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*

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

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

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

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

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

$max\_log\_size \triangleq 2$

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

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

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

$SeqMaxLen(S, n) \triangleq \text{UNION } \{[1 \,..\, m \to S] : m \in 0 \,..\, n\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$pushToClientChan(k,\ c,\ old\_watch\_ch) \triangleq$
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]

    finish_event ≜ [
        type ↦ "Finished",
        key ↦ k,
        line ↦ nil]

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

    is_nil ≜ state'[k] = nil

    add_log_cond ≜ is_running ∨ last_index < state_index

    seq_cond_non_nil ≜
        IF last_index = state_index
            THEN TRUE
            ELSE IF last_index + 1 = state_index ∧ is_running
                THEN TRUE
                ELSE FALSE

    update_seq_cond ≜
        IF is_nil
            THEN TRUE
            ELSE seq_cond_non_nil

    new_event ≜
        IF is_nil
            THEN finish_event
            ELSE 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]}

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

    ∧ UNCHANGED ⟨state, next_seq, state_seq⟩
    ∧ 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"]

        do_complete ≜
            ∧ watch_state' = [
                    watch_state EXCEPT
                        ![c][k]    = [logs ↦ old_logs, status ↦ "Completed"]]
            ∧ 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) ≜
    LET
        k ≜ watch_local_key[c]
    IN
        ∧ watch_pc[c] = "UpdateDB"
        ∧ watch_pc' = [watch_pc EXCEPT ![c] = "Init"]

        ∧ db' = [db EXCEPT ![k] = watch_state[c][k]]
        ∧ watch_local_key' = [watch_local_key EXCEPT ![c] = nil]

        ∧ UNCHANGED ⟨watch_keys, watch_chan, watch_seq⟩
        ∧ UNCHANGED ⟨watch_log_index, watch_state⟩
        ∧ UNCHANGED server_vars
        ∧ UNCHANGED ⟨pc, current_key⟩
        ∧ UNCHANGED aux_vars


ClientRestart(c) ≜
    ∧ num_client_restart < max_client_restart
    ∧ num_client_restart' = num_client_restart + 1
    ∧ watch_chan' = [watch_chan EXCEPT ![c] = consumed_chan]
    ∧ watch_keys' = [watch_keys EXCEPT ![c] = {}]
    ∧ watch_local_key' = [watch_local_key EXCEPT ![c] = nil]
    ∧ watch_log_index' = [watch_log_index EXCEPT ![c] = [k ∈ Key ↦ 0]]
    ∧ watch_seq' = [watch_seq EXCEPT ![c] = [k ∈ Key ↦ 100]]
    ∧ watch_state' = [watch_state EXCEPT ![c] = [k ∈ Key ↦ nil]]
    ∧ watch_pc' = [watch_pc EXCEPT ![c] = "Init"]
```

$\land$ UNCHANGED $server\_vars$
$\land$ UNCHANGED $main\_vars$
$\land$ UNCHANGED $\langle num\_main\_restart \rangle$

$MainRestart \triangleq$
$\land\ num\_main\_restart < max\_main\_restart$
$\land\ num\_main\_restart' = num\_main\_restart + 1$
$\land\ current\_key' = nil$
$\land\ pc' =$ "Init"
$\land$ UNCHANGED db
$\land$ UNCHANGED $\langle num\_client\_restart \rangle$
$\land$ UNCHANGED $server\_vars$
$\land$ UNCHANGED $watch\_vars$

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

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

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

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

$\lor\ MainRestart$

$\lor\ Terminated$

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

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

$$AlwaysTerminate \triangleq \Diamond TerminateCond$$

$$\begin{array}{l} AllJobsMustBeFinished \triangleq \\ \quad TerminateCond \Rightarrow \\ \qquad \forall\, k \in Key : db[k] \neq nil \wedge db[k].status = \text{“Completed”} \end{array}$$

$$\begin{array}{l} DBShouldSameAsMem \triangleq \\ \quad TerminateCond \Rightarrow \\ \qquad \forall\, k \in Key : state[k] \neq nil \Rightarrow db[k] = state[k] \end{array}$$

$$\begin{array}{l} channelInitByClient(c) \triangleq \\ \quad \wedge\, watch\_chan[c].status = \text{“Consumed”} \\ \quad \wedge\, watch\_chan[c].data = nil \end{array}$$

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

$$\begin{array}{l} channelNextByClient(c) \triangleq \\ \quad \vee\ \wedge\, watch\_chan[c].status = \text{“Empty”} \\ \qquad \wedge\, watch\_chan'[c].status = \text{“Ready”} \\ \qquad \wedge\, watch\_chan'[c].data \neq nil \\ \\ \quad \vee\ \wedge\, watch\_chan[c].status = \text{“Consumed”} \\ \qquad \wedge\, watch\_chan'[c].status = \text{“Empty”} \\ \qquad \wedge\, watch\_chan'[c].data = nil \\ \\ \quad \vee\ \wedge\, watch\_chan[c].status = \text{“Consumed”} \\ \qquad \wedge\, watch\_chan'[c].status = \text{“Ready”} \\ \qquad \wedge\, watch\_chan'[c].data \neq nil \\ \\ \quad \vee\ \wedge\ \vee\, watch\_chan[c].status = \text{“Ready”} \\ \qquad\quad \vee\, watch\_chan[c].status = \text{“Empty”} \\ \qquad \wedge\, watch\_chan'[c].status = \text{“Consumed”} \\ \qquad \wedge\, watch\_chan'[c].data = nil \end{array}$$

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

$$\begin{array}{l} ChannelSpec \triangleq \\ \quad channelInit \wedge \Box[channelNextActions]_{watch\_chan} \end{array}$$

$$Sym \triangleq Permutations(Key)$$