Lecture 27: Linearizability

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu
Worksheet #26 Solution: Use of trylock()

Rewrite the transferFunds() method below to use j.u.c. locks with calls to tryLock (see slide 5) instead of synchronized.

Your goal is to write a correct implementation that never deadlocks, unlike the buggy version below (which can deadlock).

Assume that each Account object already contains a reference to a ReentrantLock object dedicated to that object e.g., from.lock() returns the lock for the from object. Sketch your answer using pseudocode.

```
1. public void transferFunds(Account from, Account to, int amount) {
2.     while (true) {
3.         // assume that trylock() does not throw an exception
4.         boolean fromFlag = from.lock.trylock();
5.         if (!fromFlag) continue;
6.         boolean toFlag = to.lock.trylock();
7.         if (!toFlag) {
8.             from.lock.unlock(); continue;
9.         }
10.        try {
11.            from.subtractFromBalance(amount);
12.            to.addToBalance(amount);
13.            break;
14.        } finally {
15.            from.lock.unlock();
16.            to.lock.unlock();
17.        }
18.     }
```


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

```
    isolated-wait/begin  q.deq():x  isolated-end
    
Task T1               deq
    
    isolated-wait/begin  q.enq(x)  isolated-end
    
Task T2               enq
```

“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)

time

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

Task T1

q.enq(x)

Task T2

q.enq(y)

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

Task T1

q.enq(x)

Task T2

q.enq(y)  q.deq():x

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

Task T1

q.enq(x)

q.enq(x)

Task T2

q.enq(y)

q.enq(y)

q.deq():x

q.deq():y

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

Task T1
q.enq(x)
(1)

Task T2
q.enq(y)
(2)
q.enq(y)
(3)
q.deq():x
(4)
q.deq():y

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 2: is this execution linearizable?

Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt
Example 3

Is this execution linearizable? How many possible linearizations does it have?
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>Work on 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.enq(y)</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>
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 for Unit 6 is due Monday, April 12th at 11:59pm
Worksheet #27: Execution of concurrent 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>\text{Invoke } q.\text{enq}(x)</td>
<td>\text{Invoke } q.\text{enq}(y)</td>
</tr>
<tr>
<td>1</td>
<td>\text{Work on } q.\text{enq}(x)</td>
<td>\text{Return from } q.\text{enq}(y)</td>
</tr>
<tr>
<td>2</td>
<td>\text{Work on } q.\text{enq}(x)</td>
<td>\text{Return from } q.\text{enq}(y)</td>
</tr>
<tr>
<td>3</td>
<td>\text{Return from } q.\text{enq}(x)</td>
<td>\text{Invoke } q.\text{deq}()</td>
</tr>
<tr>
<td>4</td>
<td>\text{Return } x \text{ from } q.\text{deq}()</td>
<td>\text{Invoke } q.\text{deq}()</td>
</tr>
<tr>
<td>5</td>
<td>\text{Return } x \text{ from } q.\text{deq}()</td>
<td>\text{Invoke } q.\text{deq}()</td>
</tr>
</tbody>
</table>