from $<\text{CS}>$ and modify priority. When $P_1$ enters $<\text{ENTRY}>$ again at time $s'$, want$_0 = 1$, so either it will loop in line 3, else it will loop in case (2). So, when $P_1$ enters $<\text{ENTRY}>$ this time, it can never get to line 10, so case (3) is not possible for $P_1$.

   This implies that $P_1$ cannot pass line 3 after $s'$ since priority $= 0$ and want$_0 = 1$. (Case (2) will put $P_1$ above line 3 where it will be blocked at $s'$.) We have reached our first contradiction as want$_1 = 0$ and $P_0$ will escape the loop in line 10 and enter $<\text{CS}>$.

2. Only $P_0$ is in entry forever after time $t$. Thus after time $t$, want$_1 = 0$ and $P_0$ cannot loop in line 3, lines 2 - 7, or line 10. We have reached our other contradiction as $P_0$ will enter $<\text{CS}>$. Thus Peterson1 on processor $P_1$ and Peterson2 on processor $P_2$ provides no deadlock between $P_1$ and $P_2$.

\hfill \blacksquare