Exercise 1 Fondations of Distributed Systems

Anastasiya Moiseeva
Yannick Martin

Task 1:

Algorithm: Initialization Barrier

Implements:
Initialization Barrier, instance ib.

Uses:
PerfectPointToPointLinks, instance pl.

upon event <ib, Init> do
processes := Π;

upon event <p, Init> do
forall q ∈ processes do
trigger <pl, Send | q, m>;

upon event <pl, Deliver | p, m>
process := process \ {p};
if processes = ∅
trigger <ib, Ready>;

Task 2:

Module: FIFO PerfectPointToPointLinks
Module:
Name PerfectPointToPointLinks, instance pl.

Events:
Request: <pl, Send | q, m>: Request to send message m to process q.

Indication: <pl, Deliver | p, m>: Delivers message m sent by process p.

Properties:
FPL1: Reliable delivery: same as PL1
FPL2: No duplication: same as PL2
FPL3: No creation: same as PL3
FPL4: FIFO Delivery: If some process sends m_1 before the same or another process sends another message m_2, then no correct process delivers m_2 unless it has already delivered m_1.

Algorithm: FIFO Perfect Point To Point Links

Implements:

FIFOPerfectPointToPointLinks, instance fpl.

Uses:

PerfectPointToPointLinks, instance pl.

```
# Initialize the local seq. number and next seq number
upon event <fpl, Init> do
    forall p ∈ Π do
        lsn[p] := 0;
        next[p] := 1;
# Even for sending q, message.
upon event <fpl, Send | q, m> do
        # Increment lsn for the sender(q);
        lsn[q] := lsn[q] + 1;
        # Sending message and lsn;
        trigger <pl, Send | q, lsn[q]>;
# Event to deliver a message (p, sn) to a process p
upon event <pl, Deliver | p, m> do
    pending := pending ∪ {(p, m, sn)};
    # Process pending messages while there are messages with the expected sequence number.
    while exist (q, s, sn') ∈ pending such that sn' = next[q] do
        next[q] := next[q] + 1;
        pending := pending \ {(q, n, sn')};
        trigger <fpl, Deliver | q, n>;
```

Task 3:

Task 3

① ~~In~~

No Loss of Data.

- In my implementation there is an assumption that there is no data loss or corruption in transit. (Reliable delivery; PL1) In real world it may happen.
- TCP handles damaged data (Reliability)
  - Error detection (using checksum)
  - Retransmission

② Bandwidth

- The pl is assumed to have infinite capacity, so there are no bandwidth limitations.
- But in real life there are some limits.

TCP handels this using Flow control:

This is achieved by returning a „window" with the every ACK indicating a range of acceptable seq. numbers beyond the last segment successfully received. The window indicates an allowed # that sender may transmit.

Task 4.)

In the model	Java Concurrent	JavaScript
• Event-driven • Assumed that each process deals with events one at a time in a way that prevents concurrent handling of multiply events.	Not always Java Concurrency will support stat machines model, but we can make architecture that threads will be event-driven and use java.util.concurrent.atomic to make sure that operations on these variables are thread-safe without the need for explicit synchronization and create event queue to make sure we will have FIFO ordering.	• Event-driven • JS has a runtime model based on an event loop, which is responsible for collecting and processing events, and executing sub-tasks Answer: yes