MODULE *LogSync*

EXTENDS *TLC*, *Naturals*, *Sequences*

CONSTANTS *Key*, *WatchClient*, *nil*

state is in-memory data
db is the same data but on the *db*
watch_chan is the receive channel for client

VARIABLES *pc*, *current_key*, *db*,
state, *state_seq*, *next_log*, *next_seq*, *wait_list*,
watch_pc, *watch_keys*, *watch_chan*, *watch_seq*,
watch_log_index, *watch_state*, *watch_local_key*,
num_client_restart, *num_main_restart*, *num_delete_state*

$main\_vars \triangleq \langle pc, current\_key, db \rangle$

$watch\_vars \triangleq \langle watch\_pc, watch\_keys, watch\_chan,$
$watch\_seq, watch\_log\_index, watch\_state, watch\_local\_key \rangle$

$server\_vars \triangleq \langle state, state\_seq, next\_log, next\_seq, wait\_list \rangle$

$aux\_vars \triangleq \langle num\_client\_restart, num\_main\_restart, num\_delete\_state \rangle$

$vars \triangleq \langle main\_vars, server\_vars, watch\_vars, aux\_vars \rangle$

$max\_log\_size \triangleq 3$

$max\_client\_restart \triangleq 1$
$max\_main\_restart \triangleq 1$
$max\_delete\_state \triangleq 1$

$Status \triangleq \{\text{“Running”}, \text{“Completed”}, \text{“Gone”}\}$

$LogEntry \triangleq 20 \,..\, 30$

$Info \triangleq [logs : Seq(LogEntry), status : Status]$

$NullInfo \triangleq Info \cup \{nil\}$

$NullKey \triangleq Key \cup \{nil\}$

$NullLogEntry \triangleq LogEntry \cup \{nil\}$

$Event \triangleq [$
$type : \{\text{“AddLog”}, \text{“Finished”}, \text{“JobGone”}\},$
$key : Key, line : NullLogEntry]$

$NullEvent \triangleq Event \cup \{nil\}$

$Channel \triangleq [status : \{\text{"Empty"}, \text{"Ready"}, \text{"Consumed"}\}, data : NullEvent]$

$StateSeq \triangleq 100 \,..\, 120$

$WatchState \triangleq \{\text{"Init"}, \text{"AddToWaitList"}, \text{"WaitOnChan"}, \text{"UpdateDB"}\}$

$TypeOK \triangleq$
$\quad \land \quad pc \in \{\text{"Init"}, \text{"PushJob"}\}$
$\quad \land \quad current\_key \in NullKey$
$\quad \land \quad db \in [Key \to NullInfo]$

$\quad \land \quad state \in [Key \to NullInfo]$
$\quad \land \quad state\_seq \in [Key \to StateSeq]$
$\quad \land \quad next\_log \in LogEntry$
$\quad \land \quad next\_seq \in StateSeq$
$\quad \land \quad wait\_list \in [Key \to \text{SUBSET } WatchClient]$

$\quad \land \quad watch\_pc \in [WatchClient \to WatchState]$
$\quad \land \quad watch\_keys \in [WatchClient \to \text{SUBSET } Key]$
$\quad \land \quad watch\_chan \in [WatchClient \to Channel]$
$\quad \land \quad watch\_seq \in [WatchClient \to [Key \to StateSeq]]$
$\quad \land \quad watch\_log\_index \in [WatchClient \to [Key \to Nat]]$
$\quad \land \quad watch\_state \in [WatchClient \to [Key \to NullInfo]]$
$\quad \land \quad watch\_local\_key \in [WatchClient \to NullKey]$

$\quad \land \quad num\_client\_restart \in 0 \,..\, max\_client\_restart$
$\quad \land \quad num\_main\_restart \in 0 \,..\, max\_main\_restart$
$\quad \land \quad num\_delete\_state \in 0 \,..\, max\_delete\_state$

$consumed\_chan \triangleq [status \mapsto \text{"Consumed"}, data \mapsto nil]$

$Init \triangleq$
$\quad \land pc = \text{"Init"}$
$\quad \land current\_key = nil$
$\quad \land db = [k \in Key \mapsto nil]$

$\quad \land state = [k \in Key \mapsto nil]$
$\quad \land state\_seq = [k \in Key \mapsto 100]$
$\quad \land next\_log = 20$
$\quad \land wait\_list = [k \in Key \mapsto \{\}]$
$\quad \land next\_seq = 100$

$\quad \land watch\_pc = [c \in WatchClient \mapsto \text{"Init"}]$
$\quad \land watch\_keys = [c \in WatchClient \mapsto \{\}]$
$\quad \land watch\_chan = [c \in WatchClient \mapsto consumed\_chan]$
$\quad \land watch\_seq = [c \in WatchClient \mapsto [k \in Key \mapsto 100]]$
$\quad \land watch\_log\_index = [c \in WatchClient \mapsto [k \in Key \mapsto 0]]$

$\land\ watch\_state = [c \in WatchClient \mapsto [k \in Key \mapsto nil]]$
$\land\ watch\_local\_key = [c \in WatchClient \mapsto nil]$

$\land\ num\_client\_restart = 0$
$\land\ num\_main\_restart = 0$
$\land\ num\_delete\_state\ = 0$

$newJob \triangleq [logs \mapsto \langle\rangle,\ status \mapsto$ "Running"$]$

$AddDBJob(k) \triangleq$
$\land\ pc =$ "Init"
$\land\ db[k] = nil$
$\land\ pc' =$ "PushJob"
$\land\ current\_key' = k$
$\land\ db' = [db$ EXCEPT $![k] = newJob]$
$\land$ UNCHANGED $server\_vars$
$\land$ UNCHANGED $watch\_vars$
$\land$ UNCHANGED aux_vars

$updateStateSeq(k) \triangleq$
$\land\ next\_seq' = next\_seq + 1$
$\land\ state\_seq' = [state\_seq$ EXCEPT $![k] = next\_seq']$

$PushJob \triangleq$
$\land\ pc =$ "PushJob"
$\land\ pc' =$ "Init"
$\land\ current\_key' = nil$
$\land\ state' = [state$ EXCEPT $![current\_key] = db[current\_key]]$
$\land$ UNCHANGED $\langle next\_seq,\ state\_seq\rangle$
$\land$ UNCHANGED $wait\_list$
$\land$ UNCHANGED db
$\land$ UNCHANGED $next\_log$
$\land$ UNCHANGED $watch\_vars$
$\land$ UNCHANGED aux_vars

$canPushKeyToClient(k,\ c,\ old\_watch\_ch) \triangleq$
$\land\ old\_watch\_ch[c].status =$ "Empty"
$\land\ c \in wait\_list'[k]$
$\land\ watch\_seq[c][k] < state\_seq'[k]$

$pushToClientChan(k,\ c,\ old\_watch\_ch) \triangleq$
LET
$last\_index \triangleq watch\_log\_index[c][k]$
$state\_index \triangleq Len(state'[k].logs)$

```
        new_line ≜ state'[k].logs[last_index + 1]

        add_event ≜ [
            type ↦ "AddLog",
            key ↦ k,
            line ↦ new_line]

        finished_or_gone ≜
            IF state'[k].status = "Gone"
                THEN "JobGone"
                ELSE "Finished"

        finish_event ≜ [
            type ↦ finished_or_gone,
            key ↦ k,
            line ↦ nil]

        is_running ≜ state'[k].status = "Running"

        add_log_cond ≜ is_running ∨ last_index < state_index

        update_seq_cond ≜
            IF last_index ≥ state_index
                THEN TRUE
                ELSE IF last_index + 1 ≥ state_index ∧ is_running
                    THEN TRUE
                    ELSE FALSE

        new_event ≜
            IF add_log_cond
                THEN add_event
                ELSE finish_event

        new_state ≜ [status ↦ "Ready", data ↦ new_event]
    IN
        ∧ watch_chan' = [old_watch_ch EXCEPT ![c] = new_state]
        ∧ watch_log_index' = [watch_log_index EXCEPT ![c][k] = last_index + 1]
        ∧ IF update_seq_cond
            THEN watch_seq' = [watch_seq EXCEPT ![c][k] = state_seq'[k]]
            ELSE UNCHANGED watch_seq

pushToClientOrDoNothing(c, old_watch_ch) ≜
    LET
        doNothing ≜
            ∧ ∀ k ∈ Key : ¬canPushKeyToClient(k, c, old_watch_ch)
            ∧ watch_chan' = old_watch_ch
            ∧ UNCHANGED ⟨watch_seq, watch_log_index⟩
```

```
    IN
    ∨ ∃ k ∈ Key :
        ∧ canPushKeyToClient(k, c, old_watch_ch)
        ∧ pushToClientChan(k, c, old_watch_ch)
    ∨ doNothing

pushKeyOrDoNothing(k) ≜
    LET
        doPush ≜
            ∃ c ∈ WatchClient :
                ∧ canPushKeyToClient(k, c, watch_chan)
                ∧ pushToClientChan(k, c, watch_chan)

        doNothing ≜
            ∧ ∀ c ∈ WatchClient : ¬canPushKeyToClient(k, c, watch_chan)
            ∧ UNCHANGED ⟨watch_chan, watch_seq, watch_log_index⟩
    IN
        doPush ∨ doNothing

ProduceLog(k) ≜
    ∧ state[k] ≠ nil
    ∧ state[k].status = "Running"
    ∧ Len(state[k].logs) < max_log_size

    ∧ next_log' = next_log + 1
    ∧ state' = [state EXCEPT ![k].logs = Append(@, next_log')]

    ∧ updateStateSeq(k)

    ∧ UNCHANGED wait_list
    ∧ pushKeyOrDoNothing(k)

    ∧ UNCHANGED main_vars
    ∧ UNCHANGED ⟨watch_pc, watch_keys, watch_state, watch_local_key⟩
    ∧ UNCHANGED aux_vars

FinishJob(k) ≜
    ∧ state[k] ≠ nil
    ∧ state[k].status = "Running"
    ∧ state' = [state EXCEPT ![k].status = "Completed"]
    ∧ updateStateSeq(k)

    ∧ UNCHANGED wait_list
    ∧ pushKeyOrDoNothing(k)

    ∧ UNCHANGED next_log
    ∧ UNCHANGED main_vars
```

```
    ∧ UNCHANGED ⟨watch_pc, watch_keys, watch_state, watch_local_key⟩
    ∧ UNCHANGED aux_vars

new_chan ≜ [status ↦ "Empty", data ↦ nil]

NewWatchChan(c) ≜
    LET
        new_watch_ch ≜ [watch_chan EXCEPT ![c] = new_chan]
    IN
        ∧ watch_pc[c] = "Init"
        ∧ watch_pc' = [watch_pc EXCEPT ![c] = "WaitOnChan"]

        ∧ UNCHANGED server_vars
        ∧ pushToClientOrDoNothing(c, new_watch_ch)

        ∧ UNCHANGED ⟨watch_keys, watch_state, watch_local_key⟩
        ∧ UNCHANGED main_vars
        ∧ UNCHANGED aux_vars


active_keys ≜
    LET
        db_set ≜ {k ∈ Key : db[k] ≠ nil ∧ db[k].status = "Running"}
    IN
        db_set \ {current_key}

UpdateWatchKeys(c) ≜
    ∧ watch_keys[c] ≠ active_keys
    ∧ watch_keys' = [watch_keys EXCEPT ![c] = active_keys]
    ∧ UNCHANGED ⟨watch_pc, watch_chan, watch_seq, watch_log_index, watch_state⟩
    ∧ UNCHANGED watch_local_key
    ∧ UNCHANGED main_vars
    ∧ UNCHANGED server_vars
    ∧ UNCHANGED aux_vars


updateServerWaitList(c) ≜
    LET
        old_set(k) ≜ wait_list[k]
        new_set(k) ≜
            IF k ∈ watch_keys[c]
                THEN old_set(k) ∪ {c}
                ELSE old_set(k) \ {c}
    IN
        wait_list' = [k ∈ Key ↦ new_set(k)]

serverWatchClientKeys(c) ≜ {k ∈ Key : c ∈ wait_list[k]}
```

```
createPlaceHolderStateForWaitList ≜
    LET
        in_wait_list(k) ≜ wait_list'[k] ≠ {}

        keysWithNilState ≜
            {k ∈ Key : in_wait_list(k) ∧ state[k] = nil}

        new_state_fn(k) ≜
            IF k ∈ keysWithNilState
                THEN [logs ↦ ⟨⟩, status ↦ "Gone"]
                ELSE state[k]

        new_seq_fn(k) ≜
            IF k ∈ keysWithNilState
                THEN next_seq'
                ELSE state_seq[k]

        update_state ≜
            ∧ next_seq' = next_seq + 1
            ∧ state' = [k ∈ Key ↦ new_state_fn(k)]
            ∧ state_seq' = [k ∈ Key ↦ new_seq_fn(k)]

        do_nothing ≜
            UNCHANGED ⟨state, next_seq, state_seq⟩
    IN
        IF keysWithNilState ≠ {}
            THEN update_state
            ELSE do_nothing


AddToWaitList(c) ≜
    ∧ watch_keys[c] ≠ serverWatchClientKeys(c)
    ∧ updateServerWaitList(c)

    ∧ createPlaceHolderStateForWaitList
    ∧ pushToClientOrDoNothing(c, watch_chan)

    ∧ UNCHANGED ⟨watch_pc, watch_keys, watch_state, watch_local_key⟩
    ∧ UNCHANGED main_vars
    ∧ UNCHANGED next_log
    ∧ UNCHANGED aux_vars


updateStateFromChan(c) ≜
    LET
        k ≜ watch_chan[c].data.key
        type ≜ watch_chan[c].data.type
        log_line ≜ watch_chan[c].data.line
```

```
        old_state ≜ watch_state[c][k]

        old_logs ≜
            IF old_state = nil
                THEN ⟨⟩
                ELSE old_state.logs

        new_state ≜
            [logs ↦ Append(old_logs, log_line), status ↦ "Running"]

        do_add_log ≜
            ∧ watch_state' = [
                    watch_state EXCEPT ![c][k] = new_state]
            ∧ UNCHANGED watch_local_key
            ∧ watch_pc' = [watch_pc EXCEPT ![c] = "Init"]

        new_status ≜
            IF type = "JobGone"
                THEN "Gone"
                ELSE "Completed"

        do_complete ≜
            ∧ watch_state' = [
                watch_state EXCEPT
                    ![c][k] = [logs ↦ old_logs, status ↦ new_status]]
            ∧ watch_local_key' = [watch_local_key EXCEPT ![c] = k]
            ∧ watch_pc' = [watch_pc EXCEPT ![c] = "UpdateDB"]
    IN
        IF type = "AddLog"
            THEN do_add_log
            ELSE do_complete

ConsumeWatchChan(c) ≜
    ∧ watch_pc[c] = "WaitOnChan"
    ∧ watch_chan[c].status = "Ready"

    ∧ watch_chan' = [
            watch_chan EXCEPT
                ![c].status = "Consumed",
                ![c].data = nil]

    ∧ updateStateFromChan(c)

    ∧ UNCHANGED ⟨watch_keys, watch_seq, watch_log_index⟩
    ∧ UNCHANGED main_vars
    ∧ UNCHANGED server_vars
    ∧ UNCHANGED aux_vars
```

$UpdateDB(c) \triangleq$
LET
$k \triangleq watch\_local\_key[c]$
IN
$\land\ watch\_pc[c] = \text{“UpdateDB”}$
$\land\ watch\_pc' = [watch\_pc \text{ EXCEPT } ![c] = \text{“Init”}]$
$\land\ db' = [db \text{ EXCEPT } ![k] = watch\_state[c][k]]$
$\land\ watch\_local\_key' = [watch\_local\_key \text{ EXCEPT } ![c] = nil]$
$\land \text{ UNCHANGED } \langle watch\_keys, watch\_chan, watch\_seq\rangle$
$\land \text{ UNCHANGED } \langle watch\_log\_index, watch\_state\rangle$
$\land \text{ UNCHANGED } server\_vars$
$\land \text{ UNCHANGED } \langle pc, current\_key\rangle$
$\land \text{ UNCHANGED } aux\_vars$

$ClientRestart(c) \triangleq$
$\land\ num\_client\_restart < max\_client\_restart$
$\land\ num\_client\_restart' = num\_client\_restart + 1$
$\land\ watch\_chan' = [watch\_chan \text{ EXCEPT } ![c] = consumed\_chan]$
$\land\ watch\_keys' = [watch\_keys \text{ EXCEPT } ![c] = \{\}]$
$\land\ watch\_local\_key' = [watch\_local\_key \text{ EXCEPT } ![c] = nil]$
$\land\ watch\_log\_index' = [watch\_log\_index \text{ EXCEPT } ![c] = [k \in Key \mapsto 0]]$
$\land\ watch\_seq' = [watch\_seq \text{ EXCEPT } ![c] = [k \in Key \mapsto 100]]$
$\land\ watch\_state' = [watch\_state \text{ EXCEPT } ![c] = [k \in Key \mapsto nil]]$
$\land\ watch\_pc' = [watch\_pc \text{ EXCEPT } ![c] = \text{“Init”}]$
$\land \text{ UNCHANGED } server\_vars$
$\land \text{ UNCHANGED } main\_vars$
$\land \text{ UNCHANGED } \langle num\_main\_restart, num\_delete\_state\rangle$

$MainRestart \triangleq$
$\land\ num\_main\_restart < max\_main\_restart$
$\land\ num\_main\_restart' = num\_main\_restart + 1$
$\land\ current\_key' = nil$
$\land\ pc' = \text{“Init”}$
$\land \text{ UNCHANGED } db$
$\land \text{ UNCHANGED } \langle num\_client\_restart, num\_delete\_state\rangle$
$\land \text{ UNCHANGED } server\_vars$
$\land \text{ UNCHANGED } watch\_vars$

$DeleteRandomKeyInState(k) \triangleq$
$\land\ num\_delete\_state < max\_delete\_state$
$\land\ num\_delete\_state' = num\_delete\_state + 1$
$\land\ state[k] \neq nil$
$\land\ state' = [state \text{ EXCEPT } ![k] = nil]$

$\land\ state\_seq' = [state\_seq \text{ EXCEPT } ![k] = 100]$
$\land\ wait\_list' = [wait\_list \text{ EXCEPT } ![k] = \{\}]$

$\land$ UNCHANGED $\langle next\_log, next\_seq \rangle$
$\land$ UNCHANGED $\langle num\_client\_restart, num\_main\_restart \rangle$
$\land$ UNCHANGED $main\_vars$
$\land$ UNCHANGED $watch\_vars$

$statusIsFinished(st) \triangleq$
$\lor\ st =$ "Completed"
$\lor\ st =$ "Gone"

$TerminateCond \triangleq$
$\land \forall\, k \in Key : db[k] \neq nil \land statusIsFinished(db[k].status)$
$\land \forall\, k \in Key : state[k] \neq nil \Rightarrow statusIsFinished(state[k].status)$
$\land \forall\, c \in WatchClient :$
$\land\ watch\_pc[c] =$ "WaitOnChan"
$\land\ watch\_keys[c] = active\_keys$
$\land\ watch\_keys[c] = serverWatchClientKeys(c)$
$\land\ watch\_chan[c].status =$ "Empty"

$Terminated \triangleq$
$\land\ TerminateCond$
$\land$ UNCHANGED $vars$

$Next \triangleq$
$\lor \exists\, k \in Key :$
$\lor\ AddDBJob(k)$
$\lor\ ProduceLog(k)$
$\lor\ FinishJob(k)$
$\lor\ DeleteRandomKeyInState(k)$
$\lor\ PushJob$

$\lor \exists\, c \in WatchClient :$
$\lor\ NewWatchChan(c)$
$\lor\ UpdateWatchKeys(c)$
$\lor\ AddToWaitList(c)$
$\lor\ ConsumeWatchChan(c)$
$\lor\ UpdateDB(c)$
$\lor\ ClientRestart(c)$

$\lor\ MainRestart$

$\lor\ Terminated$

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

$FairSpec \triangleq Spec \land \mathrm{WF}_{vars}(Next)$

$AlwaysTerminate \triangleq \Diamond TerminateCond$

$AllJobsMustBeFinished \triangleq$
$\quad TerminateCond \Rightarrow$
$\quad\quad \forall\, k \in Key : db[k] \neq nil \land statusIsFinished(db[k].status)$

$infoEqual(db\_val,\ state\_val) \triangleq$
$\quad \land\ db\_val.status \in \{\text{“Completed”},\ \text{“Gone”}\}$
$\quad \land\ state\_val.status \in \{\text{“Completed”},\ \text{“Gone”}\}$

$\quad \land\ state\_val.status = \text{“Completed”} \Rightarrow db\_val.logs = state\_val.logs$
$\quad \land\ state\_val.status = \text{“Completed”} \Rightarrow db\_val.status = \text{“Completed”}$

$DBShouldSameAsMem \triangleq$
$\quad TerminateCond \Rightarrow$
$\quad\quad \forall\, k \in Key : state[k] \neq nil \Rightarrow infoEqual(db[k],\ state[k])$

$DBShouldSameAsMemWhenNoRestart \triangleq$
$\quad \text{LET}$
$\quad\quad cond \triangleq$
$\quad\quad\quad \land\ TerminateCond$
$\quad\quad\quad \land\ num\_main\_restart = 0$
$\quad\quad\quad \land\ num\_delete\_state\ \ = 0$
$\quad \text{IN}$
$\quad\quad cond \Rightarrow \forall\, k \in Key : state[k] = db[k] \land db[k].status = \text{“Completed”}$

$StateAlwaysMatchWaitList \triangleq$
$\quad \forall\, k \in Key :$
$\quad\quad wait\_list[k] \neq \{\} \Rightarrow state[k] \neq nil$

$StateAlwaysMatchSeq \triangleq$
$\quad \forall\, k \in Key :$
$\quad\quad state[k] = nil \Rightarrow state\_seq[k] = 100$

$channelInitByClient(c) \triangleq$
$\quad \land\ watch\_chan[c].status = \text{“Consumed”}$
$\quad \land\ watch\_chan[c].data = nil$

$channelInit \triangleq \forall\, c \in WatchClient : channelInitByClient(c)$

$channelNextByClient(c) \triangleq$
$\quad \lor\ \land\ watch\_chan[c].status = \text{“Empty”}$

$\land\ watch\_chan'[c].status = \text{“Ready”}$
$\land\ watch\_chan'[c].data \neq nil$

$\lor\ \land\ watch\_chan[c].status = \text{“Consumed”}$
$\land\ watch\_chan'[c].status = \text{“Empty”}$
$\land\ watch\_chan'[c].data = nil$

$\lor\ \land\ watch\_chan[c].status = \text{“Consumed”}$
$\land\ watch\_chan'[c].status = \text{“Ready”}$
$\land\ watch\_chan'[c].data \neq nil$

$\lor\ \land\ \lor\ watch\_chan[c].status = \text{“Ready”}$
$\lor\ watch\_chan[c].status = \text{“Empty”}$
$\land\ watch\_chan'[c].status = \text{“Consumed”}$
$\land\ watch\_chan'[c].data = nil$

$channelNextActions \triangleq \exists\, c \in WatchClient : channelNextByClient(c)$

$ChannelSpec \triangleq$
$channelInit \land \Box[channelNextActions]_{watch\_chan}$

$Sym \triangleq Permutations(Key)$