COMP 322: Fundamentals of Parallel Programming

Lecture 27: Read-Write Locks, Linearizability

Mack Joyner and Zoran Budimlić
{mjoyner, zoran}@rice.edu

http://comp322.rice.edu
Reading vs Writing

• Recall that the use of synchronization is to protect interfering accesses
  — Concurrent reads of same memory: Not a problem
  — Concurrent writes of same memory: Problem
  — Concurrent read & write of same memory: Problem

So far:
  — If concurrent write/write or read/write might occur, use synchronization to ensure one-thread-at-a-time

But:
  — This is unnecessarily conservative: we could still allow multiple simultaneous readers (as in object-based isolation)

Consider a hashtable with one coarse-grained lock
  — Only one thread can perform operations at a time

But suppose:
  — There are many simultaneous lookup operations and insert operations are rare
interface ReadWriteLock {
    Lock readLock();
    Lock writeLock();
}

• Even though the interface appears to just define a pair of locks, the semantics of the pair of locks is coupled as follows
  — Case 1: a thread has successfully acquired writeLock().lock()
    – No other thread can acquire readLock() or writeLock()
  — Case 2: no thread has acquired writeLock().lock()
    – Multiple threads can acquire readLock()
    – No other thread can acquire writeLock()

• java.util.concurrent.locks.ReadWriteLock interface is implemented by java.util.concurrent.locks.ReadWriteReentrantLock class
Hashtable Example

class Hashtable<K,V> {
    ...
    // coarse-grained, one lock for table
    ReentrantReadWriteLock lk = new ReentrantReadWriteLock();
    V lookup(K key) {
        int bucket = hasher(key);
        lk.readLock().lock(); // only blocks writers
        ... read array[bucket] ...
        lk.readLock().unlock();
    }
    void insert(K key, V val) {
        int bucket = hasher(key);
        lk.writeLock().lock(); // blocks readers and writers
        ... write array[bucket] ...
        lk.writeLock().unlock();
    }
}

Linearizability: Correctness of Concurrent Objects

• A concurrent object is an object that can correctly handle methods invoked concurrently by different tasks or threads
  —e.g., AtomicInteger, ConcurrentHashMap, ConcurrentLinkedQueue, …

• For the discussion of linearizability, we will assume that the body of each method in a concurrent object is itself sequential
  —Assume that methods do not create threads or async tasks
Linearizability: Correctness of Concurrent Objects

- Consider a simple FIFO (First In, First Out) queue as a canonical example of a concurrent object
  - Method `q.enq(o)` inserts object `o` at the tail of the queue
    - Assume that there is unbounded space available for all `enq()` operations to succeed
  - Method `q.deq()` removes and returns the item at the head of the queue.
    - Throws `EmptyException` if the queue is empty.

- Without seeing the implementation of the FIFO queue, we can tell if an execution of calls to `enq()` and `deq()` is correct or not, in a sequential program

- *How can we tell if the execution is correct for a parallel program?*
Linearization: Identifying a sequential order of concurrent method calls

Task T1
isolated-wait/begin  q.deq():x  isolated-end

Task T2
isolated-wait/begin  q.enq(x)  enq  deq  isolated-end  deq() returns x

“Linearizability” -- identify order of enq() and deq() calls that is consistent with sequential execution

Source: [http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt](http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt)
Informal Definition of Linearizability

• Assume that each method call takes effect “instantaneously” at some point in time between its invocation and return.

• An execution (schedule) is linearizable if we can choose one set of instantaneous points that is consistent with a sequential execution in which methods are executed at those points
  • It’s okay if some other set of instantaneous points is not linearizable

• A concurrent object is linearizable if all its executions are linearizable
  • Linearizability is a “black box” test based on the object’s behavior, not its internals
Example 1

Task T1

```
q.enq(x)
```

Source: [http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt](http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt)
Example 1 cont.

Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt
Example 1 cont.

Source: [http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt](http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt)
Example 1 cont.

Task T1
- q.enq(x)
- q.deq():y

Task T2
- q.enq(y)
- q.deq():x

Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt
Example 1: is this execution linearizable?

Task T1: q.enq(x) 
Task T2: q.enq(y) 
q.enq(y) 
q.deq():x 
(1) 
(2) 
(3) 
(4) 
qu.deq():y 
linearizable

Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt
Example 2: is this execution linearizable?

Task T1
- q.enq(x)

Task T2
- q.enq(y)
- q.deq(): y
- q.enq(y)

Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

(not linearizable)
Example 3

Is this execution linearizable? How many possible linearizations does it have?

\[
\begin{align*}
&\text{q.enq(x)} \\
&\text{q.enq(y)} \\
&\text{q.deq():y} \\
&\text{q.deq():x}
\end{align*}
\]

(time)

linearizable
(two possible linearizations)
Example 4: execution of an isolated implementation of FIFO queue $q$

Is this a linearizable execution?

<table>
<thead>
<tr>
<th>Time</th>
<th>Task $A$</th>
<th>Task $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invoke $q$.enq(x)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Work on $q$.enq(x)</td>
<td>Invoke $q$.enq(y)</td>
</tr>
<tr>
<td>2</td>
<td>Work on $q$.enq(x)</td>
<td>Work on $q$.enq(y)</td>
</tr>
<tr>
<td>3</td>
<td>Return from $q$.enq(x)</td>
<td>Return from $q$.enq(y)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Invoke $q$.deq()</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Work on $q$.enq(y)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Work on $q$.enq(y)</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Return from $q$.enq(y)</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Invoke $q$.deq()</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Return $x$ from $q$.deq()</td>
</tr>
</tbody>
</table>

Yes! Can be linearized as “$q$.enq(x) ; $q$.enq(y) ; $q$.deq() ; x”
Linearizability of Concurrent Objects (Summary)

Concurrent object

- A concurrent object is an object that can correctly handle methods invoked in parallel by different tasks or threads
  
  — Examples: Concurrent Queue, AtomicInteger

Linearizability

- Assume that each method call takes effect “instantaneously” at some distinct point in time between its invocation and return.
- An execution is linearizable if we can choose instantaneous points that are consistent with a sequential execution in which methods are executed at those points
- An object is linearizable if all its possible executions are linearizable
Announcements & Reminders

- Quiz #6 is due Wednesday, Mar. 30th at 11:59pm
- Hw #4 is due Friday, Apr. 1st at 11:59pm

Hw #4 myjob.slurm

```bash
#!/bin/bash

#SBATCH --job-name=comp322-hw4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --mem=16000m
#SBATCH --time=01:00:00
#SBATCH --mail-type=ALL
#SBATCH --export=ALL
#SBATCH --partition=commons
#SBATCH --exclusive

cd /home/$USER/hw4-GITID

# TODO: Change path to your hw 4 folder
source /home/mjoyner/comp322/322_setup.sh

mvn -DBoruvkaPerformanceTest test

Try --partition=scavenge
```