1 Introduction

This lecture covers additional properties of Peterson’s algorithm as well as a procedure to extend Peterson’s Algorithm to accommodate an arbitrary number of processors. Additionally, the concept of configuration similarity is introduced, both with respect to a single processor and a processor set, along with some properties of configuration similarity.

Specifically, we prove:

1. Peterson’s Algorithm satisfies the No Starvation property (Theorem 1)
2. A generalized Peterson’s algorithm satisfies the Mutual Exclusion property (Theorem 3)
3. A generalized Peterson’s algorithm satisfies the No Starvation property (Theorem 4)
4. Two configurations similar to processor $p$ or processor set $P$ remain similar after applying a $p$-only or $P$-only schedule, respectively. (Lemma 5)

2 Peterson’s Algorithm

2.1 Review

See Figure 1 for a review of Peterson’s Algorithm for $side = 0$.

2.2 The No Starvation Property

Theorem 1 Peterson’s Algorithm provides the No Starvation property.

Proof: Consider any execution and two processors, $P_0$ and $P_1$. Assume, for sake of contradiction, that $P_0$ is starved (without loss of generality). Then, starting at some time $t$, $P_0$ is stuck forever in the $\langle Entry \rangle$ section. There are two possibilities to consider:

Case 1: $P_1$ does not enter the $\langle Critical Section \rangle$ after some time $t' > t$.

Starting at $t'$, no processor enters the $\langle Critical Section \rangle$, and $P_0$ is forever in the $\langle Entry \rangle$ section. Therefore, at all times after time $t'$ there is some processor in the $\langle Entry \rangle$ section but
Algorithm 1 Peterson’s Algorithm, Side 0

1: procedure Peterson
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: else wait until want₁ = 0
9: ⟨Critical Section⟩
10: ⟨Exit⟩ priority := 1
11: want₀ := 0
12: ⟨Remainder⟩

no processor in the ⟨Critical Section⟩. This violates the No Deadlock property of Peterson’s Algorithm (Lecture 4, Theorem 4), and thus is a contradiction.

Case 2: P₁ enters the ⟨Critical Section⟩ infinitely often after some time t’ > t.
Consider a time s > t when P₁ sets priority := 0 (line 10). Since P₀ does not enter the ⟨Critical Section⟩ after time t, P₀ must be stuck in one of the following cases:

A P₀ loops in line 3
B P₀ cycles through lines 2-7
C P₀ loops in line 8

At all times s’ > s priority = 0, so P₀ must pass through both lines 3 and 5 and thus cannot be stuck in either case A or B. Therefore P₀ must loop in line 8.
Consider the next time s'' > s' that P₁ enters the ⟨Entry⟩ section. P₁ sets want₁ := 0 and loops in line 3, since want₀ = 1 and priority = 0. Therefore, as P₁ loops from this time s'', P₀ must eventually take its next step. P₀ then reads want₁ = 0 in line 8, and P₀ will enter the ⟨Critical Section⟩, contradicting the original assumption that P₀ is starved.
Either case leads to a contradiction, and thus Peterson’s Algorithm provides the No Starvation property.

2.3 A Bounded Mutual Exclusion Algorithm for n Processes
We now seek to generalize Peterson’s Algorithm to an n-processor algorithm. We construct an n-processor algorithm by repeated application of Peterson’s Algorithm in a tournament tree (a
binary tree with \( n \) leaves, where each node represents an instance of Peterson’s Algorithm and each leaf represents a processor). See Figure 1 on page 3.

Let \( n = 2^k \), giving a complete binary tree with \( 2^{k+1} - 1 \) nodes and \( 2^k \) leaves. Processor \( P_i \) (at leaf \( 2^k + i \)) executes Peterson’s Algorithm at node \( 2^{k-1} + \lfloor i/2 \rfloor \).

We call this new procedure NODE.

**Algorithm 2 NODE**

```plaintext
1: procedure NODE(\( v: \) INT, \( side: \) 0,1)
2: \langle Entry\rangle \quad \text{want}^v_{1-side} := 0
3: \quad \text{wait until} \quad \text{want}^v_{1-side} = 0 \text{ or priority}^v = side
4: \quad \text{want}^v_{side} := 1
5: \quad \text{if} \quad \text{priority}^v = 1 - side \text{ then}
6: \quad \quad \text{if} \quad \text{want}^v_{1-side} = 1 \text{ then}
7: \quad \quad \quad \text{goto} \ 2
8: \quad \quad \text{else} \quad \text{wait until} \quad \text{want}^v_{1-side} = 0
9: \quad \text{if} \ v = 1 \text{ then}
10: \quad \quad \langle Critical Section\rangle
11: \quad \quad \text{else} \ \text{NODE}(\lfloor v/2 \rfloor, \ v \mod 2)
12: \quad \quad \langle Exit\rangle \quad \text{want}^v_{side} := 0
13: \quad \quad \text{priority}^v := 1 - side
14: \quad \langle Remainder\rangle
```

**Remarks**: Informally, each processor (leaf node) runs Peterson’s Algorithm with its sibling. Whoever “wins” then advances up the tournament tree to its parent node, say node \( u \), and runs Peterson’s Algorithm again with the winner that has advanced to \( u \)’s sibling. This procedure continues until a processor advances to the root node, at which point it can finally enter the \langle Critical Section\rangle. When the winning processors exits the \langle Critical Section\rangle, each waiting processor starts to advance up the tree using the same procedure.

Note that it’s ok if the tournament tree is not complete (i.e. we have an odd number of processors); in this case, the processor with no sibling simply “wins” immediately and gets to advance.
The NODE algorithm requires $3(n - 1)$ variables to run, where $n$ is the number of processors. NODE has a better time complexity in the case of no contention than Lamport’s Bakery Algorithm. In Lamport’s Bakery Algorithm, the cost is $O(n)$ (where $n =$ the number of processors) since both the NUM and CHOOSE arrays must be scanned for every processor $P_i$. The NODE algorithm is $O(\log n)$, since each step up the tree continually halves the remaining work.

**Definition 1** Any processor $p$ is said to be in node $v$’s *processor set* if it has taken step 9 of NODE but has not yet taken step 12.

**Lemma 2** For any node $v$ of the tournament tree in the NODE algorithm, at most 1 processor in $v$’s processor set is in $v$’s ⟨Critical Section⟩ at any time $t$.

**Proof:** We prove Lemma 2 by induction on the height of node $v$.

**Base Case:** Consider a node $v$ whose children are leaves. Then the only processors that access $v$ are its two children, and clearly each will have a different side value. So the algorithm run at $v$ is Peterson’s Algorithm. Since Peterson’s Algorithm satisfies the Mutual Exclusion property (Lecture 4, Theorem 3), at most 1 processor is in $v$’s ⟨Critical Section⟩ at any given time.

**Inductive Hypothesis:** Let the Lemma 2 be true for all nodes of height $k$. That is, assume that at most 1 processor is in $v$’s ⟨Critical Section⟩ for any node $v$ at height $k$ at any time $t$.

**Inductive Step:** Consider a node $v$ at height $k + 1$ with nodes $u_l$ and $u_r$ as its left and right children, respectively. By our Inductive Hypothesis we know that at most 1 processor is in node $u_l$’s ⟨Critical Section⟩ and at most 1 processor is in node $u_r$’s ⟨Critical Section⟩ at any time $t$. Therefore, at most 2 processors run $v$’s algorithm with different side values at any given time. Since Peterson’s Algorithm satisfies the Mutual Exclusion property (Lecture 4, Theorem 3), at most 1 processor will ever be in node $v$’s ⟨Critical Section⟩ at any time.

**Theorem 3** The $n$-processor NODE algorithm satisfies the Mutual Exclusion property.

**Proof:** We prove Theorem 3 by applying Lemma 2 to the root node of the tournament tree constructed in the NODE algorithm. All processors in the tree are clearly in the root node’s processor set. Therefore, by Lemma 2, at most 1 processor in the tree is in the ⟨Critical Section⟩ at any time $t$. Thus, the Mutual Exclusion property holds.

**Theorem 4** The $n$-processor NODE algorithm satisfies the No Starvation property.

**Proof:** Suppose, for sake of contradiction, that some processor $P_i$ is starved. Let $u$ be a node such that $P_i$ enters the ⟨Critical Section⟩ of node $u$ at time $t$ but never enters the ⟨Critical Section⟩ of node $v$, the parent node of $u$, after time $t$. Let node $u'$ be the other child of node $v$. By the Lemma 2, we know there is at most 1 processor in the ⟨Critical Section⟩ of both nodes $u$ and $u'$.
Let $P_j$ be the processor in the \langle Critical Section \rangle of node $u'$, if one is active. Since $P_i$ and $P_j$ have different side values, they run Peterson’s Algorithm at node $v$. Since node $v$ provides the No Starvation property in the 2-Processor case by Theorem 1, processor $P_i$ must, at some time, be able to enter the \langle Critical Section \rangle of node $v$. This contradicts our original assumption that $P_i$ is starved, therefore the NODE algorithm satisfies the No Starvation property.

3 A Lower Bound in the Number of Read/Write Variables

Definition 2 Let $C_1$ and $C_2$ be two configurations, and let $p$ be a processor. $C_1$ and $C_2$ are similar to $p$ if

1. $p$ is in the same state in both $C_1$ and $C_2$
2. the values of all shared registers are the same in both $C_1$ and $C_2$.

Similarity between $C_1$ and $C_2$ on processor $p$ is denoted by $C_1 \overset{p}{\sim} C_2$.

Informally, this means that $p$ cannot tell the difference between $C_1$ and $C_2$.

Definition 3 Let $C_1$ and $C_2$ be two configurations, and let $P$ be a set of processor states. $C_1 \overset{P}{\sim} C_2$ if $\forall$ processors $p_i \in P$, $C_1 \overset{p_i}{\sim} C_2$.

Definition 4 Let $\sigma$ be a schedule and $p$ be a processor. $\sigma$ is $p$-only if it only contains events by processor $p$.

Definition 5 Let $\sigma$ be a schedule and $P$ be a set of processors. $\sigma$ is $P$-only if it contains only events by processors in $P$.

Definition 6 Let $\sigma$ be a schedule and $p$ be a processor. $\sigma$ is $p$-free if it contains no events by processor $p$.

Definition 7 Let $\sigma$ be a schedule and $P$ be a set of processor states. $\sigma$ is $P$-free if it contains no events by any processor $p$ in $P$.

Lemma 5 Let $C_1$ and $C_2$ be configurations, and let $P$ be a set of processors. If $C_1 \overset{P}{\sim} C_2$ and $\sigma$ is a $P$-only schedule, then $\sigma(C_1) \overset{P}{\sim} \sigma(C_2)$. 

5
Proof: We prove Lemma 5 by induction on the events of schedule $\sigma$.

**Base Case:** Let $C_1$ and $C_2$ be configurations and let $P$ be a set of processors such that $C_1 \sim^P C_2$. Let $\sigma$ be a schedule with $|\sigma| = 0$. Then clearly $\sigma(C_1) \sim^P \sigma(C_2)$ since $\sigma$ contains no processor events.

**Inductive Hypothesis:** Assume that $\sigma'(C_1) \sim^P \sigma'(C_2)$ for some arbitrary $P$-only schedule $\sigma$.

**Inductive Step:** Let $e$ be an event by some processor in $P$ such that $e$ is applicable to $\sigma(C_1)$ and $\sigma(C_2)$. Let $p$ be the processor in $P$ whose event is $e$, and let $\sigma' = \sigma e$. By our Inductive Hypothesis and the definition of similarity, $\sigma(C_1)$ and $\sigma(C_2)$ have the same shared register values and the same state for all processors in $P$. We prove that $\sigma'(C_1) \sim^P \sigma'(C_2)$.

We must consider two cases, when $e$ is a READ operation and when $e$ is a WRITE operation.

**READ Case:** $p$ reads the same register in both $\sigma(C_1)$ and $\sigma(C_2)$. Since $\sigma(C_1)$ and $\sigma(C_2)$ have the same register values, $p$ reads the same value in both executions, and $p$’s state is the same in both $\sigma'(C_1)$ and $\sigma'(C_2)$. Additionally, the register values do not change, so the register values are the same in both $\sigma'(C_1)$ and $\sigma'(C_2)$.

**WRITE Case:** $\sigma'(C_1) \sim^P \sigma'(C_2)$ by our inductive hypothesis, so $p$ writes value $v_1$ to the register $r$ in both $\sigma'(C_1)$ and $\sigma'(C_2)$. Since $\sigma(C_1)$ and $\sigma(C_2)$ have the same shared register values and $p$ has the same state before $v_1$ is written to register $r$, and since $p$ takes a step and the algorithm is deterministic, $\sigma'(C_1)$ and $\sigma'(C_2)$ have the same shared register values and $p$ has the same state in $\sigma'(C_1)$ and $\sigma'(C_2)$ after $v_1$ is written to $r$.

Clearly, no other processors in $P$ change state, and no other shared registers are modified.

Thus, if $\sigma(C_1) \sim^P \sigma(C_2)$, then $\sigma'(C_1) \sim^P \sigma'(C_2)$, where $\sigma' = \sigma e$ and $e$ is some event by a processor in $P$.

\[\square\]

**Remark:** Note that, informally, this means that if, given a set of processors $P$, a $P$-only schedule $\sigma$, and two configurations $C_1$ and $C_2$ similar for $P$, after any number of events of schedule $\sigma$, or after the application of any number of $P$-only schedules, $C_1$ and $C_2$ will remain similar for $P$ and any processor $p \in P$.

**Definition 8** A configuration $C$ is **quiescent** if all processors are in the ⟨Remainder⟩ section.