Dr. Annette Bieniusa
M.Sc. Peter Zeller

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Exercise 3: Programming Distributed Systems (SS 2019)

- To get feedback to your solution: Create a new branch in your git repository (e.g. `git checkout -b ex3`). Submit your solution to the programming exercise via your group's repository to the new branch in a folder named "ex3". When your solution is ready, create a merge request in Gitlab and assign Peter Zeller to it. This will allow us to comment on your code and give you feedback.
- There will be no exercise on May 15.
- Prepare task 1 for the exercise on Wednesday, May 22.

1 TLA$^+$ specifications

a) Formalize the following properties using temporal formulas in TLA$^+$:

RB1 (Validity) If a correct process i broadcasts message m, then i eventually delivers the message.

RB3 (No Creation) If a correct process j delivers a message m, then m was broadcast to j by some process i.

RB4 (Agreement) If a message m is delivered by some correct process i, then m is eventually delivered by every correct process j.

URB4 (Uniform Agreement) If a message m is delivered by some process i, then m is eventually delivered by every correct process j.

CB (Causal delivery) No process p delivers a message m' unless p has already delivered every message m such that $m \rightarrow m'$.

For simplicity, we assume that no message is broadcasted more than once. You can use the following variables in your properties:

- *Process*: The set of all processes.
- *correctProcess* $\in$ *SUBSET Process*: The set of correct processes.
- *Message*: The set of all messages that can be sent.
- *broadcasted* $\in$ [*Process* $\rightarrow$ *SUBSET Message*]: For each process the set of messages that are queued to be broadcasted.
- *delivered* $\in$ [*Process* $\rightarrow$ *SUBSET Message*]: For each process the set of messages that have been delivered by the broadcast algorithm.
- *happensBefore* $\in$ [*Message* $\rightarrow$ *SUBSET Message*]: For each message the set of messages that happened before it.

b) (Optional) You can find TLA$^+$ models of the following broadcast protocols in the material for this exercise:

- Best-effort broadcast
- Reliable broadcast
- Causal broadcast (waiting)

Use the TLA$^+$ model checker (TLC) to check these algorithms against your properties.

2 Practical causal broadcast

Final project: This broadcast algorithm is one part of your final project. Therefore, we will not provide solutions for this exercise.

The causal broadcast algorithm from the lecture, which you implemented for the last exercise sheet, has some practical shortcomings, which you should improve on in this exercise:

1. The algorithm reuses the reliable broadcast algorithm. This is a nice and modular solution, however it has some performance issues:
   - Without crashes, the number of messages sent by our reliable broadcast implementation is N^2 if N is the number of processes.
   - The reliable broadcast algorithm keeps a set, which grows with every delivered message.
2. The algorithm assumes the Crash-Fault Model. However, for our final project we will need an algorithm that tolerates crashes and continues to work after a crash and a restart.

Design and implement an optimized algorithm, which does not have these flaws. More precisely you should implement a module named `persistent_causal_broadcast` with the same interface as the broadcast protocols from sheet 2. The algorithm should provide the following guarantees:

Validity If a process i broadcasts message m (the `broadcast` function is called and has returned `ok`), then i eventually delivers the message.

No Duplications No message is delivered more than once.

No Creation If a process j delivers a message m, then m was broadcast to j by some process i.

Agreement If a message m is delivered by some process i, then m is eventually delivered by every process j.

Causal delivery No process p delivers a message m' unless p has already delivered every message m such that $m \rightarrow m'$.

You may assume the following fault model:

- When the broadcast process crashes, it is eventually restarted (e.g. a power-outage that is restored later).
- After a restart, no persistent state is lost (i.e. state written to disk using functions like `disk_log:sync` to ensure persistence)
- For sending messages over the link layer you can assume *No Creation* (every delivered message was sent by some process) and *No Duplication* (no message is delivered more than once).