Feedback on CO661 A1 for jj333

Final score = 88 /100
Overall comment: Well done!

Task 1 - Server - Max. score 60

Criteria	Comments	Score
Tests	round($\frac{47}{47} \times 20$)	20 / 20
Functionality		
Concurrency and exclusion	This is a good solution, but I think it contains a few bits of redundancy / odd things: I don't think its necessary to create a mutual exclusion block for `getReads` and `isWriter`; just reading a value from memory is atomic, and so it doesn't really matter if this interleaves with other stateful (possibly atomic) interactiosn with these variables. Mutual exclusion is really important when you are updating though, so its good you have included this, for example, in `readUnlock()`. I don't really understand line 199. I puzzled over this line for a long time, and tried to get a bug out of it, but coudln't. But I'm sure there is something strange here! Closing a reader releases the write semaphore, so it keeps increasing the number of available permits for the writer semaphore, even if no writer is using it. This means I can make more writers get theourh the `acquire` on line 179, but then this is guarded by line 178 which acquires all the reader permits... so it seems okay, just odd. It leads me to the question: do you even need the `writerSem`? I'm fairly certain that this is redundant because you get all the standard mutual exclusion with the way you use your reader lock.	17 / 20
Race freedom	Good mechanisms.	5 / 5
Fairness	None of the sempahores are fair, although the locks are.	2 / 5
Description	Fine.	5 / 5
Code quality (comments, format, modularity/abstraction)	Good, but could do with a few more local comments.	4 / 5
Total	Good work.	53/**60**

Task 2 - Client - Max. score 20

Criteria	Comments	Score
Opening		2 / 2
Reading		2 / 2
Writing		3 / 3
Randomness		3 / 3
Logging output		3 / 3
Client/server spawning		4 / 4
Code quality		3 / 3
Total	Good!	20/**20**

Task 3 - Model - Max. score 20

Criteria	Comments	Score
Server and locking	Okay- This idea of making the server into basically just the semaphores and having the client interact with those semaphores doesn't lead to a very clear model of your actualy code, but the semaphores are well presented and I like the use of relabelling.	3 / 6
Client and top-level	Same issue as above in terms of client/server separation, but I'm not going to double penalise this here. One strange thing though is that your client reader has a branch where it needs to acquire the write semaphore, and similarly one that. That doesn't seem right, and doesn't match your code.	2 / 4
Appropriate abstraction	Yes the level of abstraction is about right.	4 / 4
Model exhibits mutual exclusion of read and writes	Yes, I could get the properties to go through with appropriate positioning of some key observable atoms (see appendix for my tweaked version of your model)	6 / 6
Total	Pretty good. Captures the right idea, but I think it could be more clearly set up.	15/**20**

```
* Semaphores (we have a semaphore with 1 lock for writing, and one with 3 locks for read
S = acq.rel.S;
S2 = S | S;
S3 = S2 | S;
* ReadWriteSem is the custom semaphore written for read and write lock management, and i
* I previously had an implementation with further handshakes for calling the functions (
```

```
* But the transitions provided no additional functionality, and it is simpler to underst
ReadWriteSem = S[writeAcq/acq, writeRelease/rel] | S3[readAcq/acq, readRelease/rel];

* Client can either
Client = ClientRead + ClientWrite;
* ClientRead has to obtain the write lock if it is available, and then one read lock.
ClientRead = 'writeAcq.'readAcq.openRfile.ClientCloseRead + 'readAcq.openRfile.ClientClo
* ClientCloseRead releases one read lock, and the write lock if obtained.
ClientCloseRead = done.'readRelease.'writeRelease.Client + done.'readRelease.Client;
* ClientWrite obtains all read locks to prevent simulatenous reads, and obtains the writ
ClientWrite = 'readAcq.'readAcq.'readAcq.'writeAcq.openWfile.ClientCloseWrite;
* ClientCloseWrite releases all locks, to enable further reads and writes.
ClientCloseWrite = done.'writeRelease.'readRelease.'readRelease.'readRelease.Client;

* We spawn 3 clients and 1 server (called ReadWriteSem), but in actual use we can spawn
App = (Client | Client | Client | ReadWriteSem) \ {readAcq, writeAcq, readRelease, write

Start = App;
```