```
──────────────────── MODULE voldemort ────────────────────
 /*Replicated storage protocol with clienside routing.
 /*Debaditya Basak, 11 Nov 2016
EXTENDS Integers, Sequences, FiniteSets, TLC
CONSTANTS N, C, STOP, ReadQ, WriteQ, FAILNUM
ASSUME N = 5 ∧ C = 1 ∧ STOP < 10 ∧ 1 ≤ ReadQ ∧ ReadQ ≤ 3
            ∧ 1  ≤ WriteQ ∧ WriteQ   ≤ 3 ∧ 0 ≤ FAILNUM ∧ FAILNUM ≤ 2
Nodes ≜ 1 . . N
Clients ≜ N + 1 . . N + C  /*should give different ID space to Client

--algorithm voldemort{
    variable FailNum = FAILNUM,
             state = "Reading",
             state1 = "InProcess",
             wQ = WriteQ,
             up  = [n ∈ Nodes ↦ TRUE],   /*Initially all nodes are up
             db  = [n ∈ Nodes ↦ [ver ↦ 0, val ↦ 0]] ;
              /*All nodes have database, wherein [ver = 0, val = 0] stored for the item

    define {
    UpNodes ≜ {i ∈ Nodes : up[i] = TRUE}
    ReturnReadQ ≜ CHOOSE i ∈ SUBSET (UpNodes) : Cardinality(i) = ReadQ
    ReturnWriteQ ≜ CHOOSE i ∈ SUBSET (UpNodes) : Cardinality(i) = WriteQ
    NodeR ≜ CHOOSE i ∈ ReturnReadQ : ∀ j ∈ ReturnReadQ : db[i].ver ≥ db[j].ver
    NodeW(Q) ≜ CHOOSE i ∈ Q : ∀ j ∈ Q : i ≤ j
    ClientNotWriting ≜ state1 = "InProcess"     ∧ state  = "Reading"
    ClientAtomicWriteDone ≜ state = "Writing" ∧ state1 = "WriteEnd"
     }

    fair process ( c ∈ Clients )
    variable cntr = 0, hver = 0, Q = {} ;
    {
     CL: while ( cntr ≤ STOP ) {
          state := "Reading" ;
          state1 := "InProcess" ;
          cntr := cntr + 1 ;
          hver := db[NodeR].ver + 1 ;
          Q := ReturnWriteQ ;
           /*Nodes can fail or come back up between atomic states CL and CL1
          CL1: while ( Q ≠ {} ) {
                state := "Writing" ;
                db[NodeW(Q)].ver := hver || db[NodeW(Q)].val := cntr ;
                Q := Q \ {NodeW(Q)} ;
                if ( Q = {} ) {
                 state1 := "WriteEnd" ;
                  }
                }
            }
        }

    fair process ( n ∈ Nodes )
    {
     NODE: while ( TRUE ∧ ClientNotWriting ) {
       /*To make Clients-process WRITE atomic
       /*Nodes state change only when ClientNotWriting is TRUE
            if ( FailNum > 0 ∧ up[self] = TRUE ) {   /*Storage node can fail
                    up[self] := FALSE ;
                    FailNum := FailNum − 1 ;
                     }
            else if ( up[self] = FALSE ) {   /*Or recover
                    up[self] := TRUE ;
                    FailNum := FailNum + 1 ;
                     }
             }
        }

 }
  ****************************************************************************
```

```
BEGIN TRANSLATION
VARIABLES FailNum, state, state1, wQ, up, db, pc

define statement
UpNodes ≜ {i ∈ Nodes : up[i] = TRUE}
ReturnReadQ ≜ CHOOSE i ∈ SUBSET (UpNodes) : Cardinality(i) = ReadQ
ReturnWriteQ ≜ CHOOSE i ∈ SUBSET (UpNodes) : Cardinality(i) = WriteQ
NodeR ≜ CHOOSE i ∈ ReturnReadQ : ∀ j ∈ ReturnReadQ : db[i].ver ≥ db[j].ver
NodeW(Q) ≜ CHOOSE i ∈ Q : ∀ j ∈ Q : i ≤ j
ClientNotWriting ≜ state1 = "InProcess"      ∧ state  = "Reading"
ClientAtomicWriteDone ≜ state = "Writing" ∧ state1 = "WriteEnd"

VARIABLES cntr, hver, Q

vars ≜ ⟨FailNum, state, state1, wQ, up, db, pc, cntr, hver, Q⟩

ProcSet ≜ (Clients) ∪ (Nodes)

Init ≜  Global variables
          ∧ FailNum = FAILNUM
          ∧ state = "Reading"
          ∧ state1 = "InProcess"
          ∧ wQ = WriteQ
          ∧ up  = [n ∈ Nodes ↦ TRUE]
          ∧ db  = [n ∈ Nodes ↦ [ver ↦ 0, val ↦ 0]]
```

Process c

```
$\land\ cntr = [self \in Clients \mapsto 0]$
$\land\ hver = [self \in Clients \mapsto 0]$
$\land\ Q = [self \in Clients \mapsto \{\}]$
$\land\ pc = [self \in ProcSet \mapsto$ CASE $self \in Clients \rightarrow$ "CL"
                                      $\Box$ $self \in Nodes \rightarrow$ "NODE"$]$

$CL(self) \triangleq$ $\land\ pc[self] =$ "CL"
         $\land$ IF $cntr[self] \leq STOP$
              THEN $\land\ state' =$ "Reading"
                   $\land\ state1' =$ "InProcess"
                   $\land\ cntr' = [cntr$ EXCEPT $![self] = cntr[self] + 1]$
                   $\land\ hver' = [hver$ EXCEPT $![self] = db[NodeR].ver + 1]$
                   $\land\ Q' = [Q$ EXCEPT $![self] = ReturnWriteQ]$
                   $\land\ pc' = [pc$ EXCEPT $![self] =$ "CL1"$]$
              ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "Done"$]$
                   $\land$ UNCHANGED $\langle state, state1, cntr, hver, Q\rangle$
         $\land$ UNCHANGED $\langle FailNum, wQ, up, db\rangle$

$CL1(self) \triangleq$ $\land\ pc[self] =$ "CL1"
          $\land$ IF $Q[self] \neq \{\}$
               THEN $\land\ state' =$ "Writing"
                    $\land\ db' = [db$ EXCEPT $![NodeW(Q[self])].ver = hver[self],$
                                        $![NodeW(Q[self])].val = cntr[self]]$
                    $\land\ Q' = [Q$ EXCEPT $![self] = Q[self] \setminus \{NodeW(Q[self])\}]$
                    $\land$ IF $Q'[self] = \{\}$
                         THEN $\land\ state1' =$ "WriteEnd"
                         ELSE $\land$ TRUE
                              $\land$ UNCHANGED $state1$
                    $\land\ pc' = [pc$ EXCEPT $![self] =$ "CL1"$]$
               ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "CL"$]$
                    $\land$ UNCHANGED $\langle state, state1, db, Q\rangle$
          $\land$ UNCHANGED $\langle FailNum, wQ, up, cntr, hver\rangle$

$c(self) \triangleq CL(self) \lor CL1(self)$

$NODE(self) \triangleq$ $\land\ pc[self] =$ "NODE"
            $\land$ IF TRUE $\land\ ClientNotWriting$
                 THEN $\land$ IF $FailNum > 0 \land up[self] =$ TRUE
                           THEN $\land\ up' = [up$ EXCEPT $![self] =$ FALSE$]$
                                $\land\ FailNum' = FailNum - 1$
                           ELSE $\land$ IF $up[self] =$ FALSE
                                     THEN $\land\ up' = [up$ EXCEPT $![self] =$ TRUE$]$
                                          $\land\ FailNum' = FailNum + 1$
                                     ELSE $\land$ TRUE
                                          $\land$ UNCHANGED $\langle FailNum, up\rangle$
                         $\land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“NODE”}]$
                 ELSE    $\land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“Done”}]$
                         $\land\ \text{UNCHANGED}\ \langle FailNum,\ up \rangle$
            $\land\ \text{UNCHANGED}\ \langle state,\ state1,\ wQ,\ db,\ cntr,\ hver,\ Q \rangle$

$n(self) \triangleq NODE(self)$

$Next \triangleq (\exists\, self \in Clients : c(self))$
          $\lor\ (\exists\, self \in Nodes : n(self))$
          $\lor$  Disjunct to prevent deadlock on termination
            $((\forall\, self \in ProcSet : pc[self] = \text{“Done”}) \land \text{UNCHANGED}\ vars)$

$Spec \triangleq \land\ Init \land \Box[Next]_{vars}$
          $\land\ \forall\, self \in Clients : \mathrm{WF}_{vars}(c(self))$
          $\land\ \forall\, self \in Nodes : \mathrm{WF}_{vars}(n(self))$

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

 END TRANSLATION
 $InvP$ is the Invariant function which checks $db$ consistency after every atomic Client WRITE ends.
$InvP \triangleq \forall\, p \in Clients : (ClientAtomicWriteDone = \text{TRUE} \Rightarrow$
                          $(db[NodeR].ver \quad = hver[p] \land db[NodeR].val = cntr[p]))$
```

```
\* Observation: Model checking the system with $FAILNUM = ReadQ$ and $ReadQ = WriteQ$
\*          causes it to violate the Invariant. This is evident, as, the size
\*          of $WriteQ/ReadQ$ is then not enough to overcome the node failure
\*          count in case $FAILNUM$ number of nodes fail. So say $ReadQ/WriteQ = 2$
\*          and $FAILNUM = 2$. In this case the Client process atomically Reads
\*          the highest version from 2 $ReadQ$ nodes, and fetches the $WriteQ$ to
\*          write the latest entries into. But if now the system decides to
\*          fail the two nodes which were selected as the $WriteQ$, the next
\*          invocation of the Client process will write data to FAILED nodes.
\*        - This is where the Invariant will fail because it will try to get
\*          the values from a $ReadQ$ which is not consistent with nodes to
\*          which the Client process is writing the latest updates to.
\*
\*        - The Invariant is also violated if $ReadQ < \ = FAILNUM$    and
\*          $WriteQ > FAILNUM$. This is because even though $WriteQ$ selects safe
\*          number of up-nodes to perform the writes, the $ReadQ$ is only
\*          returning the highest version from a subset of nodes in $WriteQ$.
\*          Thereby failing to return the true highest version that exists in
\*          the up-nodes.
\*
\*        - So, Invariant will only be satisfied if $FAILNUM < ReadQ$ and
\*          $FAILNUM < WriteQ$. This is the pre-requisite that guarantees single
\*          -copy consistency in the system.
\*
\* $teamMember$: Debaditya $Basak$, 50206177
```