```
─────────────────────── MODULE StateSync ───────────────────────
EXTENDS TLC, Integers, Sequences

CONSTANTS Key, Client, nil

VARIABLES server_state, wait_list, push_back_list,
    client_keys, client_states,
    next_val, server_pc, client_pc, locked,
    channels, client_channel, client_queue,
    consume_channel, outer_states

vars ≜ ⟨server_state, wait_list, push_back_list,
    client_keys, client_states,
    next_val, server_pc, client_pc, locked,
    channels, client_channel, client_queue,
    consume_channel, outer_states⟩

client_vars ≜ ⟨
    client_keys, client_states, client_pc,
    consume_channel, outer_states⟩

min_val ≜ 21
max_val ≜ 23

Value ≜ min_val . . max_val

NullValue ≜ Value ∪ {nil}

KevVal ≜ [Key → NullValue]

emptyKV ≜ [k ∈ Key ↦ nil]

Pair ≜ Key × Value

NullPair ≜ Pair ∪ {nil}

Channel ≜ [data : NullPair, status : {"Empty", "Ready", "Consumed"}]

ClientState ≜ {"Init", "ClientCheckQueue", "GetFromQueue", "WaitOnChan"}

TypeOK ≜
    ∧  server_state ∈ KevVal
    ∧  wait_list ∈ [Key → SUBSET Client]
    ∧  push_back_list ∈ [Key → SUBSET Client]
    ∧  next_val ∈ (min_val − 1) . . max_val
    ∧  client_keys ∈ [Client → SUBSET Key]
    ∧  client_states ∈ [Client → KevVal]
    ∧  server_pc ∈ {"Init", "CheckWaitList", "SetBackWaitList"}
    ∧  client_pc  ∈ [Client → ClientState]
    $\land \quad locked \in \text{BOOLEAN}$
    $\land \quad channels \in Seq(Channel)$
    $\land \quad client\_channel \in [Client \rightarrow 1 \,..\, Len(channels) \cup \{nil\}]$
    $\land \quad client\_queue \in [Client \rightarrow \text{SUBSET } Key]$
    $\land \quad consume\_channel \in [Client \rightarrow 1 \,..\, Len(channels) \cup \{nil\}]$
    $\land \quad outer\_states \in [Client \rightarrow KevVal]$

$Init \triangleq$
    $\land\, server\_state = emptyKV$
    $\land\, wait\_list = [k \in Key \mapsto \{\}]$
    $\land\, push\_back\_list = [k \in Key \mapsto \{\}]$
    $\land\, client\_keys \in [Client \rightarrow \text{SUBSET } Key]$
    $\land\, \forall\, c \in Client : client\_keys[c] \neq \{\}$ client keys should not be empty
    $\land\, client\_states = [c \in Client \mapsto emptyKV]$
    $\land\, next\_val = min\_val - 1$
    $\land\, server\_pc = \text{“Init”}$
    $\land\, client\_pc = [c \in Client \mapsto \text{“Init”}]$
    $\land\, locked = \text{FALSE}$
    $\land\, channels = \langle\rangle$
    $\land\, client\_channel = [c \in Client \mapsto nil]$
    $\land\, client\_queue = [c \in Client \mapsto client\_keys[c]]$
    $\land\, consume\_channel = [c \in Client \mapsto nil]$
    $\land\, outer\_states = [c \in Client \mapsto emptyKV]$

$waitListEmpty \triangleq$
    $\forall\, k \in Key : wait\_list[k] = \{\}$

$SetServerState(k) \triangleq$
    $\land\, next\_val < max\_val$
    $\land\, \neg locked$

    $\land\, server\_pc = \text{“Init”}$
    $\land\, \text{IF } waitListEmpty$
        THEN
            $\land\, \text{UNCHANGED } locked$
            $\land\, \text{UNCHANGED } server\_pc$
        ELSE
            $\land\, locked' = \text{TRUE}$
            $\land\, server\_pc' = \text{“CheckWaitList”}$

    $\land\, next\_val' = next\_val + 1$
    $\land\, server\_state' = [server\_state \text{ EXCEPT } ![k] = next\_val']$

    $\land\, \text{UNCHANGED } \langle wait\_list, push\_back\_list\rangle$
    $\land\, \text{UNCHANGED } channels$
    ∧ UNCHANGED client_channel
    ∧ UNCHANGED client_queue
    ∧ UNCHANGED client_vars

setOutputChan(c, k) ≜
   LET
       index ≜ client_channel[c]
       oldState ≜ channels[index]
       val ≜ server_state[k]
       newState ≜ [oldState EXCEPT !.data = ⟨k, val⟩, !.status = "Ready"]
   IN
       ∧ channels′ = [channels EXCEPT ![index] = newState]
       ∧ client_channel′ = [client_channel EXCEPT ![c] = nil]

waitListEmptyNew ≜
   ∀ k ∈ Key : wait_list′[k] = {}

handleWaitEntryNoChange(c, k) ≜
    ∧ UNCHANGED channels
    ∧ UNCHANGED client_states
    ∧ UNCHANGED client_queue
    ∧ UNCHANGED client_channel
    ∧ push_back_list′ = [push_back_list EXCEPT ![k] = @ ∪ {c}]

handleWaitEntryChanged(c, k) ≜
   IF client_channel[c] ≠ nil
        THEN
            ∧ setOutputChan(c, k)
            ∧ client_states′ = [client_states EXCEPT ![c][k] = server_state[k]]
            ∧ client_queue′ = [client_queue EXCEPT ![c] = @ ∪ {k}]
            ∧ UNCHANGED push_back_list
        ELSE
            ∧ UNCHANGED channels
            ∧ UNCHANGED client_channel
            ∧ client_queue′ = [client_queue EXCEPT ![c] = @ ∪ {k}]
            ∧ UNCHANGED client_states  TODO re check
            ∧ UNCHANGED push_back_list

ServerCheckWaitList(k, c) ≜
    ∧ server_pc = "CheckWaitList"
    ∧ c ∈ wait_list[k]
    ∧ wait_list′ = [wait_list EXCEPT ![k] = @ \ {c}]

    ∧ IF client_states[c][k] = server_state[k]
        THEN handleWaitEntryNoChange(c, k)
        $\text{ELSE}\ \ handleWaitEntryChanged(c,\ k)$

    $\land\ \text{IF}\ \ waitListEmptyNew$
        $\text{THEN}$
            $\land\ server\_pc' = \text{“SetBackWaitList”}$
            $\land\ \text{UNCHANGED}\ \ locked$
        $\text{ELSE}$
            $\land\ \text{UNCHANGED}\ \ server\_pc$
            $\land\ \text{UNCHANGED}\ \ locked$

    $\land\ \text{UNCHANGED}\ \ server\_state$
    $\land\ \text{UNCHANGED}\ \ \langle client\_keys,\ client\_pc,\ consume\_channel,\ outer\_states \rangle$
    $\land\ \text{UNCHANGED}\ \ next\_val$

$ServerSetBackWaitList\ \ \triangleq$
    $\land\ server\_pc = \text{“SetBackWaitList”}$
    $\land\ server\_pc' = \text{“Init”}$
    $\land\ locked' = \text{FALSE}$
    $\land\ wait\_list' = push\_back\_list$
    $\land\ push\_back\_list' = [k \in Key \mapsto \{\}]$
    $\land\ \text{UNCHANGED}\ \ server\_state$
    $\land\ \text{UNCHANGED}\ \ channels$
    $\land\ \text{UNCHANGED}\ \ \langle client\_channel,\ client\_queue \rangle$
    $\land\ \text{UNCHANGED}\ \ client\_vars$
    $\land\ \text{UNCHANGED}\ \ next\_val$


$clientGoto(c,\ state)\ \ \triangleq\ \ client\_pc' = [client\_pc\ \text{EXCEPT}\ ![c] = state]$

$newChannel\ \ \triangleq$
    $\land\ channels' = Append(channels,\ [data \mapsto nil,\ status \mapsto \text{“Empty”}])$

$newChannelIndex\ \ \triangleq\ \ Len(channels')$

$GetState(c)\ \ \triangleq$
    $\land\ client\_pc[c] = \text{“Init”}$
    $\land\ \neg locked$
    $\land\ locked' = \text{TRUE}$
    $\land\ newChannel$
    $\land\ client\_channel' = [client\_channel\ \text{EXCEPT}\ ![c] = newChannelIndex]$
    $\land\ clientGoto(c,\ \text{“ClientCheckQueue”})$
    $\land\ \text{UNCHANGED}\ \ \langle client\_keys,\ client\_states,\ client\_queue \rangle$
    $\land\ \text{UNCHANGED}\ \ next\_val$
    $\land\ \text{UNCHANGED}\ \ \langle server\_pc,\ server\_state \rangle$
    $\land\ \text{UNCHANGED}\ \ \langle consume\_channel,\ outer\_states \rangle$
    $\land\ \text{UNCHANGED}\ \ \langle wait\_list,\ push\_back\_list \rangle$
```

```
ClientCheckQueue(c) ≜
    ∧ client_pc[c] = "ClientCheckQueue"
    ∧ IF client_queue[c] = {}
        THEN
            ∧ clientGoto(c, "WaitOnChan")
            ∧ consume_channel′ = [consume_channel EXCEPT ![c] = client_channel[c]]
            ∧ client_channel′ = [client_channel EXCEPT ![c] = nil]
            ∧ locked′ = FALSE
            ∧ UNCHANGED client_channel
        ELSE
            ∧ clientGoto(c, "GetFromQueue")
            ∧ UNCHANGED locked
            ∧ UNCHANGED client_channel
            ∧ UNCHANGED consume_channel
    ∧ UNCHANGED channels
    ∧ UNCHANGED ⟨client_queue, client_states⟩
    ∧ UNCHANGED ⟨client_keys⟩
    ∧ UNCHANGED ⟨server_pc, server_state⟩
    ∧ UNCHANGED next_val
    ∧ UNCHANGED ⟨outer_states⟩
    ∧ UNCHANGED ⟨wait_list, push_back_list⟩

GetFromQueue(c, k) ≜
    ∧ client_pc[c] = "GetFromQueue"
    ∧ k ∈ client_queue[c]
    ∧ IF client_states[c][k] = server_state[k]
        THEN
            ∧ client_queue′ = [client_queue EXCEPT ![c] = @ \ {k}]
            ∧ clientGoto(c, "ClientCheckQueue")
            ∧ wait_list′ = [wait_list EXCEPT ![k] = @ ∪ {c}]
            ∧ UNCHANGED channels
            ∧ UNCHANGED client_channel
            ∧ UNCHANGED client_states
            ∧ UNCHANGED locked
            ∧ UNCHANGED consume_channel
            ∧ UNCHANGED push_back_list
        ELSE
            ∧ clientGoto(c, "WaitOnChan")
            ∧ locked′ = FALSE
            ∧ setOutputChan(c, k)
            ∧ consume_channel′ = [consume_channel EXCEPT ![c] = client_channel[c]]
            ∧ UNCHANGED client_queue
            ∧ client_states′ = [client_states EXCEPT ![c][k] = server_state[k]]
            ∧ UNCHANGED wait_list
            $\land$ UNCHANGED $push\_back\_list$
    $\land$ UNCHANGED $\langle server\_pc, server\_state \rangle$
    $\land$ UNCHANGED $client\_keys$
    $\land$ UNCHANGED $next\_val$
    $\land$ UNCHANGED $outer\_states$

$ConsumeFromChan(c) \triangleq$
   LET
        $index \triangleq consume\_channel[c]$
        $old\_state \triangleq channels[index]$
        $k \triangleq old\_state.data[1]$
        $val \triangleq old\_state.data[2]$
        $new\_state \triangleq [old\_state$ EXCEPT $!.data = nil, !.status =$ “Consumed”$]$
   IN
        $\land client\_pc[c] =$ “WaitOnChan”
        $\land channels[index].status =$ “Ready”
        $\land clientGoto(c,$ “Init”$)$
        $\land channels' = [channels$ EXCEPT $![index] = new\_state]$
        $\land outer\_states' = [outer\_states$ EXCEPT $![c][k] = val]$
        $\land$ UNCHANGED $\langle client\_keys, client\_states, client\_queue, client\_channel \rangle$
        $\land$ UNCHANGED $consume\_channel$
        $\land$ UNCHANGED $locked$
        $\land$ UNCHANGED $\langle server\_pc, server\_state, next\_val, wait\_list \rangle$
        $\land$ UNCHANGED $push\_back\_list$

$TerminateCond \triangleq$
    $\land server\_pc =$ “Init”
    $\land \forall c \in Client :$
        $\land client\_pc[c] =$ “WaitOnChan”
        $\land channels[consume\_channel[c]].status =$ “Empty”
    $\land next\_val = max\_val$

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

$Next \triangleq$
    $\lor \exists k \in Key :$
        $\lor SetServerState(k)$
    $\lor \exists c \in Client :$
        $\lor GetState(c)$
        $\lor ClientCheckQueue(c)$
        $\lor \exists k \in Key :$
            $\lor GetFromQueue(c, k)$
            $\lor\ ServerCheckWaitList(k,\ c)$
          $\lor\ ConsumeFromChan(c)$
      $\lor\ ServerSetBackWaitList$
      $\lor\ Terminated$

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

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

$Inv \triangleq$
    $TerminateCond \Rightarrow$
        $\forall c \in Client : \forall k \in client\_keys[c] :$
            $\land\ client\_states[c][k] = server\_state[k]$
            $\land\ outer\_states[c][k] = server\_state[k]$

$AlwaysTerminate \triangleq \Diamond TerminateCond$

$ChannelInv \triangleq$
    $\forall index \in 1 \,..\, Len(channels) :$
       LET
           $ch \triangleq channels[index]$
       IN
           $\lor\ ch.data = nil \land ch.status =$ "Empty"
           $\lor\ ch.data = nil \land ch.status =$ "Consumed"
           $\lor\ ch.data \neq nil \land ch.status =$ "Ready"

$LockedCorrectly \triangleq$
    $(server\_pc =$ "Init" $\land \forall c \in Client : client\_pc[c] =$ "Init"$) \Rightarrow \neg locked$

$allChannelConsumedExceptWaiting \triangleq$
    $\forall i \in$ DOMAIN $channels :$
       $(\forall c \in Client : consume\_channel[c] \neq i) \Rightarrow channels[i].status =$ "Consumed"

$AllChannelConsumed \triangleq$
    $TerminateCond \Rightarrow allChannelConsumedExceptWaiting$

$channelPushOrRecv \triangleq$
    $\forall index \in 1 \,..\, Len(channels) :$
       LET
           $before \triangleq channels[index]$
           $after \triangleq channels'[index]$
       IN
           $\lor\ \land\ before.status =$ "Empty"
              $\land\ after.status =$ "Ready"
          $\lor \land \mathit{before.status} = \text{“Ready”}$
            $\land \mathit{after.status} = \text{“Consumed”}$
          $\lor \mathit{before} = \mathit{after}$

$\mathit{channelPushRecvOrAppend} \triangleq$
    $\lor \mathit{channelPushOrRecv}$
    $\lor \mathit{Len}(\mathit{channels}') = \mathit{Len}(\mathit{channels}) + 1$

$\mathit{ChannelPushInv} \triangleq$
    $\Box[\mathit{channelPushRecvOrAppend}]_{\mathit{channels}}$

$\mathit{Symm} \triangleq \mathit{Permutations}(\mathit{Key}) \cup \mathit{Permutations}(\mathit{Client})$
```