ACS Programming assignment 2
Resubmission

Anders Holst, wlc376

2021-01-05

This is my resubmission of programming assignment 2. This hand-in contains only new additions, and thus none of the original hand-in.

Please note: The feedback I received was not perfectly unambiguous as to which parts of the assignment I should resubmit, so I have done my best to infer it.

0 Resubmission: Implementation

In my feedback, you noted that `addCopies` should have two-level locking rather than locking the entire database, and that there was a redundant shared lock in `getBooks`. I have fixed both mistakes in my resubmission. See the attached code.

---

1 Resubmission: question 1b

- Feedback given: *Need to analysis each test in detail. Like which tests verifies what anomalies.*

`MyTests.test3()`

This test asserts consistency of editor picking by repeatedly updating editor picks using two different sets of `BookEditorPicks`. An `EditorPicker` updates editor picks `REPETITIONS` number of times, while a `ConsistencyChecker` simultaneously checks for consistency.

`MyTests.test4()`

This test asserts that threads do not deadlock when trying to wait for locks that are already deleted. A `ProducerRemover` first adds a number of copies while a `Consumer` queues for the locks on these books; before they acquire them, the books are removed by the `ProducerRemover`.

---

2 Resubmission: question 2

Feedback (1)

- Feedback given: *Your two-level lock does not require all locks at the beginning of the transaction. In some transaction, you first acquire read locks (top-level) and then after doing some operations, acquire some write locks (bottom-level).*

I do not see any place in my code where this is true. When only a top-level lock is necessary, only the top-level lock is acquired; when bottom-level locks are needed, they are acquired immediately after the top-level lock.

Feedback (2)

- Feedback given: *How about predicate reads*

My protocol solves the problem by not allowing predicate reads.

---

3 Resubmission: question 5

- Feedback given: *Discuss the overhead being paid in SingleLockConcurrentCertainBookStore and TwoLevelLockingConcurrentCertainBookStore.*

In my original answer, I had not focused on the overhead in the locking protocols as compared to using `synchronized` blocks. The single lock implementation only yields higher concurrency when multiple clients are reading at the same time, and none are writing.

At the same time, the two-level lock has a large overhead in that it needs to keep a lock for each individual book in the store. This overhead is redundant when a particular set of transactions does not contain any/many write accesses, but is more than made up for in the much increased concurrency when the set of transactions requires many exclusive locks.
