Lecture 16: Shared Registers

CS 539 / ECE 526
Distributed Algorithms
Outline

• Types of Shared Registers

• Algorithms

  – SRSW Boolean Safe → SRSW Boolean Regular

  – SRSW Regular → SRSW Atomic

  – SRSW → MRSW

  – MRSW → MRMW
Types of Shared Registers

- Boolean vs. multi-value

- Single vs. multiple reader (SR, MR)

- Single vs. multiple writer (SW, MW)
  - SRSW: reader is different from writer (otherwise, not distributed)

- Operations (read/write) can overlap
Shared Registers

• If a read does not overlap with a write, return the most recent written value

• If a read overlaps with one or more writes
  – “Safe”: can return any value
  – Regular: return the initial value or one of the written values
  – Atomic: provides an illusion that each op happens at some instant [start, end]

• Operations never overlap
Types of Shared Registers

• Discussion: is safe register too weak?
  – (Note: a “safe” register is very unsafe)

• Without special treatment, a multi-valued register is only safe
  – Old value: 011000
  – Transient: 001000
  – Transient: 001100
  – New value: 001110
Types of Shared Registers

• Discussion: is safe register too weak?
  – (Note: a “safe” register is very unsafe)

• Without special treatment, a multi-valued register is only safe
  – Old value: 011000 011000
  – Transient: 001000 011010
  – Transient: 001100 011110
  – New value: 001110 001110
Space of Shared Registers
Connection to DSM

• Last lecture: distributed algorithms to build share memory with linearizability (atomic consistency)
  – Total order (atomic) broadcast
  – ABD algorithm

• Another way to view these algorithm:
  – Atomic broadcast: MRMW atomic register
  – ABD (so far): MRSW atomic register
Outline

• Types of Shared Registers

• Algorithms
  – SRSW Boolean Safe → Regular
  – SRSW Regular → Atomic
  – SRSW → MRSW
  – MRSW → MRMW
Main Question

• How to implement ”stronger” registers from “weaker” ones?

• Why do we care if we already know how to achieve atomic (“strongest”) registers?
  – Because atomic registers are expensive and real-world systems may implement weaker registers and consistency models
Method 1: Mutual Exclusion

• Topic for next lecture

• Achieve atomicity by preventing overlapping operations altogether

• Downsides:
  – May be “blocked” by other processes for a long time
  – Not crash tolerant
Today: Method 2

- Build "stronger" registers from "weaker" ones tolerating all but one (n-1) crashes

- An algorithm tolerating n-1 crashes is also said to be \textit{wait-free}: no process waits for any other process - Algorithms today will clearly be \textit{wait-free}
SRSW Boolean Safe → Regular

• Use a single Boolean safe register b

• Read(): return b;

• Write(x):
  
  if b != x // use a read to check
  
  b = x; // perform write only if old != new
SRSW Boolean Safe $\rightarrow$ Regular

• Proof of regularity:
  – Suppose a read overlaps with 1 or more writes
  – If all those writes == original value, no actual write occurs $\rightarrow$ read returns original value
  – If one write != original value, OK for a Boolean regular register to return either 0 or 1

• Efficiency:
  – Memory cost: 1x (none)
  – Read cost: 1x (none)
  – Write cost: one read + one write
Outline

• Types of Shared Registers

• Algorithms
  – SRSW Boolean Safe $\rightarrow$ SRSW Boolean Regular
  – SRSW Regular $\rightarrow$ SRSW Atomic
  – SRSW $\rightarrow$ MRSW
  – MRSW $\rightarrow$ MRMW
SRSW Regular $\rightarrow$ Atomic

• First, what is a concrete example that a SRSW regular register fails to be atomic?
  – Recall again that writer $\neq$ reader

  \[
  \begin{bmatrix}
  \text{read}_1 \\
  \text{write} \\
  \text{read}_2
  \end{bmatrix}
  \]

  – $\text{read}_1$ returns new value & $\text{read}_2$ returns old allowed by regular, disallowed by atomic
SRSW Regular $\rightarrow$ Atomic

• Augment the value with a timestamp:
  \[
  \text{reg} = (\text{ts}, \text{val})
  \]

• Writer maintains a timestamp \( \text{ts} \)

Write(\( x \)):

\[
\begin{align*}
\text{ts} &= \text{ts} + 1; \\
\text{reg} &= (\text{ts}, x);
\end{align*}
\]
SRSW Regular \rightarrow Atomic

• Reader (\neq writer) keeps a local copy
  \text{reg\textunderscore local} = (\text{ts}, \text{val}) \text{ in another regular reg}

• \text{Read()}: 
  \begin{align*}
  \text{if \ reg.ts} & > \text{reg\textunderscore local.ts:} \\
  \text{reg\textunderscore local} & = \text{reg}; \\
  \text{return \ reg\textunderscore local.val;}
  \end{align*}
To Prove Atomicity/Linearizability

• Find a sequence $S'$ of operations s.t.
  
  – Each op is immediately followed by its response
  – Each read returns the preceding write value
  – If op1 ends before op2 starts, then op1 occurs before op2 in $S'$
SRSW Regular $\rightarrow$ Atomic

• Each write comes with a ts; Reader reads from local copy, updates local copy upon newer ts.

• Proof: construct $S'$ by ordering ops by their ts; $W$ before $R$, earlier $R$ before later $R$
  – Each op followed by its response – by construction
  – Each read returns preceding write – by construction
  – Respect real-time order of ops
    • $[ R ]$ \ or \ $[ R ]$ \ ensured by ts of $W$
    • $[ W ]$ \ or \ $[ W ]$
    • $[ R_1 ]$ \ $[ R_2 ]$: later read “sees” earlier read’s ts
SRSW Regular $\rightarrow$ Atomic

- Each write comes with a ts; Reader reads from local copy, updates local copy upon newer ts.

- Efficiency:
  - Memory cost: 2x
  - Read cost: 2 read + 1 write
  - Write cost: 1x
  - (Ignoring the use of wider register $|ts|+|val|$ vs $|val|$ )
Outline

• Types of Shared Registers

• Algorithms
  – SRSW Boolean Safe $\rightarrow$ SRSW Boolean Regular
  – SRSW Regular $\rightarrow$ SRSW Atomic
  – SRSW $\rightarrow$ MRSW
  – MRSW $\rightarrow$ MRMW
SRSW $\rightarrow$ MRSW

- Allocate one SRSW register per reader

- Read(): // by reader i
  
  return Reg[i];

- Write(x):
  
  for each i in [1, n]:
  
  Reg[i] = x;
SRSW → MRSW

- Allocate one SRSW register per reader
- Reader reads own copy
- Writer updates all copies

- Works for safe and regular registers

- Efficiency:
  - Memory cost: n (# of reader)
  - Read cost: 1x (none)
  - Write cost: n
SRSW $\rightarrow$ MRSW

• Allocate one SRSW register per reader
• Reader reads own copy
• Writer updates all copies

• Does not work for atomic registers

\[
\begin{align*}
[ \text{read by } j ] & \quad [ \text{read by } k ] \\
[ j \quad \text{write} \quad k ]
\end{align*}
\]
SRSW Atomic $\rightarrow$ MRSW Atomic

• Lesson: a read needs to ensure later reads see a value that is no older
  – Also the crux in SRSW regular $\rightarrow$ atomic

• In fact, we can prove the theorem below:
  • In a wait-free implementation of MRSW atomic register using SRSW atomic registers, at least one reader must write
SRSW Atomic $\rightarrow$ MRSW Atomic

- One reader must write. Proof:
  - Consider writer $p_w$ and two readers $p_a$, $p_b$
  - Suppose for contradiction no reader writes
  - A write by $p_w$ performs many low-level writes
  - Each of which *visible to* either $p_a$ or $p_b$
  - $p_a$ and $p_b$ read from disjoint SR registers
SRSW Atomic $\rightarrow$ MRSW Atomic

• One reader must write. Proof:
  – Some $w_a$ ($w_b$) causes $p_a$ ($p_b$) to see new value
  – $a \neq b$
    • Because each write is visible to only 1 reader

\[
\begin{align*}
  p_a & \quad \text{(READ = new)} \\
p_b & \quad \text{(READ = new)} \\
p_w & \quad \text{WRITE} \\
  \text{write } w_1 & \quad \text{write } w_2 & \cdots & \quad \text{write } w_j & \cdots & \quad \text{write } w_k
\end{align*}
\]
SRSW Atomic $\rightarrow$ MRSW Atomic

• One reader must write. Proof:
  – Some $w_a$ ($w_b$) causes $p_a$ ($p_b$) to see new value
  – $a \neq b$; WLOG $a < b$
  – New-old inversion after $w_a$ $\rightarrow$ not linearizable
SRSW Atomic $\rightarrow$ MRSW Atomic

- Lesson: a read needs to ensure later reads see a value that is no older
- Allocate $n \times n$ matrix of SRSW atomic
  - $(i, j)$: value reported by reader $i$ to reader $j$
  - The single writer writes to the diagonal
  - A reader reads a column and writes a row (other than diagonal)
SRSW Atomic $\rightarrow$ MRSW Atomic

• Write($y$):

\[
\begin{align*}
\text{ts} &= \text{ts} + 1; \\
\text{for each } i \text{ in } [1, n]: \\
\text{Reg}[i][i] &= (y, \text{ts});
\end{align*}
\]

\[
\begin{array}{cccc}
(x, t) & (x, t) & (x, t) & (x, t) \\
(x, t) & (x, t) & (x, t) & (x, t) \\
(x, t) & (x, t) & (x, t) & (x, t) \\
(x, t) & (x, t) & (x, t) & (x, t)
\end{array}
\quad
\begin{array}{cccc}
(y, t+1) & (x, t) & (x, t) & (x, t) \\
(x, t) & (y, t+1) & (x, t) & (x, t) \\
(x, t) & (x, t) & (y, t+1) & (x, t) \\
(x, t) & (x, t) & (x, t) & (y, t+1)
\end{array}
\]
SRSW Atomic → MRSW Atomic

- **Read()**: // by reader j (j=2 in example)
  
  *read entire column j*

  *write row j with highest ts pair*

  *return the value with highest ts*

```
(y, t+1)  (x, t)  (x, t)  (x, t)
(x, t)    (y, t+1) (x, t)  (x, t)
(x, t)    (x, t)   (x, t)  (x, t)
(x, t)    (x, t)   (x, t)  (x, t)
```
SRSW Atomic → MRSW Atomic

• Read(): // by reader j  (j=3 in example)

  read entire column j

  write row j with highest ts pair

  return the value with highest ts
SRSW Atomic $\rightarrow$ MRSW Atomic

• Proof of atomicity similar:
  – Construct $S'$ by ordering ops by $ts$, put $W$ before $R$, earlier $R$ before later $R$
  – Argue the three conditions for atomicity hold
    • Key step: a read ensures later reads see a value that is no older

• Efficiency:
  – Memory cost: $n^2$
  – Read cost: $n$ reads + $n$ writes
  – Write cost: $n$ writes
Outline

• Types of Shared Registers

• Algorithms
  – SRSW Boolean Safe → SRSW Boolean Regular
  – SRSW Regular → SRSW Atomic
  – SRSW → MRSW
  – MRSW → MRMW
MRSW Atomic → MRMW Atomic

• Allocate one MRSW atomic register per writer
• Augment with ts

Write(x): // by writer i

   Read all registers to find max_ts
   reg[i] = (max_ts+1, x);

Read():

   Read all registers
   Return value with max ts
MRSW Atomic $\rightarrow$ MRMW Atomic

- Can two writes have the same ts?
- Yes! Break ties deterministically using proc ID

Write(x): // by writer i

Read all registers to find max_ts
reg[i] = (max_ts + 1, x);

Read():
Read all registers
Return value with max (ts, writer proc ID)
MRSW Atomic → MRMW Atomic

- Proof: construct $S'$ by ordering all writes by $(ts, \text{writer proc ID})$
- Put each read after the write it reads from, earlier read before later read
- By construction, every op followed by response, and read returns preceding write
- Remains to prove real-time order respected
MRSW Atomic $\rightarrow$ MRMW Atomic

• Remains to prove real-time order respected
  – Case 1: $W_1$ ends before $W_2$ begins
    $\rightarrow W_2$ reads local copy updated by $W_1$ & increments ts
    $\rightarrow$ ts of $W_1 <$ ts of $W_2 \rightarrow W_1$ before $W_2$ in $S'$
  – Case 2: $W$ ends before $R$ begins $\rightarrow R$ sees (ts, WID) of $W$ or higher $\rightarrow R$ after $W$ in $S'$
  – Case 3: $R$ ends before $W$ begins $\rightarrow W$ sees (ts, WID) that $R$ reads from or higher, and then increments ts
    $\rightarrow$ ts of $R <$ ts of $W \rightarrow R$ before $W$ in $S'$
  – Case 4: $R_1$ ends before $R_2$ begins $\rightarrow R_2$ sees (ts, WID) that $R_1$ reads from or higher $\rightarrow R_2$ after $R_1$ in $S'$
Summary

• The strongest register can be built from the weakest with wait-freedom
  – SRSW safe $\rightarrow$ * regular $\rightarrow$ atomic
    * only showed binary
  – SRSW $\rightarrow$ MRSW $\rightarrow$ MRMW

  – … also with high costs