1 Introduction

The main objective of this lecture is to prove a lower bound on the number of registers (variables) required for any distributed algorithm that solves the Mutual Exclusion problem. We will work in an asynchronous shared memory model. Recall that Peterson’s Algorithm on \( n \) processors solves Mutual Exclusion by using two arrays, each of size \( n \). So \( 2n \) is an upper bound on the number of registers needed to solve ME. In these notes, we will derive a lower bound of \( n \).

More precisely, we will show that any algorithm that solves ME while also guaranteeing No Deadlock, must use at least \( n \) distinct registers (where \( n \) is the number of processors in the system). We will first prove this for Single-Writer registers, because it provides intuition for why the more complicated argument about the general case is true. Then we will prove the same lower bound for algorithms that use Multiple-Read-Multiple-Write registers.

At a very high level, the reason this lower bound exists is that No Deadlock requires that a processor be allowed to enter the Critical Section if it is not waiting for anyone else, while Mutual Exclusion requires that only one processor be in the Critical Section at a time. So a problem arises if a processor cannot tell the difference between a configuration in which no other processor wants to enter CS, and one in which another processor wants to enter CS but all traces of its desire to enter CS have been overwritten. Our proof will construct an execution in which processor \( p \) wants to enter CS, but all information it might potentially write to the \( < n \) variables in the system are overwritten by other processors before processor \( q \) has a chance to read what \( p \) might have written. Then processor \( q \) may enter CS without knowing that processor \( p \) might already be in there.

1.1 From last class

In the previous class, we proved the following lemma, which we will use today.

**Lemma 1** Let \( C_1 \) and \( C_2 \) be two configurations, and let \( P \) be a set of processors. If \( C_1 \sim P C_2 \) and \( \sigma \) is a finite \( P \)-only schedule then \( \sigma(C_1) \sim P \sigma(C_2) \).

2 Quiescent configurations

**Definition 1** A configuration \( C \) is quiescent if all processors are in the remainder section.
We now prove a few facts about executions that start from quiescent configurations.

**Lemma 2** Given a quiescent configuration $C$ and a processor $p_j$, there exists a $p_j$-only schedule $\sigma$ such that $p_j$ is in critical section in $\sigma(C)$.

**Proof:** Suppose there is some quiescent configuration $C$ such that processor $p_j$ will not be able to enter CS if it is the only processor that takes steps. Then consider any execution that starts at $C$, and no processor takes steps, ever, except for processor $p_j$. Then $p_j$ will be unable to enter CS. This violates No Deadlock, a contradiction. So, starting from any quiescent configuration, if $p_j$ is the only processor to take steps, eventually $p_j$ must enter CS.

**Lemma 3** Given a quiescent configuration $C$ and a processor $p_i$, in any $p_i$-only schedule $\sigma$, such that $p_i$ is in critical section in $\sigma(C)$, $p_i$ must write into some variable in $\sigma$.

![Figure 1: Proof of Lemma 3](image)

**Proof:** For an image of the flow of states see Figure 1. Start with a configuration $C$ that reaches $D = \sigma(C)$ by a $p_i$-only schedule $\sigma$ such that $p_i$ is in critical section in $D$. Suppose, for contradiction, that $p_i$ does not write in $\sigma$. This implies that $C \not\sim D, j \neq i$. From Lemma 2, there exists a $p_j$-only schedule $\sigma'$ from $C$ resulting in $\sigma'(C) = E$ such that $p_j$ is in critical section in $E$. By Lemma 1, the same schedule $\sigma'$ exists from $D$, resulting in $\sigma'(D) = F$, which puts $p_i$ and $p_j$ both in critical section in $F$. This violates mutual exclusion, a contradiction. So $p_i$ must write to some variable in $\sigma$. 

2
3 The easy case: Single-Writer registers

**Theorem 4** If the shared variables are single writer, at least \( n \) variables are required to provide deadlock-free mutual exclusion for \( n \) processors.

**Proof:** If there are fewer than \( n \) single-writer variables, some processor \( p_i \) never writes. Starting at a quiescent configuration \( C \), consider a \( p_i \)-only schedule \( \sigma \) where \( p_i \) enters critical section. This must exist by Lemma 2. Now, by Lemma 3, \( p_i \) must write in \( \sigma \), a contradiction. 

The intuition behind the proof of Theorem 4 is that if a processor \( p_i \) does not write it can enter critical section without telling anyone. A second processor \( p_j \) trying to enter critical section cannot tell if it is occupied. Now, if \( p_j \) does not enter critical section it violates deadlock, and if \( p_j \) does enter critical section it violates mutual exclusion.

We can extend Theorem 4 to remove the single writer clause. Here, since different processors can write to the same shared variable, not only is it important that a processor write while trying to enter the critical section, but also that its writes are observed by other processors (before they are overwritten by some other processor). The idea of the proof is as follows. Let \( A \) be a deadlock free mutual exclusion algorithm. We show an execution in which all processors are about to write into \( n \) distinct shared variables. Otherwise some of the variables could get overwritten.

4 The general case: MRMW Registers

The MRMW Register case is not as simple as the SW Register case, but the intuition is similar. We allow many processors to write to each variable, but this then permits a processor to overwrite the information written by another processor. We obtained a problem from the previous argument when we allowed a non-writing processor to act, and also allowed another processor to act when it had no way of knowing whether the non-writing processor was acting. This time, we set up an execution in which a processor writes but then what it wrote gets overwritten, so the situation is “the same” as a processor that never wrote at all. In both situations, a different processor enters CS prematurely, because there is no information available that could tell it to wait.

Setting up an argument of this nature requires an ability to time writes precisely. More or less, we want \( p \) to overwrite \( q \) as soon as \( q \) writes. So we need a definition for a processor that is about to write to a variable.

**Definition 2** A processor **covers** a variable in a configuration \( C \) if its next step from \( C \) would be to write to the variable.

**Definition 3** Configuration \( C \) is **P-quiescent** if there exists a reachable quiescent configuration \( D \) such that \( C \sim^P D \).
The idea here is that before a processor can enter critical section it must write into some variable that is not covered.

**Lemma 5** Let $C$ be a reachable configuration that is $p_i$-quiescent for some processor $p_i$. Then there exists a $p_i$-only schedule $\sigma$ such that $p_i$ is in critical section in $\sigma(C)$ and during the execution $(C, \sigma)$, $p_i$ writes into some variable that is not covered by any other processor in $C$.

![Figure 2: Existence of $\sigma$ in Proof of Lemma 5](image)

**Proof:** We first show that $\sigma$ exists.

Since $C$ is $p_i$-quiescent, there exists a quiescent configuration $D$ such that $D \succeq C$. By Lemma 2, there must exist a $p_i$-only schedule $\sigma$ from $D$ to $\sigma(D)$ such that $p_i$ is in critical section in $\sigma(D)$. By Lemma 1, the same schedule $\sigma$ applied to $C$ means that $p_i$ is in critical section in $\sigma(C)$. See Figure 2.

We next show that $p_i$ writes into some variable in execution $(C, \sigma)$ that is not covered in $C$ by any other processor. Suppose not.

Let $W$ be the set of variables covered by at least one processor other than $p_i$ in $C$. Let $P'$ be a set of processors, not including $p_i$, so that each variable in $W$ is covered by exactly one processor in $P'$. (If $p_1$ and $p_2$ both cover a variable $r$ then $p_1$ and $p_2$ would not both be in $P'$.) By Lemma 3, $p_i$ must write into something. For contradiction, we assume $p_i$ only writes into variables in $W$.

For an image of the states see Figure 3. Start with the $p_i$-quiescent configuration $C$ where $P'$ covers $W$. There exists a schedule $\rho$ where one step is taken by each processor in $P'$, so $C$ goes to $M$. Now, let $\tau$ be a schedule from $M$, where each processor in $P'$ goes from entry section, to critical section, to exit section, to remainder section, resulting in a quiescent configuration $Q$. Now, by Lemma 2, for any processor $p_j, j \neq i$, there is a $p_j$-only schedule $\pi$ from $Q$ to $S$ such that $p_j$ is in critical section in $S$.

Now, starting at $C$, consider the $p_i$-only schedule $\sigma$ defined earlier, resulting in $C' = \sigma(C)$. By our assumption, $p_i$ writes to the set of variables $R$, where $R \subseteq W$ and $R \neq \emptyset$. Also, $p_i$
is in critical section in $C'$. Then starting at $C'$, applying schedule $\rho$ results in configuration $M' = \rho(C')$.

Since $\rho$ overwrites any writes by $p_i$ in $\sigma$, the register values in $M$ and $M' = \rho(C')$ are identical. Also, the states of all processors except $p_i$ are identical in the two configurations. Therefore, $M \approx M'$, for all $j \neq i$. We can therefore apply the schedules $\tau$ and $\pi$ from configuration $M'$, resulting in configurations $Q'$ and $S'$. Since $\tau$ and $\pi$ are $p_i$-free, $Q \approx Q'$ and $S \approx S'$, for all $j \neq i$. Since $p_j$ is in critical section in $S$, it is also in critical section in $S'$. But $p_i$ is also in critical section in $S'$, violating mutual exclusion. Therefore, we have a contradiction, and $p_i$ must write to some variable in $\sigma$ not covered in $C$.

**Lemma 6** For all $k$, $1 \leq k \leq n$, and for all reachable quiescent configurations $C$, there exists $D$ reachable from $C$, by a $\{p_0, \ldots, p_{k-1}\}$-only schedule such that $p_0, \ldots, p_{k-1}$ cover $k$ distinct variables in $D$, and $D$ is $\{p_k, \ldots, p_{n-1}\}$-quiescent.

**Proof:** We prove by induction on $k$.

**Base Step $k=1$**
Start with a quiescent configuration $C$. By Lemmas 2 and 3, there is a $p_0$-only schedule $\sigma$ such that $p_i$ is in critical section in $\sigma(C)$ and $p_i$ writes to some variable in $\sigma$. Let $\sigma'$ be the longest prefix of $\sigma$ that does not contain a write and let $D = \sigma'(C)$. Therefore, in configuration $D$, $p_0$ covers variable $x$. Also, since $\sigma$ is $p_0$ only and includes no writes, $D$ is $p_1, \ldots, p_{k-1}$-quiescent. See Figure 4.

**Induction Step** (with a simplifying assumption)
Assume the lemma is true for \( k \geq 1 \) and show for \( k + 1 \). For simplicity we will first assume that in each inductive step (IH) we cover the same \( k \) distinct variables \( W \).

Starting at a quiescent configuration \( C \), by the inductive hypothesis, we can reach a \( \{ p_k, \ldots, p_{n-1} \} \)-quiescent configuration \( C_1 \) where \( \{ p_0, \ldots, p_{k-1} \} \) covers \( W \). From \( C_1 \) go to \( M_1 \) by a schedule \( \rho \) in which the processors \( p_0, \ldots, p_{k-1} \) take one step each. Then from \( M_1 \) go to quiescent configuration \( D_1 \) by a schedule \( \tau \) such that the processors \( p_0, \ldots, p_{k-1} \) go from entry section, to critical section, to exit section, to remainder section. Finally, from \( D_1 \), by the inductive hypothesis, there is a schedule \( \sigma \) that leads to the \( \{ p_k, \ldots, p_{n-1} \} \)-quiescent configuration \( C_2 \) where \( \{ p_0, \ldots, p_{k-1} \} \) covers \( W \) again. In the execution from \( C \) to \( C_2 \), only \( p_0, \ldots, p_{k-1} \) take steps.

We construct another execution starting at \( C_1 \) where \( p_k \) also takes steps, that results in a \( \{ p_{k+1}, \ldots, p_{n-1} \} \)-quiescent configuration where \( k + 1 \) distinct variables are covered. Since \( C_1 \) is \( p_k \)-quiescent, by Lemma 5, there is a \( p_k \)-only schedule \( \alpha \) such that \( p_k \) is in critical section in \( \alpha(C_1) \) and \( p_k \) writes to some variable \( x \) not covered in \( C_1 \), i.e., \( x \notin W \). Let \( \alpha' \) be the longest prefix of \( \alpha \) that contains writes only to \( W \). Therefore, in \( C'_1 = \alpha'(C_1) \), \( p_k \) covers \( x \notin W \). Also, \( p_0, \ldots, p_{k-1} \) cover \( W \) in \( C'_1 \). However, \( C'_1 \) may not be \( \{ p_{k+1}, \ldots, p_{n-1} \} \)-quiescent, as required, since \( p_k \) may write to some variables in \( \alpha' \).

From \( C'_1 \), apply the schedules \( \rho, \tau \) and \( \sigma \), resulting in \( C'_2 \). Since \( \rho \) overwrites any writes by \( p_k \) in \( \alpha' \), and \( p_k \) does not take any more steps, the pairs \( M_1 \) and \( M'_1 \), \( D_1 \) and \( D'_1 \), and \( C_2 \) and \( C'_2 \) are identical except for \( p_k \). Therefore, like \( C_2 \), \( \{ p_0, \ldots, p_{k-1} \} \) covers \( W \) in \( C'_2 \). \( C'_2 \) is \( \{ p_{k+1}, \ldots, p_{n-1} \} \)-quiescent since only \( p_0, \ldots, p_k \) take steps. Also, since \( p_k \) covers \( x \notin W \) in \( C'_2 \), \( k + 1 \) distinct variables are covered. For transitions, see Figure 5.

We have skipped over a long and technical aspect of this proof. In particular, we assumed that all quiescent configurations are “the same,” in the sense that, in every quiescent configuration, \( \{ p_0, \ldots, p_{k-1} \} \) (for whatever value of \( k \)) covers exactly the same \( W \). This need not be the case. The full argument would make use of the fact that the space \( V \) of variables is finite, and would apply the Inductive Hypothesis on the order of \( 2^V \) times, so that the Pigeonhole Principle would guarantee that two of the reached quiescent configurations did indeed share the same covered variables. Then we could use those two configurations, and two applications of the same version of the Inductive Hypothesis, to obtain a full proof of the theorem.
Figure 5: Inductive Step in Proof of Lemma 6