Safety properties - statement

Messages are delivered in order

The safety condition must hold for every possible sequence of inputs, so we may assume that the data in each message is distinct from the data in each other message. Or, the writing application can supply sequence numbers, which we require to be in order,

To prove:

> For each pair of messages sent in each execution, if write(x) appears in an execution before write(y) then read(x) appears in the execution before read(y).

Messages are delivered at most once

Same assumption above, that all messages are either distinct or numbered distinctly.

To prove:

No two distinct read actions in an execution contain the same data value (or data id).

Proof

Add an application Writer that writes from a queue, in order, starting at the front and going to the back, and an application Reader that reads from the SlidingWindowReceiver and puts the data read in a buffer, in the order it was received.

State invariant

One state invariant suffices to prove both safety properties:

> If the Writer queue equals {d_1, d_2, d_3, ... }, then in every state of every execution of the composition of Reader, Writer, SlidingWindowSender, SlidingWindowReceiver, and UnreliableChannel, the following conditions hold:

1) the Reader buffer is a prefix of the Writer buffer, i.e., the Reader buffer is { d_1, d_2, d_3, ... , d_n} for some n.
2) SlidingWindowSender.LastFrameWritten is equal to j in the last previous action of the form write(d_j) (or to 0 if there has been no write); and
3) For each write(d_i) appearing prior to any state s_k in the execution, the sequence number associated with d_i is sequence number i in all buffers in s_k, i.e., s_k.SlidingWindowSender.sendBuf[i] $\in$ { [0,0,0], [d_i,i,0] } and s_k.SlidingWindowReceiver.receiveBuf[i] $\in$ { [0,0,0], [d_i,i,0] } and there is no entry [d_i, k, 0] $\in$ UnreliableChannel.inTransit with $i \neq k$.
4) If Reader.Queue={ d_1, d_2, d_3, ... , d_n} then n=SlidingWindowReceiver.lastFrameRead.
5) If k < SlidingWindowReceiver.lastFrameRead, then read(d_k) is an action in the execution

Discussion:

We use conditions 2 through 5 in the invariant so that we can establish condition 1, which in effect guarantees that messages are delivered in order and without duplicates.

Note that this doesn't say anything about whether messages are delivered!! In general, if no messages are delivered, they can't be delivered out of order.

Proof by induction on the step of the execution:

Induction base:

In the start state, all buffers and queues except the Writer queue are empty, so the invariant holds.

Induction hypothesis:

For all executions, at step m (or in every prefix of length m), the above conditions hold in all states.

Induction step:

Let α_{m+1} be the (m+1)st action and s_{m+1} be the (m+1)st state. From the induction hypothesis, we know that in state s_m the Reader buffer was { d_1, d_2, d_3, ... , d_n } for some prefix of the Writer buffer. Let's consider the possibilities in state s_{m+1}, depending on the action α_m:

write(d_i) – need to show that conditions 2 & 3 still hold:

By the induction hypothesis, we know that SlidingWindowSender.LastFrameWritten is equal to j in the last write(d_j) or to 0 if there is no such write prior to step m.

Case 1. LastFrameWritten=0: Then i is 1, LastFrameWritten is 1, and the sequence number applied is 1. SlidingWindowSender.sendBuf[1] = [d_1, 1, 0].

Case 2. LastFrameWritten>0: Then write(d_{i-1}) was the previous write (by our definition of the Writer) and LastFrameWritten=i-1 (by the induction hypothesis). Thus SlidingWindowSender.sendBuf[i] = [d_i, i, 0] and lastFrameWritten = i.

send([d_j, j, 0], A, B) – need to show that conditions 2 & 3 still hold:

The state of SlidingWindowSender doesn't change, so there's nothing to show. The message [d_j, j, 0] is transferred as is into UnreliableChannel.inTransit.

receive(([d_j, j, 0], B, A) – need to show that conditions 3-5 hold:

Either there's no state change in SlidingWindowReceiver.receiveBuf or the message [d_j, j, 0] is transferred as is into SlidingWindowReceiver.receiveBuf[j].

read(d_k):

Nothing happens unless the sequence number in receiveBuf[lastFrameRead] > 0, in which case, before action read(d_k), the induction hypothesis requires the following to hold:

receiveBuf[lastFrameRead] = $d_{lastFrameRead}$

Reader.queue = { d_1, ..., $d_{lastFrameRead}$ }.

The precondition implies that k be lastFrameRead+1, so that Reader.queue becomes { d_1, ..., $d_{lastFrameRead+1}$ } and lastFrameRead becomes lastFrameRead+1. This proves that 4) and 5) are still true.

duplicate(d_k):

The sequence number associated with the data stays the same, so this does not violate the induction hypothesis.

drop(d_k):

If condition 3 wasn't violated before, removing a message doesn't violate it.