Lab 10: Java Locks

Instructor: Vivek Sarkar

Resource Summary

Course wiki: https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Staff Email: comp322-staff@mailman.rice.edu

Clear Login: ssh *your-netid*@ssh.clear.rice.edu and then login with your password

Sugar Login: ssh *your-netid*@sugar.rice.edu and then login with your password

Linux Tutorial visit http://www.rcsg.rice.edu/tutorials/

IMPORTANT: Please refer to the tutorial on Linux and SUGAR from Lab 5, as needed. Also, if you edit files on a PC or laptop, be sure to transfer them to SUGAR before you compile and execute them (otherwise you may compile and execute a stale/old version on SUGAR).

As in past labs, create a text file named `lab_10_written.txt` *in the* `lab_10` *directory, and enter your timings and observations there.*

1 Sorted Linked List Example using Java's Synchronized Methods

NOTE: see slides for Lectures 25 and 26 for a recap of Java's synchronized statement and locking libraries respectively.

Download the `lab10.zip` archive from the course web page. It consist of six files: `SyncList.java`, `ListDriver.java`, `ListCounter.java`, `ListSet.java`, `ListTest.java`, `RWMix.java`. Of these, you only need to focus on `SyncList.java`, which contains a thread-safe implementation of a sorted linked list that supports `contains()`, `add()` and `remove()` methods. The default driver options repeatedly calls these three methods with a distribution that aims for 98% read operations (calls to `contains()`), 1% add operations, and 1% remove operations. Since all three methods are declared as `synchronized` in SyncList.java, all calls will be serialized on a single `SyncList` object.

For this section, your tasks are as follows:

1. Compile all Java files by issuing the command, `javac *.java`.
2. Execute the `SyncList` class with the default driver options for 1, 2, 4, 8 threads, by issuing the following commands:
   ```
   java ListDriver -t 1 -b ListTest -s SyncList
   java ListDriver -t 2 -b ListTest -s SyncList
   java ListDriver -t 4 -b ListTest -s SyncList
   java ListDriver -t 8 -b ListTest -s SyncList
   ```

   In the above command, `-t` is used to specify the number of threads. The `-b` option is used to specify the benchmark used to measure the performance of the list implementation. We currently use the `ListTest.java` benchmark. The `-s` option is used to specify the list implementation.

   Observe the performance reported next to the text "Operations per seconds:". Since this is a throughput metric, a larger value will indicate better performance. How does the performance vary with number of threads? Can you explain why this happens? Write your observations in `lab_10_written.txt`.

2 Use of Coarse-Grained Locking instead of Java's Synchronized Methods

The goal of this section is to replace the use of Java's synchronized method in `SyncList.java` by explicit locking instead. For this section, your tasks are as follows:

1. Make a copy of `SyncList.java` named `CoarseList.java`.
2. Replace two occurrences of "SyncList" by "CoarseList" in `CoarseList.java`.
3. Allocate a single instance of `ReentrantLock` when creating an instance of `CoarseList`. See slides 19 and 20 in Lecture 26 for this step, and the remaining steps below.
4. Replace the three occurrences of "synchronized" by appropriate calls to `lock()` and `unlock()`. Remember to use a try-finally block as follows to ensure that `unlock()` is always called:

   ```
   lock.lock();
   try { ... }
   finally { lock.unlock(); }
   ```

5. Compile all Java files by issuing the command, `javac *.java`.
6. Execute the `CoarseList` class with the default driver options for 1, 2, 4, 8 threads, by issuing the following commands:

   ```
   java ListDriver -t 1 -b ListTest -s CoarseList
   java ListDriver -t 2 -b ListTest -s CoarseList
   java ListDriver -t 4 -b ListTest -s CoarseList
   java ListDriver -t 8 -b ListTest -s CoarseList
   ```

   How does the performance compare with the performance observed for `SyncList`? Write your observations in `lab_10_written.txt`.

3 Use of Read-Write Locks

The goal of this section is to replace the use of a `ReentrantLock` in `CoarseList.java` by a `ReentrantReadWriteLock`, so as to leverage the fact that the majority of the operations (98% by default) are calls to `contains()` which are read-only in nature. For this section, your tasks are as follows:

1. Make a copy of `CoarseList.java` named `CoarseRWList.java`.
2. Replace two occurrences of "CoarseList" by "CoarseRWList" in `CoarseRWList.java`.
3. Replace the instance of `ReentrantLock` by an instance of `ReadWriteReentrantLock`. See slides 26 and 27 in Lecture 26 for this step, and the remaining steps below.
4. Replace the calls to lock() by readLock.lock() or writeLock.lock() where appropriate. Likewise for unlock().
5. Compile all Java files by issuing the command, `javac *.java`.
6. Execute the `CoarseRWList` class with the default driver options for 1, 2, 4, 8 threads, by issuing the following commands:

   ```
   java ListDriver -t 1 -b ListTest -s CoarseRWList
   ```

```
java ListDriver -t 2 -b ListTest -s CoarseRWList
java ListDriver -t 4 -b ListTest -s CoarseRWList
java ListDriver -t 8 -b ListTest -s CoarseRWList
```

How does the performance compare with the performance observed for `CoarseList`? Write your observations in `lab_10_written.txt`.

7. This example also allows for selection of different fractions of read (combine), add, and remove operations. For practicality, it is important to use the same fraction for add and remove operations (otherwise the list will become grow too large or too small). To see an increased benefit with read-write locks, you can add the "-r 100 -a 0 -d 0" options to the driver program to make the fraction of read operations 100% (an extreme case).

   Now execute the following commands to compare the performance of `CoarseList` and `CoarseRWList` on 8 threads with 100% read operations:

   ```
   java ListDriver -t 8 -b ListTest -r 100 -a 0 -d 0 -s CoarseList
   java ListDriver -t 8 -b ListTest -r 100 -a 0 -d 0 -s CoarseRWList
   ```

   Write your observations in `lab_10_written.txt`.

4 Turning in your lab work

1. *NOTE: there is no quiz for Lab 10.*
2. Check that all the work for today's lab is in the `lab_10` directory. If not, make a copy of any missing files/folders there. It's fine if you include more rather than fewer files — don't worry about cleaning up intermediate/temporary files.
3. Before you leave, create a zip file of your work by changing to the parent directory for `lab_10/` and issuing the following command, "`zip -r lab_10.zip lab_10`".
4. Use the turn-in script to submit the contents of the `lab_10.zip` file as a new `lab_10` directory in your turnin repository as explained in Lab 1. You can always examine the most recent contents of your svn repository by visiting `https://svn.rice.edu/r/comp322/turnin/S13/`*your-netid*.