MODULE *backpressure*

EXTENDS *FiniteSets*, *Integers*, *Sequences*, *TLC*

$Null \triangleq 0$
$Cowns \triangleq 1 \,..\, 4$
$BehaviourLimit \triangleq 4$
$OverloadThreshold \triangleq 2$
$PriorityLevels \triangleq \{-1, 0, 1\}$

$Min(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s \setminus \{x\} : y > x$
$Max(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s \setminus \{x\} : y < x$

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$

VARIABLES *fuel*, *queue*, *scheduled*, *running*, *priority*, *blocker*, *mutor*, *mute*
$vars \triangleq \langle fuel, queue, scheduled, running, priority, blocker, mutor, mute\rangle$

$EmptyQueue(c) \triangleq Len(queue[c]) = 0$

$Sleeping(c) \triangleq scheduled[c] \land EmptyQueue(c)$

$Available(c) \triangleq scheduled[c] \land \neg EmptyQueue(c)$

$Overloaded(c) \triangleq Len(queue[c]) > OverloadThreshold$

$CurrentMessage(c) \triangleq \text{IF } EmptyQueue(c) \text{ THEN } \{\} \text{ ELSE } Head(queue[c])$

$LowPriority(cs) \triangleq \{c \in cs : priority[c] = -1\}$

$HighPriority(cs) \triangleq \{c \in cs : priority[c] = 1\}$

$RequiresPriority(c) \triangleq$
  $\lor\ Overloaded(c)$
  $\lor \exists\, m \in Range(queue[c]) : \exists\, k \in m \setminus \{c\} : priority[k] = 1$

RECURSIVE $Blockers(\_)$
$Blockers(c) \triangleq$
  IF $blocker[c] = Null$ THEN $\{\}$ ELSE $\{blocker[c]\} \cup Blockers(blocker[c])$
$Prioritizing(cs) \triangleq$
  LET $unprioritized \triangleq \{c \in cs : priority[c] < 1\}$IN
  $unprioritized \cup \text{UNION } \{Blockers(c) : c \in unprioritized\}$

$ValidMutor(c) \triangleq$
  $\lor\ (priority[c] = 1) \land Overloaded(c)$
  $\lor\ (priority[c] = -1)$

$Init \triangleq$
  $\land\ fuel = BehaviourLimit$
  $\land\ queue = [c \in Cowns \mapsto \langle\{c\}\rangle]$

$\land\ scheduled = [c \in Cowns \mapsto \text{TRUE}]$
$\land\ running = [c \in Cowns \mapsto \text{FALSE}]$
$\land\ priority = [c \in Cowns \mapsto 0]$
$\land\ blocker = [c \in Cowns \mapsto Null]$
$\land\ mutor = [c \in Cowns \mapsto Null]$
$\land\ mute = [c \in Cowns \mapsto \{\}]$

$Terminating \triangleq$
$\land\ \forall\, c \in Cowns : EmptyQueue(c)$
$\land\ \text{UNCHANGED}\ vars$

$Acquire(cown) \triangleq$
LET $msg \triangleq CurrentMessage(cown)$IN
$\land\ Available(cown)$
$\land\ cown < Max(msg)$
$\land\ \text{IF}\ priority[cown] = 1\ \text{THEN}$
LET $prioritizing \triangleq Prioritizing(\{Min(\{c \in msg : c > cown\})\})$IN
LET $unmuting \triangleq LowPriority(prioritizing)$IN
$\land\ priority' = [c \in prioritizing \mapsto 1]\ @@\ priority$
$\land\ scheduled' = (cown :> \text{FALSE})\ @@\ [c \in unmuting \mapsto \text{TRUE}]\ @@\ scheduled$
ELSE
$\land\ scheduled' = (cown :> \text{FALSE})\ @@\ scheduled$
$\land\ \text{UNCHANGED}\ \langle priority, mute\rangle$
$\land$ LET $next \triangleq Min(\{c \in msg : c > cown\})$IN
$\land\ blocker' = (cown :> next)\ @@\ blocker$
$\land$ LET $q \triangleq (cown :> Tail(queue[cown]))\ @@\ queue$IN
$queue' = (next :> Append(queue[next], msg))\ @@\ q$
$\land\ \text{UNCHANGED}\ \langle fuel, running, mutor, mute\rangle$

$Prerun(cown) \triangleq$
LET $msg \triangleq CurrentMessage(cown)$IN
$\land\ scheduled[cown]$
$\land\ \neg running[cown]$
$\land\ \text{IF}\ msg = \{\}\ \text{THEN FALSE ELSE}\ cown = Max(msg)$
$\land\ priority' = (cown :> \text{IF}\ RequiresPriority(cown)\ \text{THEN}\ 1\ \text{ELSE}\ 0)\ @@\ priority$
$\land\ running' = (cown :> \text{TRUE})\ @@\ running$
$\land\ blocker' = [c \in msg \mapsto Null]\ @@\ blocker$
$\land\ \text{UNCHANGED}\ \langle fuel, queue, scheduled, mutor, mute\rangle$

$Send(cown) \triangleq$
LET $senders \triangleq CurrentMessage(cown)$IN
$\land\ running[cown]$
$\land\ fuel > 0$
$\land\ \exists\, receivers \in \text{SUBSET}\ Cowns :$
$\land\ Cardinality(receivers) > 0$
$\land\ queue' =$

$(Min(receivers) :> Append(queue[Min(receivers)], receivers))$ @@ $queue$
$\land$ IF $priority[Min(receivers)] = 1$ THEN
LET $prioritizing \triangleq Prioritizing(\{Min(receivers)\})$IN
LET $unmuting \triangleq LowPriority(prioritizing)$IN
$\land priority' = [c \in prioritizing \mapsto 1]$ @@ $priority$
$\land scheduled' = [c \in unmuting \mapsto$ TRUE$]$ @@ $scheduled$
$\land$ LET $mutors \triangleq \{c \in receivers \setminus senders : ValidMutor(c)\}$IN
IF
$\land mutors \neq \{\}$
$\land mutor[cown] = Null$
$\land \forall\, c \in senders : priority[c] = 0$
$\land \forall\, c \in senders : c \notin receivers$ *TODO*: justify
THEN
$\land mutor' = (cown :> Min(mutors))$ @@ $mutor$
ELSE
$\land$ UNCHANGED $\langle mutor \rangle$
ELSE
$\land$ UNCHANGED $\langle scheduled, priority, mutor \rangle$
$\land fuel' = fuel - 1$
$\land$ UNCHANGED $\langle running, blocker, mute \rangle$

$Complete(cown) \triangleq$
LET $msg \triangleq CurrentMessage(cown)$IN
$\land running[cown]$
$\land$ IF $mutor[cown] \neq Null$ THEN
LET $muting \triangleq \{c \in msg : priority[c] = 0\}$IN
$\land priority' = [c \in muting \mapsto -1]$ @@ $priority$
$\land mute' = (mutor[cown] :> mute[mutor[cown]] \cup muting)$ @@ $mute$
$\land scheduled' = [c \in msg \mapsto c \notin muting]$ @@ $scheduled$
ELSE
$\land scheduled' = [c \in msg \mapsto$ TRUE$]$ @@ $scheduled$
$\land priority' =$
$(cown :>$ IF $Len(queue[cown]) = 1$ THEN 0 ELSE $priority[cown])$ @@
$[c \in msg \setminus \{cown\} \mapsto$ IF $EmptyQueue(c)$ THEN 0 ELSE $priority[c]]$ @@
$priority$
$\land$ UNCHANGED $\langle mute \rangle$
$\land queue' = (cown :> Tail(queue[cown]))$ @@ $queue$
$\land running' = (cown :>$ FALSE$)$ @@ $running$
$\land mutor' = (cown :> Null)$ @@ $mutor$
$\land$ UNCHANGED $\langle fuel, blocker \rangle$

$Unmute \triangleq$
LET $invalid\_keys \triangleq \{c \in$ DOMAIN $mute : priority[c] = 0\}$IN
LET $unmuting \triangleq$ UNION $Range([k \in invalid_keys \mapsto LowPriority(mute[k])])$IN
$\land unmuting \neq \{\}$

$\land\ priority' = [c \in unmuting \mapsto 0] \mathbin{@@} priority$
$\land\ mute' = [c \in invalid\_keys \mapsto \{\}] \mathbin{@@} mute$
$\land\ scheduled' = [c \in unmuting \mapsto \text{TRUE}] \mathbin{@@} scheduled$
$\land\ \text{UNCHANGED}\ \langle fuel,\ queue,\ running,\ blocker,\ mutor \rangle$

$Run(cown) \triangleq$
$\lor\ Acquire(cown)$
$\lor\ Prerun(cown)$
$\lor\ Send(cown)$
$\lor\ Complete(cown)$

$Next \triangleq \exists\, c \in Cowns : Run(c) \lor Unmute$

$Spec \triangleq$
$\land\ Init$
$\land\ \Box[Next \lor Terminating]_{vars}$
$\land\ \forall\, c \in Cowns : \mathrm{WF}_{vars}(Run(c))$
$\land\ \mathrm{WF}_{vars}(Unmute)$

Utility Functions

$Pick(s) \triangleq \text{CHOOSE}\ x \in s : \text{TRUE}$

$ReduceSet(op(\_, \_),\ set,\ acc) \triangleq$
LET $f[s \in \text{SUBSET}\ set] \triangleq$
IF $s = \{\}$ THEN acc ELSE LET $x \triangleq Pick(s)$IN $op(x, f[s \setminus \{x\}])$
IN $f[set]$

$MutedBy(a,\ b) \triangleq (a \in mute[b]) \land (priority[a] = -1)$
$Muted(c) \triangleq \exists\, k \in Cowns : MutedBy(c,\ k)$

$AcquiredBy(a,\ b) \triangleq (a < b) \land (a \in \text{UNION}\ Range(queue[b]))$
$Acquired(c) \triangleq \exists\, k \in Cowns : AcquiredBy(c,\ k)$

$Required(c) \triangleq \exists\, k \in Cowns : (k < c) \land (c \in \text{UNION}\ Range(queue[k]))$

https://*github.com*/tlaplus/Examples/blob/master/specifications/*TransitiveClosure*/*TransitiveClosure.tla*#*L*114
$TC(R) \triangleq$
LET
$S \triangleq \{r[1] : r \in R\} \cup \{r[2] : r \in R\}$
RECURSIVE $TCR(\_)$
$TCR(T) \triangleq$
IF $T = \{\}$ THEN R
ELSE
LET
$r \triangleq \text{CHOOSE}\ s \in T : \text{TRUE}$
$RR \triangleq TCR(T \setminus \{r\})$
IN

$$RR \cup \{\langle s,\, t\rangle \in S \times S : \langle s,\, r\rangle \in RR \land \langle r,\, t\rangle \in RR\}$$

IN

$TCR(S)$

$CylcicTransitiveClosure(R(\_,\, \_)) \triangleq$
LET $s \triangleq \{\langle a,\, b\rangle \in Cowns \times Cowns : R(a,\, b)\}$
IN $\exists\, c \in Cowns : \langle c,\, c\rangle \in TC(s)$

Temporal Properties

The model does not livelock.
$Termination \triangleq \Diamond\Box(\forall\, c \in Cowns : Sleeping(c))$

Invariants

The message limit for TLC is enforced (the model has finite state space).
$MessageLimit \triangleq$
LET $msgs \triangleq ReduceSet(\text{LAMBDA}\ c,\, sum : sum + Len(queue[c]),\, Cowns,\, 0)$IN
$msgs \leq (BehaviourLimit + Max(Cowns))$

The running *cown* is scheduled and the greatest *cown* in the head of its queue.
$RunningIsScheduled \triangleq$
$\forall\, c \in Cowns : running[c] \Rightarrow scheduled[c] \land (c = Max(CurrentMessage(c)))$

A *cown* is not its own *mutor*.
$CownNotMutedBySelf \triangleq \forall\, c \in Cowns : c \notin mute[c]$

A low-priority *cown* is muted.
$LowPriorityMuted \triangleq \forall\, c \in Cowns : (priority[c] = -1) \Rightarrow Muted(c)$

There cannot be message that has acquired a high-priority *cown* and has acquired, or is in the queue of, a low-priority *cown*.
$Nonblocking \triangleq$
$\forall\, c \in Cowns : \forall\, m \in Range(queue[c]) :$
$\forall\, \langle l,\, h\rangle \in LowPriority(m) \times HighPriority(m) : (c \leq h) \lor (c < l)$

All cowns in a running message have no blocker.
$RunningNotBlocked \triangleq$
$\forall\, c \in Cowns : running[c] \Rightarrow (\forall\, k \in CurrentMessage(c) : blocker[k] = Null)$

An unscheduled *cown* is either muted or acquired.
$UnscheduledByMuteOrAcquire \triangleq$
$\forall\, c \in Cowns : \neg((priority[c] = -1) \lor Acquired(c)) \equiv scheduled[c]$

A *cown* in the queue of a greater *cown* is unscheduled.
$BehaviourAcquisition \triangleq$
$\forall\, c \in Cowns : \forall\, k \in \text{UNION}\ Range(queue[c]) : (k < c) \Rightarrow \neg scheduled[k]$

A *cown* can only be acquired by at most one *cown*.

$$AcquiredOnce \triangleq$$
$$\forall \langle a,\ b,\ c \rangle \in Cowns \times Cowns \times Cowns :$$
$$(AcquiredBy(a,\ b) \land AcquiredBy(a,\ c)) \Rightarrow (b = c)$$

All messages in a *cown*'s queue must contain the *cown*.

$$SelfInQueueMessages \triangleq \forall\, c \in Cowns : \forall\, m \in Range(queue[c]) : c \in m$$

A high-priority *cown* is in a queue of a high-priority *cown*.

$$HighPriorityInUnblockedQueue \triangleq$$
$$\forall\, c \in HighPriority(Cowns) :$$
$$\exists\, k \in HighPriority(Cowns) : c \in \text{UNION}\ Range(queue[k])$$

Warning: not enforced by implementation.

$$SleepingIsNormal \triangleq \forall\, c \in Cowns : Sleeping(c) \Rightarrow (priority[c] = 0)$$

High-priority cowns has messages in its queue or is acquired.

$$HighPriorityHasWork \triangleq \forall\, c \in HighPriority(Cowns) :$$
$$\lor \neg EmptyQueue(c)$$
$$\lor Acquired(c)$$

A muted *cown* has only one *mutor* in the mute map.

$$MuteSetsDisjoint \triangleq \forall \langle a,\ b \rangle \in Cowns \times Cowns :$$
$$((mute[a] \cap mute[b]) \neq \{\}) \Rightarrow (a = b)$$

The transitive closure of the relation *MutedBy* has no cycles.

$$AcyclicTCMute \triangleq \neg CylcicTransitiveClosure(MutedBy)$$