MODULE $LamportMutex\_1$

TLA+ specification of *Lamport*'s distributed mutual-exclusion algorithm that appeared as an example in L. *Lamport*: Time, Clocks and the Ordering of Events in a Distributed System. *CACM* 21(7):558 – 565, 1978.

EXTENDS $Naturals$, $Sequences$

The parameter N represents the number of processes. The parameter $maxClock$ is used only for model checking in order to keep the state space finite.

CONSTANT N, $maxClock$

ASSUME $NType \triangleq N \in Nat$
ASSUME $maxClockType \triangleq maxClock \in Nat$

$Proc \triangleq 1 \,..\, N$
$Clock \triangleq Nat \setminus \{0\}$

For model checking, add $ClockConstraint$ as a state constraint to ensure a finite state space and override the definition of $Clock$ by $1 \,..\, maxClock + 1$ so that TLC can evaluate the definition of $Message$.

VARIABLES
- $clock$, — local clock of each process
- req, — requests received from processes (clock transmitted with request)
- ack, — acknowledgements received from processes
- $network$, — messages sent but not yet received
- $crit$ — set of processes in critical section

Messages used in the algorithm.

$ReqMessage(c) \triangleq [type \mapsto \text{"req"},\ clock \mapsto c]$
$AckMessage \triangleq [type \mapsto \text{"ack"},\ clock \mapsto 0]$
$RelMessage \triangleq [type \mapsto \text{"rel"},\ clock \mapsto 0]$

$Message \triangleq \{AckMessage,\ RelMessage\} \cup \{ReqMessage(c) : c \in Clock\}$

The type correctness predicate.

$TypeOK \triangleq$

$clock[p]$ is the local clock of process p

$\land\ clock \in [Proc \rightarrow Clock]$

$req[p][q]$ stores the clock associated with request from q received by p, 0 if none

$\land\ req \in [Proc \rightarrow [Proc \rightarrow Nat]]$

$ack[p]$ stores the processes that have ack'ed p's request

$\land\ ack \in [Proc \rightarrow \text{SUBSET } Proc]$

$network[p][q]$: queue of messages from p to q – pairwise $FIFO$ communication

$\land\ network \in [Proc \rightarrow [Proc \rightarrow Seq(Message)]]$

set of processes in critical section: should be empty or singleton

$\land\ crit \in \text{SUBSET } Proc$

The initial state predicate.

$$Init \triangleq$$
$$\land\ clock = [p \in Proc \mapsto 1]$$
$$\land\ req = [p \in Proc \mapsto [q \in Proc \mapsto 0]]$$
$$\land\ ack = [p \in Proc \mapsto \{\}]$$
$$\land\ network = [p \in Proc \mapsto [q \in Proc \mapsto \langle\rangle]]$$
$$\land\ crit = \{\}$$

$beats(p, q)$ is true if process p believes that its request has higher priority than q's request. This is true if either p has not received a request from q or p's request has a smaller clock value than q's. If there is a tie, the numerical process ID decides.

$$beats(p, q) \triangleq$$
$$\lor\ req[p][q] = 0$$
$$\lor\ req[p][p] < req[p][q]$$
$$\lor\ req[p][p] = req[p][q] \land p < q$$

Broadcast a message: send it to all processes except the sender.

$$Broadcast(s, m) \triangleq$$
$$[r \in Proc \mapsto \text{IF } s = r \text{ THEN } network[s][r] \text{ ELSE } Append(network[s][r], m)]$$

Process p requests access to critical section.

$$Request(p) \triangleq$$
$$\land\ req[p][p] = 0$$
$$\land\ req' = [req \text{ EXCEPT } ![p][p] = clock[p]]$$
$$\land\ network' = [network \text{ EXCEPT } ![p] = Broadcast(p, ReqMessage(clock[p]))]$$
$$\land\ ack' = [ack \text{ EXCEPT } ![p] = \{p\}]$$
$$\land\ \text{UNCHANGED } \langle clock, crit\rangle$$

Process p receives a request from q and acknowledges it.

$$ReceiveRequest(p, q) \triangleq$$
$$\land\ network[q][p] \neq \langle\rangle$$
$$\land\ \text{LET } m \triangleq Head(network[q][p])$$
$$\quad\quad c \triangleq m.clock$$
$$\text{IN}\ \land\ m.type = \text{“req”}$$
$$\quad\land\ req' = [req \text{ EXCEPT } ![p][q] = c]$$
$$\quad\land\ clock' = [clock \text{ EXCEPT } ![p] = \text{IF } c > clock[p] \text{ THEN } c + 1 \text{ ELSE } @ + 1]$$
$$\quad\land\ network' = [network \text{ EXCEPT } ![q][p] = Tail(@),$$
$$\quad\quad ![p][q] = Append(@, AckMessage)]$$
$$\quad\land\ \text{UNCHANGED } \langle ack, crit\rangle$$

Process p receives an acknowledgement from q.

$$ReceiveAck(p, q) \triangleq$$
$$\land\ network[q][p] \neq \langle\rangle$$
$$\land\ \text{LET } m \triangleq Head(network[q][p])$$
$$\text{IN}\ \land\ m.type = \text{“ack”}$$

$\land\ ack' = [ack \text{ EXCEPT } ![p] = @ \cup \{q\}]$
$\land\ network' = [network \text{ EXCEPT } ![q][p] = Tail(@)]$
$\land$ UNCHANGED $\langle clock,\ req,\ crit\rangle$

Process p enters the critical section.

$Enter(p) \triangleq$
$\land\ ack[p] = Proc$
$\land\ \forall q \in Proc \setminus \{p\} : beats(p,\ q)$
$\land\ crit' = crit \cup \{p\}$
$\land$ UNCHANGED $\langle clock,\ req,\ ack,\ network\rangle$

Process p exits the critical section and notifies other processes.

$Exit(p) \triangleq$
$\land\ p \in crit$
$\land\ crit' = crit \setminus \{p\}$
$\land\ network' = [network \text{ EXCEPT } ![p] = Broadcast(p,\ RelMessage)]$
$\land\ req' = [req \text{ EXCEPT } ![p][p] = 0]$
$\land\ ack' = [ack \text{ EXCEPT } ![p] = \{\}]$
$\land$ UNCHANGED $clock$

Process p receives a release notification from q.

$ReceiveRelease(p,\ q) \triangleq$
$\land\ network[q][p] \neq \langle\rangle$
$\land$ LET $m \triangleq Head(network[q][p])$
IN $\land\ m.type =$ "rel"
$\land\ req' = [req \text{ EXCEPT } ![p][q] = 0]$
$\land\ network' = [network \text{ EXCEPT } ![q][p] = Tail(@)]$
$\land$ UNCHANGED $\langle clock,\ ack,\ crit\rangle$

Next-state relation.

$Next \triangleq$
$\lor\ \exists p \in Proc : Request(p) \lor Enter(p) \lor Exit(p)$
$\lor\ \exists p \in Proc : \exists q \in Proc \setminus \{p\} :$
$ReceiveRequest(p,\ q) \lor ReceiveAck(p,\ q) \lor ReceiveRelease(p,\ q)$

$vars \triangleq \langle req,\ network,\ clock,\ ack,\ crit\rangle$

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

A state constraint that is useful for validating the specification using finite-state model checking.

$ClockConstraint \triangleq \forall p \in Proc : clock[p] \leq maxClock$

No channel ever contains more than three messages. In fact, no channel ever contains more than one message of the same type, as proved below.

$BoundedNetwork \triangleq \forall p, q \in Proc : Len(network[p][q]) \leq 3$

The main safety property of mutual exclusion.

$Mutex \triangleq \forall p, q \in crit : p = q$

$Live \triangleq \forall p \in Proc : \Diamond(p \in crit)$

* Modification History
* Last modified Sun *Oct* 07 00:34:02 *EDT* 2018 by *mehuljain*
* Created Sun *Sep* 30 20:38:42 *EDT* 2018 by *mehuljain*