MODULE *syncCon*1

EXTENDS *Integers*, *Sequences*, *FiniteSets*, *TLC*

CONSTANT N, *FAILNUM*

ASSUME $N \leq 5 \ \land 0 \leq FAILNUM \land FAILNUM \leq 2$

$Nodes \triangleq 1 \,..\, N$

```
--algorithm syncCon1
{
    variable FailNum = FAILNUM,
             up = [n ∈ Nodes ↦ TRUE],
             pt = [n ∈ Nodes ↦ 0],
             t = [n ∈ Nodes ↦ FALSE],
             d = [n ∈ Nodes ↦ -1],
             mb = [n ∈ Nodes ↦ {}];

    define {
    SetMin(S) ≜ CHOOSE i ∈ S : ∀ j ∈ S : i ≤ j
    UpNodes ≜ {n ∈ Nodes : up[n] = TRUE}
    }

    macro MaybeFail( ) {
        if ( FailNum > 0 ∧ up[self] )
            { either
                { up[self] := FALSE; FailNum := FailNum - 1; }
              or skip; } ;

    }

    fair process ( n ∈ Nodes )
    variable v = 0, pv = 0, Q = {};
    {
P:  if ( up[self] ) {
        v := self;
        Q := Nodes;
PS: while ( up[self] ∧ Q ≠ {} ) {
            with ( p ∈ Q ) {
                mb[p] := mb[p] ∪ {v}; \* Append the value of self nede to message bus of p node
                Q := Q \ {p}; \* Remove the p from Q to send message to other nodes
                MaybeFail() ; \* Call fail macro to fail the node if FailNum has not reached the maximum limit
             } ;
     } ;

    if ( up[self] ) pt[self] := pt[self] + 1;

PR: await (up[self] ∧ (∀ k ∈ UpNodes : pt[self] = pt[k])); \* Execute receive step only if the node is up and all the
        {
        d[self] := SetMin(mb[self]); \* Set the decision to the min of received value
```

```
        t[self] := TRUE ;                    \* Terminate the node once it has finalized the decision
      } await
    } if
  } process
} \* algorithm
BEGIN TRANSLATION
VARIABLES FailNum, up, pt, t, d, mb, pc

define statement
SetMin(S) ≜ CHOOSE i ∈ S : ∀ j ∈ S : i ≤ j
UpNodes ≜ {n ∈ Nodes : up[n] = TRUE}

VARIABLES v, pv, Q

vars ≜ ⟨FailNum, up, pt, t, d, mb, pc, v, pv, Q⟩

ProcSet ≜ (Nodes)

Init ≜ Global variables
        ∧ FailNum = FAILNUM
        ∧ up = [n ∈ Nodes ↦ TRUE]
        ∧ pt = [n ∈ Nodes ↦ 0]
        ∧ t = [n ∈ Nodes ↦ FALSE]
        ∧ d = [n ∈ Nodes ↦ −1]
        ∧ mb = [n ∈ Nodes ↦ {}]
        Process n
        ∧ v = [self ∈ Nodes ↦ 0]
        ∧ pv = [self ∈ Nodes ↦ 0]
        ∧ Q = [self ∈ Nodes ↦ {}]
        ∧ pc = [self ∈ ProcSet ↦ "P"]

P(self) ≜ ∧ pc[self] = "P"
          ∧ IF up[self]
              THEN ∧ v' = [v EXCEPT ![self] = self]
                   ∧ Q' = [Q EXCEPT ![self] = Nodes]
                   ∧ pc' = [pc EXCEPT ![self] = "PS"]
              ELSE ∧ pc' = [pc EXCEPT ![self] = "Done"]
                   ∧ UNCHANGED ⟨v, Q⟩
          ∧ UNCHANGED ⟨FailNum, up, pt, t, d, mb, pv⟩

PS(self) ≜ ∧ pc[self] = "PS"
           ∧ IF up[self] ∧ Q[self] ≠ {}
               THEN ∧ ∃ p ∈ Q[self] :
                        ∧ mb' = [mb EXCEPT ![p] = mb[p] ∪ {v[self]}]
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ Q' = [Q \text{ EXCEPT } ![self] = Q[self] \setminus \{p\}] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \text{ IF } FailNum > 0 \wedge up[self] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{THEN } \wedge \vee \wedge up' = [up \text{ EXCEPT } ![self] = \text{FALSE}] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge FailNum' = FailNum - 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \vee \wedge \text{TRUE} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge \text{UNCHANGED } \langle FailNum,\ up \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE } \wedge \text{TRUE} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge \text{UNCHANGED } \langle FailNum,\ up \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“PS”}] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ pt' = pt \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE } \wedge \text{IF } up[self] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{THEN } \wedge\ pt' = [pt \text{ EXCEPT } ![self] = pt[self] + 1] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE } \wedge \text{TRUE} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \wedge\ pt' = pt \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“PR”}] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \text{UNCHANGED } \langle FailNum,\ up,\ mb,\ Q \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge \text{UNCHANGED } \langle t,\ d,\ v,\ pv \rangle
\end{array}
$$

$$
\begin{array}{l}
PR(self) \triangleq \wedge\ pc[self] = \text{“PR”} \\
\qquad\qquad\quad \wedge\ (up[self] \wedge (\forall\, k \in UpNodes : pt[self] = pt[k])) \\
\qquad\qquad\quad \wedge\ d' = [d \text{ EXCEPT } ![self] = SetMin(mb[self])] \\
\qquad\qquad\quad \wedge\ t' = [t \text{ EXCEPT } ![self] = \text{TRUE}] \\
\qquad\qquad\quad \wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}] \\
\qquad\qquad\quad \wedge \text{UNCHANGED } \langle FailNum,\ up,\ pt,\ mb,\ v,\ pv,\ Q \rangle
\end{array}
$$

$$
n(self) \triangleq P(self) \vee PS(self) \vee PR(self)
$$

$$
\begin{array}{l}
Next \triangleq (\exists\, self \in Nodes : n(self)) \\
\qquad\quad \vee \text{ Disjunct to prevent deadlock on termination} \\
\qquad\qquad ((\forall\, self \in ProcSet : pc[self] = \text{“Done”}) \wedge \text{UNCHANGED } vars)
\end{array}
$$

$$
\begin{array}{l}
Spec \triangleq \wedge\ Init \wedge \Box[Next]_{vars} \\
\qquad\quad \wedge\ \forall\, self \in Nodes : \mathrm{WF}_{vars}(n(self))
\end{array}
$$

$$
Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})
$$

END TRANSLATION

* Below property is for termination

$$
FinalState \triangleq \text{TRUE} \rightsquigarrow (\forall\, i \in Nodes : up[i] = \text{TRUE} \Rightarrow t[i] = \text{TRUE})
$$

---

* Below are the invariants

$$
Inv \triangleq \forall\, i, j \in Nodes : (t[i] \wedge t[j]) \Rightarrow (d[i] = d[j])
$$

---

* Modification History

* Last modified *Tue Oct* 24 23:39:13 *EDT* 2017 by Deep

* Created Sun *Oct* 22 13:02:23 *EDT* 2017 by Deep

* DEEP *NARAYAN MISHRA – PERSON* NO 50245878

* As it is given in the problem statement, every progress broadcasts its initial
* value v to all other nodes in one round. After this round, each process decides
* the minimum value it received among all the nodes.

* This is implemented by first defining the variables which holds (up, round,
* termination, decision, and mailbox) values of all the nodes. All nodes are ran
* using strong fairness, so that all the nodes get chance to executes its step.

* The whole process is divided into two steps *PS* (process send), and *PR*
* (process receives). When the first round beings the process initializes its value (v)
* to self id, and Q is initialized to all the nodes in the system.
*
* Step *PS*: The process executes this step if it is up. During *PS* state process appends
* its value (v) to mailbox of all the nodes including itself. It also calls the macro
* *MaybeFail* (however this will not fail any node when *FAILNUM* is set to zero). After sending
* the value to all the nodes, process moves to next round.
*
* Step *PR*: A node waits for all the node to move to next round before executing this step.
* Once all the nodes are moved to the next round. Current process will find the min of the
* value received from all the node and set this as it's decision. After this it will terminate

* To check the validity of the program it will satisfy the agreement property.
* As per agreement property - Two correct processes can not commit to different decision variables.
* The above program will work if there is no failure of nodes. Because all the nodes will reach the
* Step *PR* and wait for other node to complete sending message to all. Once all the nodes have completed
* sending message to all the nodes, all the node will decide the minimum value from the messages received.
* Since all the nodes are receiving the value from all the nodes the decision of all the nodes will be same
* However, if there is a node failure. The failed node might have sent its minimum value to some nodes
* and failed before sending minimum to all the nodes. In this case, if the failed node value was
* least of all the nodes, the minimum calculated by few nodes (who received this value) will differ from others.
* Hence the agreement property is violated in the current model if there is any failure.