```
──────────────────────────── MODULE syncCon2 ────────────────────────────
EXTENDS Integers, Sequences, FiniteSets, TLC
CONSTANTS N, FAILNUM
ASSUME N ≤ 5 ∧ 0 ≤ FAILNUM ∧ FAILNUM ≤ 4
Nodes ≜ 1 . . N
Process ≜ 1 . . N
--algorithm syncCon2
{ variables 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
     }

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

    fair process ( n ∈ Process )
    variable v = 0, Q = {} ;
    {
P: await (up[self] ∧ ∀ s ∈ Nodes : mb[s] = {} ∧ (∀ i ∈ Nodes : pt[i] = pt[self]) ∧ t[self] = FALSE) ; {
    if ( pt[self] = 0 ) v := self ;   If in round 0, each node broadcast its own value
    else v := d[self] ;   If in other round, each node broadcast the minimum value it received.
    Q := Nodes ;   Broadcast value to Nodes, if one node is crashed, we needn't broad value to it.
PS: while ( up[self] ∧ Q ≠ {} ) {
        with ( p ∈ Q ) {
           MaybeFail() ;   In process n, each time we add v to mb[p], this process might be fail, and once the process
                           fail, we set up[self] fail and after that, all operations will be invalid.
           if ( up[self] = TRUE )   Test if operations have been failed before
              mb[p] := mb[p] ∪ {v} ;   In process n, add v to mb[1] to mb[N]
           Q := Q \ {p} ;   For each element p in Q, we have to broadcast value v to mb[p], after adding v to mb[p],
                            remove p from Q in case of adding v to mb[p] again.
         } ;
       } ;
      if ( up[self] ) pt[self] := pt[self] + 1 ;
      if ( up[self] = FALSE ) Nodes := Nodes \ {self} ;   If this process crashed, remove this node from nodes
PR: await (up[self] ∧ (∀ i ∈ Nodes : pt[i] = pt[self])) ;   Wait until all process finished
      d[self] := SetMin(mb[self]) ;
    $mb[self] := \{\}$ ;   The mailbox should be empty before every round begin.
    if ( $\forall i \in Nodes : d[self] = d[i]$ ) {
        $t[self] :=$ TRUE ;   If all $d[i]$ in $Nodes$ are the same, the result
                              is consensus, then this process terminate normally.
     }
    else goto $P$ ;   If some $d[i]$ in $Nodes$ is different, some nodes must be crashed, so go to the next round.
   } ;
 }
}
BEGIN TRANSLATION
VARIABLES $FailNum, up, pt, t, d, mb, pc$

define statement
$SetMin(S) \triangleq$ CHOOSE $i \in S : \forall j \in S : i \leq j$

VARIABLES $v, Q$

$vars \triangleq \langle FailNum, up, pt, t, d, mb, pc, v, Q \rangle$

$ProcSet \triangleq (Process)$

$Init \triangleq$  Global variables
          $\land FailNum = FAILNUM$
          $\land up = [n \in Nodes \mapsto$ TRUE$]$
          $\land pt = [n \in Nodes \mapsto 0]$
          $\land t = [n \in Nodes \mapsto$ FALSE$]$
          $\land d = [n \in Nodes \mapsto -1]$
          $\land mb = [n \in Nodes \mapsto \{\}]$
          Process $n$
          $\land v = [self \in Process \mapsto 0]$
          $\land Q = [self \in Process \mapsto \{\}]$
          $\land pc = [self \in ProcSet \mapsto$ "P"$]$

$P(self) \triangleq$ $\land pc[self] =$ "P"
            $\land (up[self] \land \forall s \in Nodes : mb[s] = \{\} \land (\forall i \in Nodes : pt[i] = pt[self]) \land t[self] =$ FALSE$)$
            $\land$ IF $pt[self] = 0$
                  THEN $\land v' = [v$ EXCEPT $![self] = self]$
                  ELSE $\land v' = [v$ EXCEPT $![self] = d[self]]$
            $\land Q' = [Q$ EXCEPT $![self] = Nodes]$
            $\land pc' = [pc$ EXCEPT $![self] =$ "PS"$]$
            $\land$ UNCHANGED $\langle FailNum, up, pt, t, d, mb \rangle$

$PS(self) \triangleq$ $\land pc[self] =$ "PS"
             $\land$ IF $up[self] \land Q[self] \neq \{\}$
                   THEN $\land \exists p \in Q[self]$ :
                             $\land$ IF $FailNum > 0 \land up[self]$
                                   THEN $\land \lor \land up' = [up$ EXCEPT $![self] =$ FALSE$]$
```

```
                                                   ∧ FailNum′ = FailNum − 1
                                                ∨  ∧ TRUE
                                                   ∧ UNCHANGED ⟨FailNum, up⟩
                                       ELSE  ∧ TRUE
                                             ∧ UNCHANGED ⟨FailNum, up⟩
                              ∧ IF up′[self] = TRUE
                                     THEN  ∧ mb′ = [mb EXCEPT ![p] = mb[p] ∪ {v[self]}]
                                     ELSE  ∧ TRUE
                                           ∧ mb′ = mb
                              ∧ Q′ = [Q EXCEPT ![self] = Q[self] \ {p}]
                           ∧ pc′ = [pc EXCEPT ![self] = "PS"]
                           ∧ pt′ = pt
                    ELSE   ∧ IF up[self]
                                 THEN  ∧ pt′ = [pt EXCEPT ![self] = pt[self] + 1]
                                 ELSE  ∧ TRUE
                                       ∧ pt′ = pt
                           ∧ IF up[self] = FALSE
                                 THEN  ∧ Nodes′ = Nodes \ {self}
                                 ELSE  ∧ TRUE
                           ∧ pc′ = [pc EXCEPT ![self] = "PR"]
                           ∧ UNCHANGED ⟨FailNum, up, mb, Q⟩
               ∧ UNCHANGED ⟨t, d, v⟩

PR(self) ≜  ∧ pc[self] = "PR"
            ∧ (up[self] ∧ (∀ i ∈ Nodes : pt[i] = pt[self]))
            ∧ d′ = [d EXCEPT ![self] = SetMin(mb[self])]
            ∧ mb′ = [mb EXCEPT ![self] = {}]
            ∧ IF ∀ i ∈ Nodes : d′[self] = d′[i]
                  THEN  ∧ t′ = [t EXCEPT ![self] = TRUE]
                        ∧ pc′ = [pc EXCEPT ![self] = "Done"]
                  ELSE  ∧ pc′ = [pc EXCEPT ![self] = "P"]
                        ∧ t′ = t
            ∧ UNCHANGED ⟨FailNum, up, pt, v, Q⟩

n(self) ≜ P(self) ∨ PS(self) ∨ PR(self)

Next ≜ (∃ self ∈ Process : n(self))
           ∨  Disjunct to prevent deadlock on termination
              ((∀ self ∈ ProcSet : pc[self] = "Done") ∧ UNCHANGED vars)

Spec ≜ ∧ Init ∧ □[Next]_vars
       ∧ ∀ self ∈ Process : WF_vars(n(self))

Termination ≜ ◇(∀ self ∈ ProcSet : pc[self] = "Done")

END TRANSLATION
```

$$\mathit{agreement} \triangleq (\forall\, i, j \in \mathit{Nodes} : t[i] \land t[j] \Rightarrow d[i] = d[j])$$

* Modification History

* Last modified *Tue Oct* 24 21:28:08 *EDT* 2017 by *lenovo*

* Created *Tue Oct* 24 00:19:48 *EDT* 2017 by *lenovo*