ECS518U Operating Systems

Lab 9: Read / Write Lock

This exercise is assessed (Assessment: 20/03 or 27/03 during your lab slot only).

Note: You are not to use any of the interfaces and classes from `java.util.concurrent.locks` – i.e. you are to implement the exercise "from scratch".

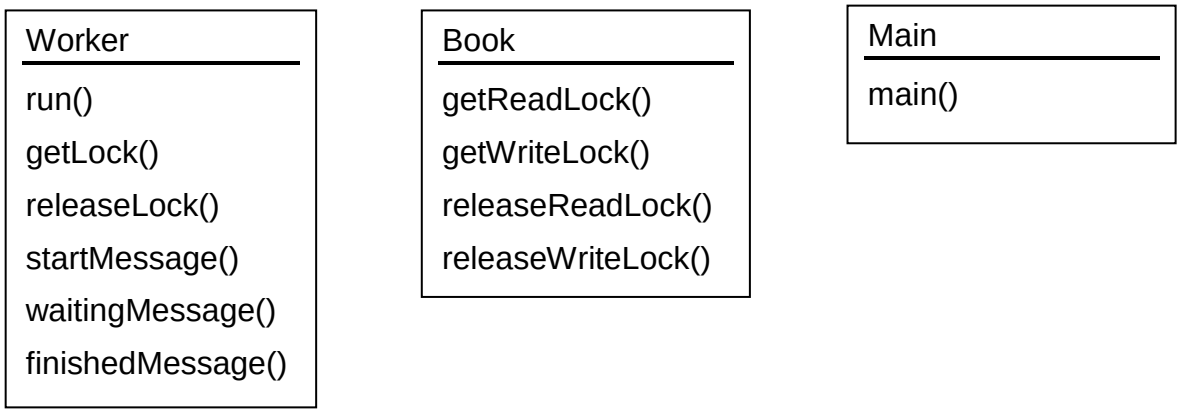
1 Requirements

You are required to implement a read / write lock with the following properties:

1. The lock allows only a single writer to access a shared object at one time. The shared object is said to be 'write locked'.
2. The lock allows multiple readers to access a shared object concurrently. The shared object is said to be 'read locked'.
3. When the shared object is write locked, no other thread can access it.
4. When the shared object is read locked, other threads can read it but not write to it.

2 Design Hints

Three classes are needed: A Main class containing the main() method, a Worker class and a resource (Book) class:



1. The worker class models the readers and writers. The worker objects repeatedly request locks then work (i.e. read or write, modelled by sleeping for a random delay) before releasing the lock.
2. On each iteration the worker chooses randomly to request a read lock or a write lock. The probability of writing (e.g. 0.8) is a constant of the class. You can use a random number between 0-1 to compare with the probability of writing to determine whether a thread will try to read or to write.
3. You can use an 'assert' to check that the correct type of lock is released. However, there is no requirement to check that the lock is released by the correct thread (though this capability would be useful for debugging).
4. The main thread should create a number of workers and then sleep for a while before interrupting the workers.
5. Threads that are interrupted should terminate, releasing any lock held first.

6. The worker threads should print messages to show their activity, as shown in the next section (suggested use of the ..Message() methods in this class).

3 Expected Output

An example output with 4 worker threads and write probability of 80% is given below. Messages are written outside synchronised methods (synchronised blocks should be as short as possible and should avoid including I/O). Be ready to argue the correctness of your implementation.

```
Number of threads = 4
Thread 0 with write lock
probability 0.8
Thread 1 with write lock
probability 0.8
Thread 2 with write lock
probability 0.8
Thread 3 with write lock
probability 0.8
Thread 1 is requesting to write
Thread 0 is requesting to read
Thread 1 is writing
Thread 2 is requesting to write
Thread 3 is requesting to read
Thread 1 has finished writing
Thread 0 is reading
Thread 3 is reading
Thread 1 is requesting to write
Thread 0 has finished reading
Thread 0 is requesting to write
Thread 3 has finished reading
Thread 3 is requesting to read
Thread 1 is writing
Thread 1 has finished writing
Thread 2 is writing
Thread 1 is requesting to read
Thread 2 has finished writing
Thread 2 is requesting to write
Thread 2 is writing
Thread 2 has finished writing
Thread 0 is writing
Thread 2 is requesting to write
Thread 0 has finished writing
Thread 3 is reading
Thread 1 is reading
Thread 0 is requesting to write
Thread 1 has finished reading
Thread 1 is requesting to write
Thread 3 has finished reading
Thread 0 is writing
Thread 3 is requesting to read
Thread 2 is writing
Thread 0 has finished writing
Thread 0 is requesting to write
Thread 2 has finished writing
Thread 1 is writing
Thread 2 is requesting to write
Thread 1 has finished writing
Thread 3 is reading
Thread 1 is requesting to write
Thread 3 has finished reading
Thread 0 is writing
Thread 3 is requesting to read
Thread 2 is writing
Thread 0 has finished writing
Thread 0 is requesting to write
All threads interrupted
Thread 0 completing
Thread 1 completing
Thread 3 completing
Thread 2 completing
```

Please note that the output is for illustration only – in reality it is very likely that you will observe that writer threads get locked just waiting to write – see challenge problem.

4 Challenge Problem (Bonus, required for an A* grade)

A thread that requests a write lock may be made to wait forever, since our threads can request and get read locks after the write lock has been requested. (If you set the read/write probability to 50% you will see that much more that 50% of the time is spent reading.). Modify your implementation to prevent writer starvation. Is this enough to make the lock fair?