MODULE *PendingKeys*

EXTENDS *TLC*, *Naturals*

CONSTANTS *Key*, *Slave*, *Client*, *nil*

VARIABLES *info*, *pending*, *pc*,
slave_map,
client_slave, *client_state*, *client_keys*,
num_action

$aux\_vars \triangleq \langle client\_slave, num\_action \rangle$

$vars \triangleq \langle$
info, *pending*, *pc*,
slave_map,
client_state, *client_keys*,
aux_vars
$\rangle$

$max\_action \triangleq 7$

$Seq \triangleq 0 \,..\, 30$

$NullSlave \triangleq Slave \cup \{nil\}$

$SlaveState \triangleq [$
running : SUBSET *Key*,
latest_seq : *Seq*,
wait_list : SUBSET *Client*
$]$

$Channel \triangleq [data : \text{SUBSET } Key, status : \{\text{“Empty”}, \text{“Ready”}, \text{“Consumed”}\}]$

$ClientState \triangleq [$
chan : *Channel*,
consumed_seq : *Seq*
$]$

$init\_slave\_state \triangleq [$
$running \mapsto \{\}$,
$latest\_seq \mapsto 0$,
$wait\_list \mapsto \{\}$
$]$

$init\_client\_state \triangleq [$
$chan \mapsto [data \mapsto \{\}, status \mapsto \text{“Consumed”}]$,

$consumed\_seq \mapsto 0$
]

$TypeOK \triangleq$
$\land \quad info \in [Key \to NullSlave]$
$\land \quad pending \subseteq Key$
$\land \quad pc \in [Client \to \{\text{“Init”}, \text{“GetRunningKeys”}, \text{“WaitOnChan”}\}]$
$\land \quad client\_slave \in [Client \to Slave]$

$\land \quad slave\_map \in [Slave \to SlaveState]$
$\land \quad client\_state \in [Client \to ClientState]$
$\land \quad client\_keys \in [Client \to \text{SUBSET } Key]$
$\land \quad num\_action \in 0 \,..\, max\_action$

$Init \triangleq$
$\land\, info = [k \in Key \mapsto nil]$
$\land\, pending = \{\}$
$\land\, pc = [s \in Client \mapsto \text{“Init”}]$
$\land\, client\_slave \in [Client \to Slave]$

$\land\, slave\_map = [s \in Slave \mapsto init\_slave\_state]$
$\land\, client\_state = [c \in Client \mapsto init\_client\_state]$
$\land\, client\_keys = [c \in Client \mapsto \{\}]$
$\land\, num\_action = 0$

$allowAction \triangleq$
$\land\, num\_action < max\_action$
$\land\, num\_action' = num\_action + 1$

$pushToClient(client\_set,\ old\_state) \triangleq$
LET
$get\_slave(c) \triangleq slave\_map'[client\_slave[c]]$

$curr\_seq(c) \triangleq get\_slave(c).latest\_seq$

$can\_push(c) \triangleq$
$\land\, c \in client\_set$
$\land\, old\_state[c].chan.status = \text{“Empty”}$
$\land\, old\_state[c].consumed\_seq < curr\_seq(c)$

$new\_chan(c) \triangleq$
$[data \mapsto get\_slave(c).running,\ status \mapsto \text{“Ready”}]$

$new\_state(c) \triangleq$
$[chan \mapsto new\_chan(c),\ consumed\_seq \mapsto curr\_seq(c)]$

```
        new_state_or_unchanged(c) ≜
            IF can_push(c)
                THEN new_state(c)
                ELSE old_state[c] UNCHANGED
    IN
        [c ∈ Client ↦ new_state_or_unchanged(c)]

AddKey(k) ≜
    LET
        added ≜
            IF k ∈ pending
                THEN {}
                ELSE {k}

        do_add_key(s) ≜
            ∧ info[k] = nil
            ∧ allowAction
            ∧ info' = [info EXCEPT ![k] = s]
            ∧ slave_map' = [slave_map EXCEPT
                    ![s].latest_seq = @ + 1,
                    ![s].running = @ ∪ added]
            ∧ client_state' = pushToClient(slave_map[s].wait_list, client_state)
            ∧ UNCHANGED pending
            ∧ UNCHANGED pc
            ∧ UNCHANGED client_slave
            ∧ UNCHANGED client_keys
    IN
        ∃ s ∈ Slave : do_add_key(s)

RemoveKey(k) ≜
    LET
        do_remove_key(s) ≜
            ∧ info[k] ≠ nil
            ∧ info[k] = s
            ∧ allowAction
            ∧ info' = [info EXCEPT ![k] = nil]

            ∧ slave_map' = [slave_map EXCEPT
                    ![s].running = @ \ {k},
                    ![s].latest_seq = @ + 1]
            ∧ client_state' = pushToClient(slave_map[s].wait_list, client_state)

            ∧ UNCHANGED pending
            ∧ UNCHANGED pc
            ∧ UNCHANGED client_slave
```

```
            ∧ UNCHANGED client_keys
    IN
        ∃ s ∈ Slave : do_remove_key(s)

addPendingKeyUpdateSlaveMap(k) ≜
    LET
        s ≜ info[k]
    IN
        ∧ slave_map' = [slave_map EXCEPT
                ![s].running = @ \ {k},
                ![s].latest_seq = @ + 1
            ]
        ∧ client_state' = pushToClient(slave_map[s].wait_list, client_state)

AddPendingKey(k) ≜
    ∧ k ∉ pending
    ∧ allowAction
    ∧ pending' = pending ∪ {k}
    ∧ IF info[k] ≠ nil
        THEN addPendingKeyUpdateSlaveMap(k)
        ELSE
            ∧ UNCHANGED slave_map
            ∧ UNCHANGED client_state
    ∧ UNCHANGED info
    ∧ UNCHANGED pc
    ∧ UNCHANGED client_slave
    ∧ UNCHANGED client_keys

removePendingKeyUpdateSlaveMap(k) ≜
    LET
        s ≜ info[k]
    IN
        ∧ slave_map' = [slave_map EXCEPT
                ![s].running = @ ∪ {k},
                ![s].latest_seq = @ + 1
            ]
        ∧ client_state' = pushToClient(slave_map[s].wait_list, client_state)

RemovePendingKey(k) ≜
    ∧ k ∈ pending
    ∧ allowAction
    ∧ pending' = pending \ {k}

    ∧ IF info[k] ≠ nil
```

```
        THEN removePendingKeyUpdateSlaveMap(k)
        ELSE
            ∧ UNCHANGED slave_map
            ∧ UNCHANGED client_state

    ∧ UNCHANGED info
    ∧ UNCHANGED pc
    ∧ UNCHANGED client_slave
    ∧ UNCHANGED client_keys


InitClient(c) ≜
    LET
        s ≜ client_slave[c]
    IN
        ∧ pc[c] = "Init"
        ∧ pc' = [pc EXCEPT ![c] = "GetRunningKeys"]
        ∧ slave_map' = [slave_map EXCEPT ![s].wait_list = @ ∪ {c}]
        ∧ UNCHANGED client_state
        ∧ UNCHANGED client_keys
        ∧ UNCHANGED info
        ∧ UNCHANGED pending
        ∧ UNCHANGED aux_vars


init_channel ≜ [data ↦ {}, status ↦ "Empty"]

GetRunningKeys(c) ≜
    LET
        new channel and assign to client_state
        updated_chan ≜ [client_state EXCEPT ![c].chan = init_channel]
    IN
        ∧ pc[c] = "GetRunningKeys"
        ∧ pc' = [pc EXCEPT ![c] = "WaitOnChan"]

        ∧ UNCHANGED slave_map
        ∧ client_state' = pushToClient({c}, updated_chan)

        ∧ UNCHANGED client_keys
        ∧ UNCHANGED info
        ∧ UNCHANGED pending
        ∧ UNCHANGED aux_vars


ConsumeChan(c) ≜
    ∧ pc[c] = "WaitOnChan"
    ∧ client_state[c].chan.status = "Ready"
    ∧ pc' = [pc EXCEPT ![c] = "GetRunningKeys"]
```

$\land\ client\_state' = [client\_state \text{ EXCEPT } ![c].chan.status = \text{“Consumed”}]$
$\land\ client\_keys' = [client\_keys \text{ EXCEPT } ![c] = client\_state[c].chan.data]$
$\land$ UNCHANGED $slave\_map$
$\land$ UNCHANGED $info$
$\land$ UNCHANGED $pending$
$\land$ UNCHANGED aux_vars

$clientWaitOnChan(c) \triangleq$
$\land\ pc[c] = \text{“WaitOnChan”}$
$\land\ client\_state[c].chan.status = \text{“Empty”}$

$TerminateCond \triangleq$
$\land \forall\, c \in Client : clientWaitOnChan(c)$
$\land\ num\_action = max\_action$

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

$Next \triangleq$
$\lor \exists\, k \in Key :$
$\lor\ AddKey(k)$
$\lor\ RemoveKey(k)$
$\lor\ AddPendingKey(k)$
$\lor\ RemovePendingKey(k)$
$\lor \exists\, c \in Client :$
$\lor\ InitClient(c)$
$\lor\ GetRunningKeys(c)$
$\lor\ ConsumeChan(c)$
$\lor\ Terminated$

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

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

$AlwaysTerminate \triangleq\ \Diamond TerminateCond$

$running\_keys(s) \triangleq\ \{k \in Key : info[k] = s\} \setminus pending$

$ClientKeysMatchSharedState \triangleq$
$\forall\, c \in Client :$
$clientWaitOnChan(c) \Rightarrow$
$\land\ client\_keys[c] = running\_keys(client\_slave[c])$

$SlaveMapRunningMatchSharedState \triangleq$

$\forall\, s \in Slave :$
$slave\_map[s].running = running\_keys(s)$

$channelAction(c) \triangleq$
$\lor \land client\_state[c].chan.status =$ "Consumed"
$\land client\_state'[c].chan.status =$ "Empty"
$\lor \land client\_state[c].chan.status =$ "Consumed"
$\land client\_state'[c].chan.status =$ "Ready"
$\lor \land client\_state[c].chan.status =$ "Empty"
$\land client\_state'[c].chan.status =$ "Ready"
$\lor \land client\_state[c].chan.status =$ "Ready"
$\land client\_state'[c].chan.status =$ "Consumed"
$\lor client\_state'[c].chan = client\_state[c].chan$

$allChannelAction \triangleq$
$\forall\, c \in Client : channelAction(c)$

$ChannelSpec \triangleq$
$\Box[allChannelAction]_{client\_state}$

$ReadyAlwaysConsumed \triangleq$
$\forall\, c \in Client :$
$client\_state[c].chan.status =$ "Ready"
$\leadsto client\_state[c].chan.status =$ "Consumed"