```
1  ---------- MODULE AsyncFinishReplication ----------
2  EXTENDS Integers

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

7  VARIABLES exec_state,    the execution state of the program: running, success, or fatal
8            clients,      clients sending value update requests to master and backup
9            master,       array of master instances, only one is active
10           backup,       array of backup instances, only one is active
11           msgs,         in-flight messages
12           killed        number of invoked kill actions to master or backup

14 ------------------------------------------------------
15 Vars ≜ ⟨exec_state, clients, master, backup, msgs, killed⟩
16 ------------------------------------------------------
17 C ≜ INSTANCE Commons
18 ------------------------------------------------------
19 TypeOK ≜
     Variables type constrains
23   ∧ clients ∈ [C!CLIENT_ID → C!Client]
24   ∧ master ∈ [C!INSTANCE_ID → C!Master]
25   ∧ backup ∈ [C!INSTANCE_ID → C!Backup]
26   ∧ exec_state ∈ {"running", "success", "fatal"}
27   ∧ msgs ⊆ C!Messages
28   ∧ killed ∈ 0 .. MAX_KILL

30 ------------------------------------------------------
31 MaxOneActiveMaster ≜
     Return true if maximum one active master exists, and false otherwise
35   LET activeM ≜ C!FindMaster(C!INST_STATUS_ACTIVE)
36       otherIds ≜ C!INSTANCE_ID \ {activeM.id}
37   IN  IF activeM = C!NOT_MASTER
38       THEN TRUE   zero active masters
39       ELSE LET otherActiveMs ≜ {m ∈ otherIds : master[m].status = C!INST_STATUS_ACTIVE}
40            IN  IF otherActiveMs = {} THEN TRUE   no other active masters
41                ELSE FALSE   other active masters exist

43 MaxOneActiveBackup ≜
     Return true if maximum one active backup exists, and false otherwise
47   LET activeB ≜ C!FindBackup(C!INST_STATUS_ACTIVE)
48       otherIds ≜ C!INSTANCE_ID \ {activeB.id}
49   IN  IF activeB = C!NOT_BACKUP
50       THEN TRUE   zero active backups
51       ELSE LET otherActiveBs ≜ {b ∈ otherIds : backup[b].status = C!INST_STATUS_ACTIVE}
52            IN  IF otherActiveBs = {} THEN TRUE   no other active backups
```

```
                      ELSE FALSE   other active backup exist

StateOK ≜
    State invariants
    1. on successful termination: the final version equals CLIENT_NUM
    2. on fatal termination: there must be a client whose master is lost and whose backup is lost
       or is unknown
    3. before termination:
       a) master version ≥ backup version
       b) master and backup version should not exceed CLIENT_NUM
       c) maximum one active master and maximum one active backup
    LET curMaster ≜ C!LastKnownMaster
        curBackup ≜ C!LastKnownBackup
    IN   IF exec_state = "success"
          THEN ∧ curMaster.version = CLIENT_NUM
               ∧ curBackup.version = CLIENT_NUM
          ELSE IF exec_state = "fatal"
          THEN ∃ c ∈ C!CLIENT_ID :
                 ∧ clients[c].phase = C!PH2_COMPLETED_FATAL
                 ∧ master[clients[c].masterId].status = C!INST_STATUS_LOST
                 ∧ IF clients[c].backupId ≠ C!UNKNOWN_ID
                     THEN backup[clients[c].backupId].status = C!INST_STATUS_LOST
                     ELSE TRUE
          ELSE ∧ curMaster.version ≥ curBackup.version
               ∧ curMaster.version ≤ CLIENT_NUM
               ∧ curBackup.version ≤ CLIENT_NUM
               ∧ MaxOneActiveMaster
               ∧ MaxOneActiveBackup

|----------------------------------------------------------------------|
MustTerminate ≜
    Temporal property: the program must eventually terminate either successfully or fatally
   ◇(exec_state ∈ {"success", "fatal"})
|----------------------------------------------------------------------|
Init ≜
    Initialiaze variables
   ∧ exec_state = "running"
   ∧ clients = [i ∈ C!CLIENT_ID ↦ [id ↦ i, phase ↦ C!PH1_PENDING,
                   value ↦ i,  masterId ↦ C!FIRST_ID, backupId ↦ C!UNKNOWN_ID]]
   ∧ backup = [i ∈ C!INSTANCE_ID ↦
                  IF i = C!FIRST_ID
                   THEN [id ↦ C!FIRST_ID, masterId ↦ C!FIRST_ID, status ↦ C!INST_STATUS_ACTIV
                         value ↦ 0, version ↦ 0]
                   ELSE [id ↦ i, masterId ↦ C!UNKNOWN_ID, status ↦ C!INST_STATUS_NULL,
                         value ↦ 0, version ↦ 0]]
   ∧ master = [i ∈ C!INSTANCE_ID ↦
```

```
106                 IF i = C!FIRST_ID
107                  THEN [id ↦ C!FIRST_ID, backupId ↦ C!FIRST_ID, status ↦ C!INST_STATUS_ACTIV
108                        value ↦ 0, version ↦ 0]
109                  ELSE [id ↦ i, backupId ↦ C!UNKNOWN_ID, status ↦ C!INST_STATUS_NULL,
110                        value ↦ 0, version ↦ 0]]
111    ∧ msgs = {}
112    ∧ killed = 0

114 ├──────────────────────────────────────────────────────────────────────┤
115 E_KillingMaster ≜
      Kill the active master instance.
119    ∧ exec_state = "running"
120    ∧ killed < MAX_KILL
121    ∧ LET activeM ≜ C!FindMaster(C!INST_STATUS_ACTIVE)
122      IN  ∧ activeM ≠ C!NOT_MASTER
123          ∧ master′ = [master EXCEPT ![activeM.id].status = C!INST_STATUS_LOST]
124          ∧ killed′ = killed + 1
125    ∧ UNCHANGED ⟨exec_state, clients, backup, msgs⟩

127 E_KillingBackup ≜
      Kill the active backup instance.
131    ∧ exec_state = "running"
132    ∧ killed < MAX_KILL
133    ∧ LET activeB ≜ C!FindBackup(C!INST_STATUS_ACTIVE)
134      IN  ∧ activeB ≠ C!NOT_BACKUP
135          ∧ backup′ = [backup EXCEPT ![activeB.id].status = C!INST_STATUS_LOST]
136          ∧ killed′ = killed + 1
137    ∧ UNCHANGED ⟨exec_state, clients, master, msgs⟩

139 C_Starting ≜
      Client start the replication process by sending "do" to master
143    ∧ exec_state = "running"
144    ∧ LET client ≜ C!FindClient(C!PH1_PENDING)
145      IN  ∧ client ≠ C!NOT_CLIENT
146          ∧ C!SendMsg([from ↦ "c",
147                       to ↦ "m",
148                       clientId ↦ client.id,
149                       masterId ↦ client.masterId,
150                       backupId ↦ C!UNKNOWN_ID,
151                       value ↦ client.value,
152                       tag ↦ "masterDo"])
153          ∧ clients′ = [clients EXCEPT ![client.id].phase = C!PH2_WORKING]
154    ∧ UNCHANGED ⟨exec_state, master, backup, killed⟩

156 M_Doing ≜
```

```
    Master receiving "do", updating value, and sending "done"
160 ∧ exec_state = "running"
161 ∧ LET msg ≜ C!FindMessageToWithTag("m", C!INST_STATUS_ACTIVE, "masterDo")
162   IN ∧ msg ≠ C!NOT_MESSAGE
163      ∧ master' = [master EXCEPT ![msg.masterId].value = master[msg.masterId].value + msg.value,
164                                 ![msg.masterId].version = master[msg.masterId].version + 1]
165      ∧ C!ReplaceMsg(msg, [from ↦ "m",
166                           to ↦ "c",
167                           clientId ↦ msg.clientId,
168                           masterId ↦ msg.masterId,
169                           backupId ↦ master[msg.masterId].backupId,
170                           value ↦ 0,
171                           tag ↦ "masterDone"])
172 ∧ UNCHANGED ⟨exec_state, clients, backup, killed⟩

174 C_HandlingMasterDone ≜
    Client receiving "done" from master, and forwarding action to backup
178 ∧ exec_state = "running"
179 ∧ LET msg ≜ C!FindMessageToClient("m", "masterDone")
180   IN ∧ msg ≠ C!NOT_MESSAGE
181      ∧ C!ReplaceMsg(msg, [from ↦ "c",
182                           to ↦ "b",
183                           clientId ↦ msg.clientId,
184                           masterId ↦ msg.masterId,
185                           backupId ↦ msg.backupId,
186                           value ↦ clients[msg.clientId].value,
187                           tag ↦ "backupDo"])
188       update our knowledge about the backup identity
189      ∧ clients' = [clients EXCEPT ![msg.clientId].backupId = msg.backupId]
190 ∧ UNCHANGED ⟨exec_state, master, backup, killed⟩

192 B_Doing ≜
    Backup receiving "do", updating value, then sending "done"
196 ∧ exec_state = "running"
197 ∧ LET msg ≜ C!FindMessageToWithTag("b", C!INST_STATUS_ACTIVE, "backupDo")
198   IN ∧ msg ≠ C!NOT_MESSAGE
199        Master info is consistent between client and backup
200      ∧ msg.masterId = backup[msg.backupId].masterId
201      ∧ backup' = [backup EXCEPT ![msg.backupId].value = backup[msg.backupId].value + msg.value,
202                                 ![msg.backupId].version = backup[msg.backupId].version + 1]
203      ∧ C!ReplaceMsg(msg, [from ↦ "b",
204                           to ↦ "c",
205                           clientId ↦ msg.clientId,
206                           masterId ↦ msg.masterId,
207                           backupId ↦ msg.backupId,
```

```
                                        value ↦ 0,
                                        tag ↦ "backupDone"])
  ∧ UNCHANGED ⟨exec_state, clients, master, killed⟩

B_DetectingOldMasterId ≜
  Backup receiving "do" and detecting that the client is using an old master id. It does not
  update the value, and it sends the new master id to the client
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToWithTag("b", C!INST_STATUS_ACTIVE, "backupDo")
    IN  ∧ msg ≠ C!NOT_MESSAGE
            Master has changed, client must restart
        ∧ msg.masterId ≠ backup[msg.backupId].masterId
        ∧ C!ReplaceMsg(msg, [from ↦ "b",
                             to ↦ "c",
                             clientId   ↦ msg.clientId,
                             masterId ↦ backup[msg.backupId].masterId,
                             backupId ↦ msg.backupId,
                             value ↦ 0,
                             tag ↦ "newMasterId"])
  ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩

C_HandlingBackupDone ≜
  Client receiving "done" from backup. Replication completed
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToClient("b", "backupDone")
    IN  ∧ msg ≠ C!NOT_MESSAGE
        ∧ C!RecvMsg(msg)
        ∧ clients' = [clients EXCEPT ![msg.clientId].phase = C!PH2_COMPLETED]
                  if all clients completed, then terminate the execution successfully
        ∧ IF ∀ c ∈ C!CLIENT_ID : clients'[c].phase = C!PH2_COMPLETED
          THEN exec_state' = "success"
          ELSE exec_state' = exec_state
  ∧ UNCHANGED ⟨master, backup, killed⟩

──────────────────────────────────────────────────────────────────────────

C_HandlingMasterDoFailed ≜
  Client received the system's notification of a dead master, and is requesting the backup to return
  the new master info
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToWithTag("m", C!INST_STATUS_LOST, "masterDo")
        knownBackup ≜ IF msg ≠ C!NOT_MESSAGE
                      THEN C!FindBackup(C!INST_STATUS_ACTIVE)
                      ELSE C!NOT_BACKUP
    IN  ∧ msg ≠ C!NOT_MESSAGE
        ∧ IF knownBackup = C!NOT_BACKUP
```

```
260             THEN ∧ C!RecvMsg(msg)
261                  ∧ exec_state′ = "fatal"
262                  ∧ clients′ = [clients EXCEPT ![msg.clientId].phase = C!PH2_COMPLETED_FATAL]
263             ELSE ∧ C!ReplaceMsg(msg, [from ↦ "c",
264                                       to ↦ "b",
265                                       clientId ↦ msg.clientId,
266                                        send the client's master knowledge,
267                                        to force the backup to not respond until rereplication
268                                       masterId ↦ clients[msg.clientId].masterId,
269                                       backupId ↦ knownBackup.id,
270                                       value ↦ 0,
271                                       tag ↦ "backupGetNewMaster"])
272                  ∧ exec_state′ = exec_state
273                  ∧ clients′ = clients
274   ∧ UNCHANGED ⟨master, backup, killed⟩

276 C_HandlingBackupDoFailed ≜
    Client received the system's notification of a dead backup, and is requesting the master to return
    the new backup info
281   ∧ exec_state = "running"
282   ∧ LET msg ≜ C!FindMessageToWithTag("b", C!INST_STATUS_LOST, "backupDo")
283     IN  ∧ msg ≠ C!NOT_MESSAGE
284         ∧ C!ReplaceMsg(msg, [from ↦ "c",
285                              to ↦ "m",
286                              clientId ↦ msg.clientId,
287                              masterId ↦ clients[msg.clientId].masterId,
288                               send the client's backup knowledge,
289                               to force the master to not respond until rereplication
290                              backupId ↦ clients[msg.clientId].backupId,
291                              value ↦ 0,
292                              tag ↦ "masterGetNewBackup"])
293   ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩

295 ├──────────────────────────────────────────────────────────────────────┤
296 M_GettingNewBackup ≜
    Master responding to client with updated backup identity
300   ∧ exec_state = "running"
301   ∧ LET msg ≜ C!FindMessageToWithTag("m", C!INST_STATUS_ACTIVE, "masterGetNewBackup")
302     IN  ∧ msg ≠ C!NOT_MESSAGE
303           master must not respond until it recovers the dead backup
304         ∧ msg.backupId ≠ master[msg.masterId].backupId
305         ∧ C!ReplaceMsg(msg, [from ↦ "m",
306                              to ↦ "c",
307                              clientId ↦ msg.clientId,
308                              masterId ↦ msg.masterId,
```

```
                                    backupId ↦ master[msg.masterId].backupId,
                                    value ↦ 0,
                                    tag ↦ "newBackupId"])
  ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩

B_GettingNewMaster ≜
  Backup responding to client with updated master identity
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToWithTag("b", C!INST_STATUS_ACTIVE, "backupGetNewMaster")
    IN  ∧ msg ≠ C!NOT_MESSAGE
          backup must not respond until it recovers the dead master
        ∧ msg.masterId ≠ backup[msg.backupId].masterId
        ∧ C!ReplaceMsg(msg, [from ↦ "b",
                             to ↦ "c",
                             clientId ↦ msg.clientId,
                             masterId ↦ backup[msg.backupId].masterId,
                             backupId ↦ msg.backupId,
                             value ↦ 0,
                             tag ↦ "newMasterId"])
  ∧ UNCHANGED ⟨exec_state, clients, master, backup, killed⟩

──────────────────────────────────────────────────────────────────────

C_HandlingBackupGetNewMasterFailed ≜
  The client handling the failure of the backup, when the client asked the backup to return the
  new master identity. The client mannually searches for the master. If manual search does not
  find a master, a fatal error occurs. Otherwise, the client updates it's masterId and eventually
  restarts. Restarting is safe because this action is reached only if "masterDo" fails
  ∧ exec_state = "running"
  ∧ LET msg ≜ C!FindMessageToWithTag("b", C!INST_STATUS_LOST, "backupGetNewMaster")
        foundMaster ≜ C!FindMaster(C!INST_STATUS_ACTIVE)
    IN  ∧ msg ≠ C!NOT_MESSAGE
        ∧ C!RecvMsg(msg)
        ∧ IF foundMaster = C!NOT_MASTER  no live master found
          THEN ∧ exec_state' = "fatal"
               ∧ clients' = [clients EXCEPT ![msg.clientId].phase = C!PH2_COMPLETED_FATAL]
          ELSE ∧ exec_state' = exec_state
                 at this point, the live master must have been changed
               ∧ foundMaster.id ≠ clients[msg.clientId].masterId
                 change status to pending to be eligible for restart
               ∧ clients' = [clients EXCEPT ![msg.clientId].masterId = foundMaster.id,
                                            ![msg.clientId].phase = C!PH1_PENDING]
  ∧ UNCHANGED ⟨master, backup, killed⟩

C_HandlingMasterGetNewBackupFailed ≜
  The client handling the failure of the master when the client asked the master to return the
  new backup identity. The failure of the master is fatal. If a recovered master exists we should
  not search for it, because it may have the old version before masterDone.
```

```
366     ∧ exec_state = "running"
367     ∧ LET msg ≜ C!FindMessageToWithTag("m", C!INST_STATUS_LOST, "masterGetNewBackup")
368       IN  ∧ msg ≠ C!NOT_MESSAGE
369           ∧ exec_state' = "fatal"
370           ∧ clients' = [clients EXCEPT ![msg.clientId].phase = C!PH2_COMPLETED_FATAL]
371           ∧ C!RecvMsg(msg)
372     ∧ UNCHANGED ⟨master, backup, killed⟩

374 ├──────────────────────────────────────────────────────────────────────┤
375 C_UpdatingBackupId ≜
376     ∧ exec_state = "running"
377     ∧ LET msg ≜ C!FindMessageToClient("m", "newBackupId")
378       IN  ∧ msg ≠ C!NOT_MESSAGE  receive new backup identity, and complete request,
379                                  don't restart, master is alive and up to date
380           ∧ C!RecvMsg(msg)
381           ∧ clients' = [clients EXCEPT ![msg.clientId].backupId = msg.backupId,
382                                        ![msg.clientId].phase = C!PH2_COMPLETED]
383             if all clients completed, then terminate the execution successfully
384           ∧ IF ∀ c ∈ C!CLIENT_ID : clients'[c].phase = C!PH2_COMPLETED
385             THEN exec_state' = "success"
386             ELSE exec_state' = exec_state
387     ∧ UNCHANGED ⟨master, backup, killed⟩

389 C_UpdatingMasterId ≜
    Client receiving a new master identify from a live backup and is preparing to restart by changing
    its phase to pending
394     ∧ exec_state = "running"
395     ∧ LET msg ≜ C!FindMessageToClient("b", "newMasterId")
396       IN  ∧ msg ≠ C!NOT_MESSAGE
397           ∧ C!RecvMsg(msg)
398           ∧ clients' = [clients EXCEPT ![msg.clientId].masterId = msg.masterId,
399                                        ![msg.clientId].phase = C!PH1_PENDING]
400     ∧ UNCHANGED ⟨exec_state, master, backup, killed⟩
401 ├──────────────────────────────────────────────────────────────────────┤
402 M_CreatingNewBackup ≜
    Master creating a new backup using its own exec_state. Master does not process any client
    requests during recovery
407     ∧ exec_state = "running"
408     ∧ LET activeM ≜ C!FindMaster(C!INST_STATUS_ACTIVE)
409           activeB ≜ C!FindBackup(C!INST_STATUS_ACTIVE)
410           lostB ≜ C!LastLostBackup
411       IN  ∧ activeM ≠ C!NOT_MASTER  active master exists
412           ∧ activeB = C!NOT_BACKUP  active backup does not exist
413           ∧ lostB ≠ C!NOT_BACKUP  a lost backup exists
414           ∧ LET newBackupId ≜ lostB.id + 1  new backup id is the following id of the dead backup
```

```
                IN    ∧ newBackupId ≤ C!MAX_INSTANCE_ID
                      ∧ backup′ = [backup EXCEPT ![newBackupId].status = C!INST_STATUS_ACTIVE,
                                                 ![newBackupId].masterId = activeM.id,
                                                 ![newBackupId].value = activeM.value,
                                                 ![newBackupId].version = activeM.version]
                      ∧ master′ = [master EXCEPT ![activeM.id].backupId = newBackupId ]
    ∧ UNCHANGED ⟨exec_state, clients, msgs, killed⟩

B_CreatingNewMaster ≜
    Backup creating a new master using its own exec_state. Backup does not process any client
    requests during recovery
    ∧ exec_state = "running"
    ∧ LET activeM ≜ C!FindMaster(C!INST_STATUS_ACTIVE)
          activeB ≜ C!FindBackup(C!INST_STATUS_ACTIVE)
          lostM ≜ C!LastLostMaster
      IN  ∧ activeM = C!NOT_MASTER  active master does not exist
          ∧ activeB ≠ C!NOT_BACKUP  active backup exists
          ∧ lostM ≠ C!NOT_MASTER  a lost master exists
          ∧ LET newMasterId ≜ lostM.id + 1
            IN  ∧ newMasterId ≤ C!MAX_INSTANCE_ID
                ∧ master′ = [master EXCEPT ![newMasterId].status = C!INST_STATUS_ACTIVE,
                                           ![newMasterId].backupId = activeB.id,
                                           ![newMasterId].value = activeB.value,
                                           ![newMasterId].version = activeB.version]
                ∧ backup′ = [backup EXCEPT ![activeB.id].masterId = newMasterId ]
    ∧ UNCHANGED ⟨exec_state, clients, msgs, killed⟩

Next ≜
    ∨ E_KillingMaster
    ∨ E_KillingBackup
    ∨ C_Starting
    ∨ M_Doing
    ∨ C_HandlingMasterDone
    ∨ B_Doing
    ∨ B_DetectingOldMasterId
    ∨ C_HandlingBackupDone
    ∨ C_HandlingMasterDoFailed
    ∨ C_HandlingBackupDoFailed
    ∨ M_GettingNewBackup
    ∨ B_GettingNewMaster
    ∨ C_HandlingBackupGetNewMasterFailed
    ∨ C_HandlingMasterGetNewBackupFailed
    ∨ C_UpdatingBackupId
    ∨ C_UpdatingMasterId
    ∨ M_CreatingNewBackup
```

$\lor$ *B_CreatingNewMaster*

Liveness $\triangleq$

$\land \mathrm{WF}_{Vars}(E\_KillingMaster)$
$\land \mathrm{WF}_{Vars}(E\_KillingBackup)$
$\land \mathrm{WF}_{Vars}(C\_Starting)$
$\land \mathrm{WF}_{Vars}(M\_Doing)$
$\land \mathrm{WF}_{Vars}(C\_HandlingMasterDone)$
$\land \mathrm{WF}_{Vars}(B\_Doing)$
$\land \mathrm{WF}_{Vars}(B\_DetectingOldMasterId)$
$\land \mathrm{WF}_{Vars}(C\_HandlingBackupDone)$
$\land \mathrm{WF}_{Vars}(C\_HandlingMasterDoFailed)$
$\land \mathrm{WF}_{Vars}(C\_HandlingBackupDoFailed)$
$\land \mathrm{WF}_{Vars}(M\_GettingNewBackup)$
$\land \mathrm{WF}_{Vars}(B\_GettingNewMaster)$
$\land \mathrm{WF}_{Vars}(C\_HandlingBackupGetNewMasterFailed)$
$\land \mathrm{WF}_{Vars}(C\_HandlingMasterGetNewBackupFailed)$
$\land \mathrm{WF}_{Vars}(C\_UpdatingBackupId)$
$\land \mathrm{WF}_{Vars}(C\_UpdatingMasterId)$
$\land \mathrm{WF}_{Vars}(M\_CreatingNewBackup)$
$\land \mathrm{WF}_{Vars}(B\_CreatingNewMaster)$

Specification

$Spec \triangleq Init \land \Box[Next]_{Vars} \land Liveness$

THEOREM $Spec \Rightarrow \Box(TypeOK \land StateOK)$
