Homework Assignment 3 (12.5 points)

4DC3, Winter 2019

Due: March 22, 23:59pm

Marking

This assignment accounts for 12.5% of your final mark.

Instructions

Please submit your solution on Avenue as a single ZIP before the deadline. Only the last submitted version will be considered for grading. You are encouraged to use the `mix` tool to create a project directory (see the tutorial slides). Please see the course outline for the policy on late submissions.

Problem 1 (Simulating Totally Ordered Broadcast)**.** The Erlang/Elixir virtual machine provides facilities for asynchronous point-to-point message passing. You are asked to simulate broadcast functionality on top of point-to-point channels to achieve more advanced quality of service requirements.

- Write a module `TOBroadcast` that implements *totally-ordered broadcast*. Your implementation should follow the pseudo code of the TO-Broadcast simulation described in detail in the course lectures slides on Avenue (see `Content` → Broadcast Simulations). This includes keeping messages that are not yet ready to be to-bcast-received in a `pending` set, keeping track of timestamp estimates in `ts`, and sending `ts-up` messages when needed.
- The module should provide (at least) the following interface functions:

  `start(name,participants)`: Spawns a new process p that executes a function `run`. Process p keeps track of its name, the PID of the parent process, and the list of the names of the `participants` in its local state. Returns p's process ID.

  `bc_send(msg,origin)`: Initiates the broadcast of message `msg` by the process p_i with name `origin` to all participating processes. This will simply send a message `{:input,:bc_initiate,msg}` to p_i to simulate receiving an input from the above layer.
- The following messages should be handled/sent in the function `run`.
  - `{:input,:bc_initiate,msg}`: described above.
  - `{:ts_up,t,origin_name}`: a timestamp update message with new timestamp `t`, originating from process `origin_name`.
  - `{:bc_msg,msg,t,origin_name}`: a broadcast message carrying the desired message and the timestamp `t` originating from process `origin_name`. Note that Elixir/Erlang guarantees FIFO point-to-point channels.
  - When a message `msg` is ready to be to-bcast-received at a process p_i, where `msg` was sent by a process with name `origin`, then p_i should send a notification message of the form `{:output,:to_bcast_rcvd,msg,origin}` to its parent process. Here we assume that all participating processes are spawned by the same parent process.

- **Grading:**
  1. Up to 40% if the output messages are sent correctly, under the assumption that all broadcasts are performed by the same process.
  2. Up to 100% if the implementation follows the pseudo code and satisfies the properties of totally-ordered broadcast.

[Points: 10]

Problem 2. Consider a broadcast service B that has the following guarantees: If messages m_1 and m_2 are received by a non-faulty process p_i, then

(1) every other non-faulty process p_j is guaranteed to also receive m_1 and m_2, and
(2) p_i and p_j receive m_1 and m_2 in the same order.

Provide an argument why it is impossible to implement B in the standard asynchronous point-to-point message passing system, when up to f out of n processes can fail by crashing, where $f \in [1, n-1]$.

[Points: 2.5]