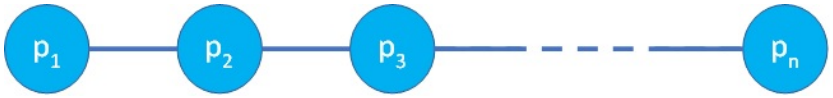
Exercise week 2 - October 6th, 2022 - Exercise 5. Commented Solution

Let us consider a distributed system composed of N processes $p_1, p_2, \ldots p_n$ each one having a unique integer identifier. Processes are arranged in line topology as in the following figure



Let us assume that there are no failures in the system (i.e., processes are always correct) and that topology links are implemented through perfect point-to-point links.

Write the pseudo-code of a distributed algorithm that is able to build the abstraction of a perfect point-to-point link between any pair of processes (also between those that are not directly connected).

General Comments

The text says that processes have access to a perfect point-to-point link primitive, let us refer to this component with $\mathcal{PL}_l$. This primitive allows every process to exchange messages with at most two other processes, the neighbor processes in the line.

Let us recall the events and properties of a perfect point-to-point primitive.

Module:

Name: PerfectPointToPointLinks, **instance** *pl*.

Events:

Request: $\langle$ *pl*, *Send* | q, m $\rangle$: Requests to send message m to process q.

Indication: $\langle$ *pl*, *Deliver* | p, m $\rangle$: Delivers message m sent by process p.

Properties:

PL1: *Reliable delivery:* If a correct process p sends a message m to a correct process q, then q eventually delivers m.

PL2: *No duplication:* No message is delivered by a process more than once.

PL3: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.

Every process can thus trigger *Send* events and catch *Deliver* events of the $\mathcal{PL}_l$ component.

The exercise requests to define a protocol a link abstraction, let us name it $\mathcal{PL}_g$, that implements a perfect point-to-point link between all pairs of processes. In the initial setting, every process can exchange messages only with at most two processes, namely the neighbor processes on the line. It follows that if two processes are not linked by $\mathcal{PL}_l$, such as processes p_1 and p_4, they cannot exchange messages in the current setting. The target of the exercise is the definition of a protocol that implements a communication primitive guaranteeing all the properties of a perfect point-to-point link between all pairs of processes.

Protocol Idea

Each process is associated with an integer identifier and it is placed in a topology (the line) such that all the processes with a lower identifier are placed on one side (from the prospective of a selected process) and all processes with a higher identifier are located on the other side. Furthermore, the processes are placed in order on the line with respect to their identifiers. It follows that if process p_1 wants to communicate with process p_4, it can send its messages to process p_2, that will relay such messages to p_3, which finally forwards them to p_4. More in detail, if the destination of a message is a process with a higher identifier with respect to a selected process, then it is sufficient to forward the message to the neighbors with greater ID till reaching the destination process, and vice-versa for a destination with a lower identifier.

Pseudo-code

Algorithm 1 $\mathcal{PL}_g$

```
1: procedure INIT
2: |  ID ← unique integer identifier

3: upon event ⟨PL_g, Send | dest, m⟩ do
4: |  if dest > ID then
5: |  |  trigger ⟨PL_l, Send | ID + 1, ⟨ID, dest, m⟩⟩
6: |  else if dest < ID then
7: |  |  trigger ⟨PL_l, Send | ID − 1, ⟨ID, dest, m⟩⟩⟩
8: |  else
9: |  |  trigger ⟨PL_l, Send | ID, ⟨ID, dest, m⟩⟩⟩

10: upon event ⟨PL_l, Deliver | q, ⟨source, dest, m⟩⟩ do
11: |  if dest == ID then
12: |  |  trigger ⟨PL_g, Deliver | source, m⟩
13: |  else if dest > ID then
14: |  |  trigger ⟨PL_l, Send | ID + 1, ⟨source, dest, m⟩⟩
15: |  else
16: |  |  trigger ⟨PL_l, Send | ID − 1, ⟨source, dest, m⟩⟩
```

Pseudo-code Comments

The first upon block (lines 3-9) rules how a process must act when the *Send* operation of the global perfect point-to-point primitive we are developing, $\mathcal{PL}_g$, is triggered. It is a tiny procedure that simply starts the propagation of a new message, $\langle source, dest, m\rangle$, over the proper link, $\mathcal{PL}_l$. Again, a content m (let use this word to distinguish m from $\langle source, dest, m\rangle$) targeted to a process with an higher identifier (*dest*) is reachable (on the line) by relaying the content to the neighbor with higher ID and vice-versa.

The second upon block rules how a process that has received a message $\langle source, dest, m\rangle$ from the primitive $\mathcal{PL}_l$ (namely the local perfect point-to-point link) must act. This procedure compares the *dest* field contained inside the received message with the process identifier. If the process is the destination of a content, then it triggers the *Deliver* operation of the primitive we are developing (content m was targeted to this process). Otherwise, it continues the propagation of the message $\langle source, dest, m\rangle$ over the proper perfect point-to-point link. Notice that we need to forward the information about the source ID *source* and the destination id *dest* insider the message, otherwise a process (not linked with the source process) receiving a content m from $\mathcal{PL}_l$ cannot assert whether the content is targeted to it-self and which process was its source.

Informal Correctness Proofs

We briefly check the correctness of the provided pseudo-code. The $\mathcal{PL}_g$ primitive we defined must handle two events, $\langle \mathcal{PL}_g, Send \mid dest, m \rangle$ and $\langle \mathcal{PL}_g, Deliver \mid source, m \rangle$. Furthermore, it must guarantee the *reliable delivery*, *no duplication* and *No creation* property of a perfect point-to-point link abstraction.

Reliable delivery: if a $\langle \mathcal{PL}_g, Send \mid dest, m \rangle$ event is generated (from the application layer), then the first upon procedure reacts to this event by starting the propagation of the message $\langle source, dest, m \rangle$ over the proper perfect point-to-point link, triggering a *Send* event on the $\mathcal{PL}_l$ primitive. The $\mathcal{PL}_l$ abstraction guarantees all the perfect point-to-point link properties between two linked processes. Therefore, every *Send* event of the $\mathcal{PL}_l$ component generates a *Deliver* event on the destination process (e.g. a $\langle \mathcal{PL}_l, Send \mid p_2, \langle source, dest, m \rangle \rangle$ event on process p_1 generates the $\langle \mathcal{PL}_l, Deliver \mid p_1, \langle source, dest, m \rangle \rangle$ event on process p_2). The second upon procedure guarantees the propagation of the message $\langle source, dest, m \rangle$ till process $p_i, i = dest$. Only process $p_i, i = dest$ triggers the $\langle \mathcal{PL}_g, Deliver \mid source, m \rangle$ event. The propagation policy eventually guarantees the occurrence of this event.

No duplication: $\mathcal{PL}_l$ guarantees no duplication of the messages diffused by the primitive. For a single $\langle \mathcal{PL}_g, Send \mid dest, m \rangle$ event a single message $\langle source, dest, m \rangle$ is generated and propagated over the line using $\mathcal{PL}_l$, furthermore every process relays at most once a single $\langle source, dest, m \rangle$ message.

No creation: $\langle \mathcal{PL}_g, Deliver \mid source, m \rangle$ event can be triggered only from the reception of a message $\langle source, dest, m \rangle$. The message $\langle source, dest, m \rangle$ is initially generated by the processes only managing a $\langle \mathcal{PL}_g, Send \mid dest, m \rangle$ event.