MODULE *Commons*

EXTENDS *Integers*

CONSTANTS *CLIENT_NUM*, the number of clients
MAX_KILL maximum allowed kill events

VARIABLES *state*, the program state, running or terminated
clients, clients sending value update requests to master and backup
master, pool of master instances, only one is active
backup, pool of backup instances, only one is active
msgs, in-flight messages
killed number of invoked kill actions to master or backup

Identifiers related to master and backup instance ids

$FIRST\_ID \triangleq 1$
$MAX\_INSTANCE\_ID \triangleq MAX\_KILL + 1$
$INSTANCE\_ID \triangleq FIRST\_ID \,..\, MAX\_INSTANCE\_ID$
$UNKNOWN\_ID \triangleq 0$
$NOT\_INSTANCE\_ID \triangleq -1$

Identifiers related to master and backup instance statuses

$INST\_STATUS\_NULL \triangleq$ "null" null, not used yet
$INST\_STATUS\_ACTIVE \triangleq$ "active" active and handling client requests
$INST\_STATUS\_LOST \triangleq$ "lost" lost
$INST\_STATUS\_BUSY \triangleq$ "busy" busy recoverying the other replica
$NOT\_STATUS \triangleq$ "invalid" invalid status
$INSTANCE\_STATUS \triangleq \{INST\_STATUS\_NULL, INST\_STATUS\_ACTIVE, INST\_STATUS\_LOST, INST\_STATUS\_BUSY\}$

$LiveStatus \triangleq \{INST\_STATUS\_ACTIVE, INST\_STATUS\_BUSY\}$

Master instance record structure

$Master \triangleq [id : INSTANCE\_ID, backupId : INSTANCE\_ID \cup \{UNKNOWN\_ID\}, status : INSTANCE\_STATUS, value : Nat, version : Nat]$

Invalid master instance

$NOT\_MASTER \triangleq [id \mapsto NOT\_INSTANCE\_ID, backupId \mapsto NOT\_INSTANCE\_ID, status \mapsto NOT\_STATUS, value \mapsto -1, version \mapsto -1]$

Backup instance record structure

$Backup \triangleq [id : INSTANCE\_ID, masterId : INSTANCE\_ID \cup \{UNKNOWN\_ID\}, status : INSTANCE\_STATUS, value : Nat, version : Nat]$

Invalid backup instance

$NOT\_BACKUP \triangleq [id \mapsto NOT\_INSTANCE\_ID,\ masterId \mapsto NOT\_INSTANCE\_ID,$
$\quad status \mapsto NOT\_STATUS,\ value \mapsto -1,\ version \mapsto -1]$

$SearchForMaster \triangleq$

Return the live master, or NOT_MASTER if master is lost

LET $mset \triangleq \{m \in INSTANCE\_ID : master[m].status \in LiveStatus\}$
IN IF $mset = \{\}$ THEN NOT_MASTER
ELSE $master[(\text{CHOOSE } x \in mset : \text{TRUE})]$

$LastLostMaster \triangleq$

Return the lost master, or NOT_MASTER if master is alive

LET $mset \triangleq \{m \in INSTANCE\_ID : master[m].status = INST\_STATUS\_LOST\}$
IN IF $mset = \{\}$ THEN NOT_MASTER
ELSE $master[(\text{CHOOSE } n \in mset : \forall\, m \in mset : n \geq m)]$

$FindMaster(mStatus) \triangleq$

Return the master with given status or NOT_MASTER otherwise

LET $mset \triangleq \{m \in INSTANCE\_ID : master[m].status = mStatus\}$
IN IF $mset = \{\}$ THEN NOT_MASTER
ELSE $master[(\text{CHOOSE } x \in mset : \text{TRUE})]$

$LastKnownMaster \triangleq$

Return the last known master, whether active, busy or lost

LET $mset \triangleq \{m \in INSTANCE\_ID : master[m].status \neq INST\_STATUS\_NULL\}$
IN $master[(\text{CHOOSE } n \in mset : \forall\, m \in mset : n \geq m)]$

$LiveBackup \triangleq$

Return the active back, or NOT_BACKUP if backup is lost or busy

LET $bset \triangleq \{b \in INSTANCE\_ID : backup[b].status \in LiveStatus\}$
IN IF $bset = \{\}$ THEN NOT_BACKUP
ELSE $backup[(\text{CHOOSE } x \in bset : \text{TRUE})]$

$FindBackup(bStatus) \triangleq$

Return the backup with given status or NOT_BACKUP otherwise

LET $bset \triangleq \{b \in INSTANCE\_ID : backup[b].status = bStatus\}$
IN IF $bset = \{\}$ THEN NOT_BACKUP
ELSE $backup[(\text{CHOOSE } x \in bset : \text{TRUE})]$

$LastLostBackup \triangleq$

Return the lost backup, or NOT_BACKUP if backup is alive

LET $bset \triangleq \{b \in INSTANCE\_ID : backup[b].status = INST\_STATUS\_LOST\}$
IN IF $bset = \{\}$ THEN NOT_BACKUP
ELSE $backup[(\text{CHOOSE } n \in bset : \forall\, m \in bset : n \geq m)]$

$SearchForBackup \triangleq$

Return the live backup, or NOT_BACKUP if backup is lost

LET $bset \triangleq \{b \in INSTANCE\_ID : backup[b].status \in LiveStatus\}$
IN IF $bset = \{\}$ THEN NOT_BACKUP
ELSE $backup[(\text{CHOOSE } x \in bset : \text{TRUE})]$

$LastKnownBackup \triangleq$

Return the last known backup, whether active, busy or lost

LET $bset \triangleq \{m \in INSTANCE\_ID : backup[m].status \neq INST\_STATUS\_NULL\}$
IN $backup[(\text{CHOOSE } n \in bset : \forall\, m \in bset : n \geq m)]$

Identifiers related to client ids and phases

$CLIENT\_ID \triangleq 1 \,..\, CLIENT\_NUM$
$NOT\_CLIENT\_ID \triangleq -1$

client phases

$CLIENT\_PHASE \triangleq 1 \,..\, 4$
$PH1\_PENDING \triangleq 1$
$PH2\_WORKING \triangleq 2$
$PH2\_COMPLETED \triangleq 3$
$PH2\_COMPLETED\_FATAL \triangleq 4$
$NOT\_CLIENT\_PHASE \triangleq -1$

Client record structure

$Client \triangleq [id : CLIENT\_ID,\ phase : CLIENT\_PHASE,\ value : Nat,$
$masterId : INSTANCE\_ID,$ the master instance last communicated with
$backupId : INSTANCE\_ID \cup \{UNKNOWN\_ID\}$ the backup instance last communicated with, initially unl
$]$

Invalid client instance

$NOT\_CLIENT \triangleq [id \mapsto NOT\_CLIENT\_ID,\ phase \mapsto NOT\_CLIENT\_PHASE,\ value \mapsto 0]$

$FindClient(phase) \triangleq$

Return a client matching the given phase, or NOT_CLIENT otherwise

LET $cset \triangleq \{c \in CLIENT\_ID : clients[c].phase = phase\}$
IN IF $cset = \{\}$ THEN NOT_CLIENT
ELSE $clients[(\text{CHOOSE } x \in cset : \text{TRUE})]$

Message record structure

$Messages \triangleq [from : \{\text{“c”},\ \text{“m”},\ \text{“b”},\ \text{“sys”}\},\ to : \{\text{“c”},\ \text{“m”},\ \text{“b”}\},$
$clientId : CLIENT\_ID,$
$masterId : INSTANCE\_ID \cup \{UNKNOWN\_ID\},$
$backupId : INSTANCE\_ID \cup \{UNKNOWN\_ID\},$
$value : Nat,$
$tag : \{\text{“masterDo”},\ \text{“backupDo”},$
“masterDone”, “backupDone”,
“masterDoFailed”, “backupDoFailed”,

```
                    "masterGetNewBackup", "backupGetNewMaster",
                    "newBackupId", "newMasterId",
                    "backupGetNewMasterFailed", "masterGetNewBackupFailed"
                  }]

Invalid message instance
NOT_MESSAGE ≜ [from ↦ "na", to ↦ "na"]

SendMsg(m) ≜
  Add message to the msgs set
  msgs' = msgs ∪ {m}

RecvMsg(m) ≜
  Delete message from the msgs set
  msgs' = msgs \ {m}

ReplaceMsg(toRemove, toAdd) ≜
  Remove an existing message and add another one
  msgs' = (msgs \ {toRemove}) ∪ {toAdd}

FindMessageToWithTag(to, status, optionalTag) ≜
  Return a message matching the given criteria, or NOT_MESSAGE otherwise
  LET mset ≜ {m ∈ msgs : ∧ m.to = to
                         ∧ IF to = "m"
                           THEN master[m.masterId].status = status
                           ELSE IF to = "b"
                           THEN backup[m.backupId].status = status
                           ELSE FALSE
                         ∧ IF optionalTag = "NA"
                           THEN TRUE
                           ELSE m.tag = optionalTag}
  IN  IF mset = {} THEN NOT_MESSAGE
      ELSE (CHOOSE x ∈ mset : TRUE)

FindMessageTo(to, status) ≜ FindMessageToWithTag(to, status, "NA")

FindMessageToClient(from, tag) ≜
  Return a message sent to client matching given criteria, or NOT_MESSAGE otherwise
  LET mset ≜ {m ∈ msgs : ∧ m.from = from
                         ∧ m.to = "c"
                         ∧ m.tag = tag}
  IN  IF mset = {} THEN NOT_MESSAGE
      ELSE (CHOOSE x ∈ mset : TRUE)

\* Modification History
```

\ * Last modified *Mon Mar* 19 19:10:52 *AEDT* 2018 by *u*5482878
\ * Last modified Sat *Mar* 17 16:13:02 *AEDT* 2018 by *shamouda*
\ * Created *Mon Mar* 05 13:44:57 *AEDT* 2018 by *u*5482878