1 Introduction

Previously, we examined 8 types of registers that can simulate one another. The eight types of registers are a combination of the following types:

\[ \{ \text{MRSW, SRSW} \} \times \{ \text{safe, regular} \} \times \{ \text{binary, } k\text{-ary} \} \]

where MRSW stands for Multi-Reader-Single-Writer and SRSW stands for Single-Reader-Single-Writer. In this lecture we will investigate the simulation of atomic registers.

2 Defining Linearizability and Atomicity

Any operations that take finite time can be placed into an execution of operations. Intuitively we can think of these operations taking place in time (from left to right) as follows:

\[
\begin{array}{c}
[\text{Read}] \\
[\text{Write}] \\
t
\end{array}
\]

where [ represents the invocation of an operation \( \text{inv}(op) \) and ] represents the response of an operation \( \text{resp}(op) \). (Here \( op \in \{ \text{Read, Write} \} \)) In this example we can come up with a single time \( t \) between [] when the Read actual occurs.

Definition 1 (Linearizability) An execution is linearizable if there exists a permutation \( \pi \) of all operations in \( \sigma \), such that

1. For each object \( X, \pi|X \) is legal, i.e. \( \pi \) satisfies the semantics of \( X \). \( \pi|X \) is the subsequence of \( \pi \) containing exactly the operations specific to \( X \).

2. If the response of \( op_1 \) precedes invocation of \( op_2 \), then \( op_1 \) appears before \( op_2 \) in \( \pi \).

We also refer to linearizability as atomicity with no semantic difference between the two terms.

There exists an equivalent, alternative definition:
Definition 2 (Linearizability) An execution is linearizable if there exists a mapping \( \text{lin} : \sigma \rightarrow \mathbb{R} \) from the set of all operations \( \text{op} \) in \( \sigma \) to real time, such that

1. For each object \( X, \pi|X \) is legal, where \( \pi \) is the permutation of operations listed in order of the \( \text{lin}(\text{op}) \) function.

2. For each operation \( \text{op} \), \( \text{time}[\text{inv}(\text{op})] \leq \text{lin}(\text{op}) \leq \text{time}[\text{resp}(\text{op})] \).

3 Counter Example for \( k \)-ary Atomicity

Lemma 1 The algorithm for the construction of logical, regular, \( k \)-ary registers out of SRSW binary, regular registers does not create atomic logical \( k \)-ary registers out of SRSW binary atomic registers.

Proof: Assume that we can use the \( k \)-ary, regular, logical register algorithm can create atomic logical \( k \)-ary registers out of SRSW binary atomic registers. The algorithm follows (with atomic registers):

1: procedure \( \text{WRITE}(v) \)
2: write \( X_v = 1 \)
3: for \( i = v - 1 \) to 0
4: write \( X_i = 0 \)
5: ACK
6: end procedure

1: procedure \( \text{READ} \)
2: for \( i = 0 \) to \( k - 1 \)
3: read \( (X_i) \) and wait for return
4: if return \( (1) \)
5: then RETURN \( (i) \)
6: end procedure

Examine the following execution for the constructed logical \( k \)-ary register. First \( \text{Write}_w \) is invoked. Next \( \text{Read}_v \) is invoked. It reads \( \text{Read}_v(x_w = 0) \). Since \( v > w \), \( x_w \) is read before \( x_v \). Next \( x_w \) is written, \( \text{Write}_w(x_w = 1) \) and \( \text{Write}_w \) terminates. Next \( \text{Write}_v \) is invoked. \( x_v \) is set to 1, \( \text{Write}_v(x_v = 1) \). \( \text{Read}_v \) then reads and returns \( v \), \( \text{Read}_v(x_v = 1) \). \( \text{Read}_w \) is then invoked. Since \( v > w \), \( x_v \) was written to before \( x_w \) is cleared by \( \text{Write}_v \) and \( x_w = 1 \) is read, \( \text{Read}_w(x_w = 1) \). Finally \( x_w \) is cleared by the write, \( \text{Write}_v(x_w = 0) \) and \( \text{Write}_v \) terminates. We thus have the following execution:

\[
[ \text{Read}(v) ][ \text{Read}(w) ]
[ \text{Write}(w) ][ \text{Write}(v) ]
\]

where \( \text{Read}(v) \) means the operation returned the value of register \( v \) (i.e. \( x_v = 1 \)) and \( \text{Write}(v) \) means the operation wrote to \( x_v \) (i.e. \( v := 1 \)). From definition 1 we see that \( \text{Read}(v) \) precedes \( \text{Read}(w) \) in \( \pi \) and \( \text{Write}(w) \) precedes \( \text{Write}(v) \) and \( \text{Read}(w) \) in \( \pi \). The only possible permutation for the execution is \( \text{Write}(w), \text{Read}(v), \text{Write}(v), \text{Read}(w) \). However, there does not exist \( \pi|X \) that is legal, a contradiction. Thus the algorithm is not linearizable.
4 k-ary Atomic Logical Registers using Binary SRSW registers

We now take the SRSW binary atomic registers and show that we can construct a SRSW k-ary atomic algorithm. Examine the following algorithms for writing and reading. The write algorithm remains the same as in lecture 13:

1: procedure \textbf{Write}(v)
2: \hspace{1em} write \(x_v = 1\)
3: \hspace{1em} for \(i = v - 1\) down to 0 do
4: \hspace{2em} write \(x_i = 0\)
5: \hspace{1em} end for
6: \hspace{1em} ACK
7: end procedure

The following read algorithm has been changed from the one presented in lecture 13:

1: procedure \textbf{Read}(v)
2: \hspace{1em} \(i = 0\)
3: \hspace{1em} while \(i < k\) and \(x_i \neq 1\) do
4: \hspace{2em} \(i := i + 1\)
5: \hspace{1em} end while
6: \hspace{1em} \(v := i\)
7: \hspace{1em} \(i := i - 1\)
8: \hspace{1em} while \(i \geq 0\) do
9: \hspace{2em} if \(x_i = 1\) then
10: \hspace{3em} \(v := i\)
11: \hspace{2em} end if
12: \hspace{2em} \(i := i - 1\)
13: \hspace{1em} end while
14: \hspace{1em} RETURN\((v)\)
15: end procedure

We henceforth refer to the above algorithms as the k-ary SRSW Atomic Algorithm.

5 Linearizability of the SRSW Atomic Algorithm

\textbf{Theorem 2} The k-ary SRSW Atomic Algorithm is linearizable. i.e. There exists \(\pi\) that is a linearization of \(\alpha\).

\textbf{Proof:} Fix some admissible execution \(\alpha\). There exists linearization points for the physical operations. We define the “current value” of a configuration to be the lowest index \(k\) for which \(x_k = 1\) and \(x_{k-1} = x_{k-2} = \ldots = x_0 = 0\). The values of the physical registers is based on linearization points of physical operations.

We define a linearization of the logical operations \(\pi\).
1. Order each **Write** in $\pi$ by their actual order in $\alpha$. Since the algorithm is in SW, no writes overlap, and this is well-defined.

2. Consider each **Read** in the order they occur in $\alpha$. Since the algorithm is in SR, no reads overlap, and this is well-defined. We say that **Read** $r$ reads from **Write** $w$ if $R$ returns $v$ and $w$ contains the last physical write to $x_v = 1$ that precedes $r$’s only physical read of $x_v = 1$.

Thus, place each **Read** $r$ immediately before the **Write** following the **Write** that $r$ reads from in $\pi$. We now need to prove that $\pi$ is a linearization of $\alpha$.

By construction, $\pi$ already satisfies the semantics, since we place a **Read** after the **Write** it reads from but before the next **Write**.

We need to show that the order of operations is maintained in $\pi$ by examining the 4 cases below:

1. A **Write** following a **Write** in $\alpha$: A **Write** following another **Write** is preserved by construction.

2. A **Write** following a **Read** in $\alpha$: If a **Read** $r$ precedes a **Write** $w$ in $\alpha$, then clearly $r$ reads from an earlier write, and thus will be placed earlier than $w$ in $\pi$.

**Lemma 3** Consider two values $u$ and $v$ where $u < v$. If **Read** $r$ returns $v$ and if $r$’s read of $x_u$ during the upward scan reads from a physical write contained in $w$, then $r$ does not read from any **Write** that precedes $w$.

**Proof**: Suppose in contradiction that $r$ reads from a **Write** that precedes $w$, named $w_1$.

\[
\begin{align*}
\left[ \begin{array}{c}
\hline
x_u = 0 \\
\hline
\end{array} \right] & \quad \left[ \begin{array}{c}
x_v - 1 = 0 \\
\hline
x_v = 1 \\
\hline
\end{array} \right] \\
\left[ \begin{array}{c}
\hline
x_v = 1 \\
\hline
\end{array} \right] & \quad \left[ \begin{array}{c}
x_u = 0 \\
\hline
W_1 \\
\hline
\end{array} \right] & \quad \left[ \begin{array}{c}
x_v - 1 = 0 \\
\hline
\end{array} \right]
\end{align*}
\]

Let $v_1$ be the value written by $w_1$. Notice that $u < v$. Also, $v_1 < v$ since otherwise $w_1$ would overwrite $w$’s write to $x_v$ before $r$ could read $x_0$, a contradiction since $r$ reads $x_v = 1$.

Thus in $r$’s upward scan, $r$ reads $x_u = 0$ then $x_{v_1} = 0$, and then $x_v = 1$. Thus there must be another **Write** $w_2$ that writes $x_{v_1} = 0$ before $R$ reads $x_{v_1} = 0$. It follows that $w_2$ writes to some $v_2$ where $v > v_2 > v_1$. By the same argument above, there must be a **Write** $w_3$ that writes $x_{v_2} = 0$ and so on for $w_n, n \in N$. We have thus reached a contradiction since there are only a finite number of physical registers $x_i$ where $u < i < v$, and thus by the pigeon hole principle we will have a write $v_n \not< v$.

3. A **Read** $r$ following a **Write** $w$ in $\alpha$:
4. A Read $r_2$ follows a Read $r_1$ in $\pi$:

Suppose for contradiction that $r_1$ follows $r_2$ in $\pi$. This implies $r_1$ reads $v_1$ from Write $w_1$ and $r_2$ reads $v_2$ from Write $w_2$ where $w_1$ follows $w_2$. Examine the case where $v_1 = v_2$. $w_2$ writes $x_{v_2} = 1$ followed by $w_1$ writing $x_{v_1} = 1$. Since this is followed by $r_2$’s last read of $x_{v_1}$, $r_2$ cannot read the value written by $w_2$, as $w_1$ overwrote it.

Next examine the case where $v_1 > v_2$. Since $r_2$ reads from $w_2$, no write to $x_{v_2}$ is linearized between $w_2$’s write of $x_{v_2} = 1$ and $r_2$’s read of $x_{v_2} = 1$. Since $r_1$ reads from $w_1$, $w_1$’s write $x_{v_1} = 1$ precedes $r_1$’s read of $x_{v_1} = 1$ and ends after $r_2$ reads $x_{v_2} = 1$. However, $r_1$’s read of $x_{v_2}$ during downward scan would return 1, not 0, so $r_1$ should return $v_2$, a contradiction.

Finally examine the case where $v_1 < v_2$. Using lemma 3, $r_2$ cannot read any write before $w_1$. Since $r_1$ reads from $w_1$, $w_1$’s write of $x_{v_1}$’s write of $x_{v_1} = 1$ precedes $r_1$’s read of $x_{v} = 1$. Since $r_2$ returns $v_2 > v_1$, $r_2$’s first (from the upward scan) read of $v_1 = 0$. So, there must be a later Write after $w_1$ containing a write $x_{v_1} = 0$ that $r_2$ reads from. Now, by lemma 3, $r_2$ cannot read (Read) from an earlier Write, so $r_2$ cannot read from $w_2$, a contradiction.

Thus $r_2$ follows $r_1$ in $\pi$. ■