Properties:

CTL:
AG (((writer_1 = s2) -> !(writer_2 = s2)) & ((writer_2 = s2) -> !(writer_1 = s2)))
AG (reader_1_read -> any_writer_wrote)

LTL:

Mutual exclusion: G (((activeReaders > 0) -> !(activeWriters > 0))
& ((activeWriters > 0) -> !(activeReaders >0))
& (activeWriters <= 1))
The last expression satisfies the exclusive write property

Write-first: It is always true that if a reader and writer want to operate on data at the same time, the writer gets preference is the data has been read at least once
G (((waitingReaders > 0) & (waitingWriters > 0) & (read = true)) -> X((activeWriters>0))

Liveness: G ((activeWriters > 0) -> X (activeReaders > 0))

Steps to convert to JML:

Mutual exclusion : The G gets converted to an invariant, since it means that the mutual exclusion property has to hold at all points in the program.
We can also seperate the activeWriters <= 1 property into a seperate invariant for the sake of convenience

((activeReaders>0) -> ! (activeWriters>0)) & ((activeWriters>0) -> ! (activeReaders>0))
≡ (! (activeReaders>0) v ! (activeWriters>0)) & (! (activeWriters>0) v ! (activeReaders>0))
≡ (! (activeReaders > 0) v ! (activeWriters > 0))
≡ ! (activeReaders > 0 & activeWriters > 0)

Exclusive write: activeWriters <= 1 ≡ ! (activeWriters > 1)

The write first property leads to the following JML assertions.

Post C for startRead: The writer first property implies that a read may be performed if and only if the following conditions apply in addition to data>=0:
a) activeWriters = 0
b) waitingWriters = 0
or
a) read = false
Since the satisfaction of all of these conditions means that a read can be performed, the number of activeReaders must go up by 1. Therefore, we get the postcondition:

activeReaders>=1 & ((old(activeWriters) + old(waitingWriters) = 0 ll !(old(read)))&&(data>=0))
=> old(activeReaders) + 1 = activeReaders

Post C for endRead: Once a read is done, since the write first property demands that a writer be given preference after the first time a piece of written data is read. Additionally, since the reader has finished reading, the number of active readers must go down by one. Therefore the postcondition:
old(activeReaders) + 1 = activeReaders & read = true

Post C for startWrite: Mutual exclusion means that only one writer can modify the data at one time. Plus, if someone is reading the data, a write cannot be performed, so when a write occurs, there must neither any active readers, nor any active writers. Therefore, at the end of startWriter, there must be only one active writer. Additionally, for a write to be performed, any data that was written previously must be read at least once (liveness). Therefore the postcondition:

activeWriters = 1 & (old(activeWriters) + old(activeReaders) = 0 & read)
=> old(activeWriters)+1 = activeWriters

Post C for endWrite: At the end of a write, the liveness property implies that the next operation to be performed on the data must be a read. Thus, the endWrite method must ensure that read is set to false. Additionally, since the write is over, there must be no more active Writers. Hence the postcondition:

activeWriters = 0 & read = false