A Comparison of Relativistic Programming and Reader-Writer Locks

Philip W. Howard
phil.w.howard@gmail.com
Observations

• Read concurrency is harmless
• Read-Write concurrency can be problematic
• Write-Write concurrency can be problematic
Read-Write Concurrency

• Prevent it
• Manage it

• Reader-Writer Locks
• Relativistic Programming
# Primitives

<table>
<thead>
<tr>
<th></th>
<th>RWL</th>
<th>RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-read</td>
<td><strong>blocks</strong> waiting for writers to finish.</td>
<td>Doesn't block, doesn't exclude writers</td>
</tr>
<tr>
<td></td>
<td><em>Excludes</em> writers</td>
<td></td>
</tr>
<tr>
<td>end-read</td>
<td>non-blocking</td>
<td>non-blocking</td>
</tr>
<tr>
<td>start-write</td>
<td><strong>blocks</strong> waiting for readers and writers,</td>
<td><strong>blocks</strong> waiting for writers, not readers</td>
</tr>
<tr>
<td></td>
<td><em>excludes</em> readers and writers</td>
<td></td>
</tr>
<tr>
<td>end-write</td>
<td>non-blocking</td>
<td>non-blocking</td>
</tr>
</tbody>
</table>
# Concurrency

<table>
<thead>
<tr>
<th></th>
<th>RWL</th>
<th>RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Read</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Read-Write</td>
<td>Not allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Write-Write</td>
<td>Not allowed</td>
<td>Unspecified</td>
</tr>
</tbody>
</table>
Anatomy of a RWL

```c
void read_lock(lock)
{
    while (any_writers(lock))
    {
        // do nothing
    }
    // what if new writer appears here?
    add_reader(lock);
}
```
Performance of a RWL

Operations/sec

Millions

Threads

- nolock
- rp
- rwlr

June 2, 2011
# Primitives

<table>
<thead>
<tr>
<th></th>
<th>RWL</th>
<th>RP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>start-read</strong></td>
<td>blocks waiting for writers to finish. Excludes writers</td>
<td>Doesn’t block, doesn’t exclude writers</td>
</tr>
<tr>
<td><strong>end-read</strong></td>
<td>non-blocking</td>
<td>non-blocking</td>
</tr>
<tr>
<td><strong>start-write</strong></td>
<td>blocks waiting for readers and writers, excludes readers and writers</td>
<td>blocks waiting for writers, excludes writers, not readers</td>
</tr>
<tr>
<td><strong>end-write</strong></td>
<td>non-blocking</td>
<td>non-blocking</td>
</tr>
</tbody>
</table>
What’s the difference?

- Ordering
- Safety
- Programming
Linked List Insert

Number of visible states?
RP Timing
wait-for-readers
RP Timing same as RWL?
RP Timing same as RWL?

**RP**

**RWLR**

**RWLW**
Delete

• Same as Insert

• Memory reclamation
Move

2^8 possible states
Reducing the number of states

4 possible states

wait-for-readers can reduce this to 3
Properties

• Given:
  • wait-for-readers between each write
  • atomicity of read and write operations

• Then:
  • Reads are totally ordered with respect to writes
  • Reads will not observe partially complete writes
Atomicity of Reads

Add 2:A:B

Add 6:A:B

shortest route A:B
\textit{wait-for-readers} fixes this too

\begin{itemize}
  \item Add 2:A:B
  \item wait-for-readers
  \item Add 6:A:B
  \item shortest route A:B
\end{itemize}
wait-for-readers timing
Summary

• RP allows earliest possible read and write
• RP allows more concurrency
  – more read concurrency
  – read-write concurrency
• RP limits complexity
  – n visible states (instead of $2^n$)
  – prefix property
Questions?

phil.w.howard@gmail.com