MODULE *backpressure*

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

$Null \triangleq 0$
$Cowns \triangleq 1 \,..\, 2$ # *TODO*: 4
$MaxMessageCount \triangleq 1$ # *TODO*: 4
$MaxMessageSize \triangleq 3$
$OverloadThreshold \triangleq 2$
$PriorityLevels \triangleq \{-1, 0, 1\}$

$Pick(s) \triangleq$ CHOOSE $x \in s :$ TRUE
$Min(s) \triangleq$ CHOOSE $x \in s : \forall\, y \in s \setminus \{x\} : y > x$
$Max(s) \triangleq$ CHOOSE $x \in s : \forall\, y \in s \setminus \{x\} : y < x$
$Range(f) \triangleq \{f[x] : x \in$ DOMAIN $f\}$
$Subsets(s, min, max) \triangleq$
  $\{x \in$ SUBSET $s : (Cardinality(x) \geq min) \land (Cardinality(x) \leq max)\}$
RECURSIVE $Concat(\_)$
$Concat(s) \triangleq$ IF $s = \{\}$ THEN $\langle\rangle$ ELSE LET $x \triangleq Pick(s)$IN $\quad x \circ Concat(s \setminus \{x\})$

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

$Messages \triangleq$ UNION $\{Range(queue[c]) : c \in Cowns\}$
$EmptyQueue(c) \triangleq Len(queue[c]) = 0$
$Overloaded(c) \triangleq Len(queue[c]) \geq OverloadThreshold$
$Enqueue(c, m) \triangleq c :> Append(queue[c], m)$
$Dequeue(c) \triangleq c :> Tail(queue[c])$

RECURSIVE $Blockers(\_)$
$Blockers(c) \triangleq$
  IF $blocker[c] = Null$ THEN $\{\}$
  ELSE $\{blocker[c]\} \cup Blockers(blocker[c])$

$Running(c) \triangleq \exists\, k \in Cowns : running[k] \land c \in Head(queue[k])$

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

$Normal(c) \triangleq priority[c] = 0$
$Prioritized(c) \triangleq priority[c] = 1$
$Muted(c) \triangleq priority[c] = -1$

$MutedBy(a, b) \triangleq$
  $\land\, Muted(a)$

$\land \exists\, m \in \mathit{Range}(\mathit{queue}[b]) : (b \notin m) \land (a \in m)$

$\mathit{Init} \triangleq$
$\land\ \mathit{fuel} = \mathit{MaxMessageCount}$
$\land\ \mathit{queue} = [c \in \mathit{Cowns} \mapsto \langle\{c\}\rangle]$
$\land\ \mathit{scheduled} = [c \in \mathit{Cowns} \mapsto \text{TRUE}]$
$\land\ \mathit{running} = [c \in \mathit{Cowns} \mapsto \text{FALSE}]$
$\land\ \mathit{mutor} = [c \in \mathit{Cowns} \mapsto \mathit{Null}]$
$\land\ \mathit{priority} = [c \in \mathit{Cowns} \mapsto 0]$
$\land\ \mathit{blocker} = [c \in \mathit{Cowns} \mapsto \mathit{Null}]$

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

$\mathit{ExternalReceive}(\mathit{cown}) \triangleq$
$\land\ \mathit{fuel} > 0$
———
$\land$ UNCHANGED $\langle \mathit{scheduled}, \mathit{running}, \mathit{mutor}, \mathit{priority}, \mathit{blocker}\rangle$
$\land\ \mathit{fuel}' = \mathit{fuel} - 1$
\# Receive a message from an external source
$\land \exists\, \mathit{others} \in \mathit{Subsets}(\{c \in \mathit{Cowns} : c > \mathit{cown}\},\ 0,\ \mathit{MaxMessageSize} - 1) :$
$\quad \mathit{queue}' = \mathit{Enqueue}(\mathit{cown},\ \{\mathit{cown}\} \cup \mathit{others})$ @@ $\mathit{queue}$

$\mathit{Acquire}(\mathit{cown}) \triangleq$
$\land\ \mathit{scheduled}[\mathit{cown}]$
$\land\ \neg \mathit{running}[\mathit{cown}]$
$\land\ \neg \mathit{EmptyQueue}(\mathit{cown})$
$\land\ \mathit{cown} \in \mathit{Head}(\mathit{queue}[\mathit{cown}])$
$\land\ \mathit{cown} < \mathit{Max}(\mathit{Head}(\mathit{queue}[\mathit{cown}]))$
———
$\land$ UNCHANGED $\langle \mathit{fuel}, \mathit{running}, \mathit{mutor}\rangle$
\# *Unschedule* and forward the message to the next *cown*.
$\land$ LET
$\quad \mathit{msg} \triangleq \mathit{Head}(\mathit{queue}[\mathit{cown}])$
$\quad \mathit{next} \triangleq \mathit{Min}(\{c \in \mathit{msg} : c > \mathit{cown}\})$
IN
$\quad \land\ \mathit{queue}' = \mathit{Enqueue}(\mathit{next},\ \mathit{msg})$ @@ $\mathit{Dequeue}(\mathit{cown})$ @@ $\mathit{queue}$
$\quad \land\ \mathit{blocker}' = (\mathit{cown} :> \mathit{next})$ @@ $\mathit{blocker}$
\# *Prioritize* this *cown* and next if either are prioritized. *Unmute* next.
$\quad \land$ IF $\exists\, c \in \{\mathit{cown},\ \mathit{next}\} : \mathit{Prioritized}(c)$ THEN
$\quad\quad \land\ \mathit{priority}' = (\mathit{next} :> 1)$ @@ $\mathit{priority}$
$\quad\quad \land\ \mathit{scheduled}' = (\mathit{next} :> \text{TRUE})$ @@ $(\mathit{cown} :> \text{FALSE})$ @@ $\mathit{scheduled}$
ELSE
$\quad\quad \land$ UNCHANGED $\langle \mathit{priority}\rangle$

$\land\ scheduled' = (cown :> \text{FALSE})\ @@\ scheduled$

$Unmute(cown)\ \triangleq$
$\land\ scheduled[cown]$
$\land\ \neg running[cown]$
$\land\ \neg EmptyQueue(cown)$
$\land\ cown \notin Head(queue[cown])$
———
$\land$ UNCHANGED $\langle fuel,\ running,\ mutor,\ blocker\rangle$
# Remove message from queue.
$\land\ queue' = Dequeue(cown)\ @@\ queue$
# Reschedule muted cowns.
$\land$ LET $muted\ \triangleq\ \{c \in Head(queue[cown]) : Muted(c)\}$ IN
$\land\ priority' = [c \in muted \mapsto 0]\ @@\ priority$
$\land\ scheduled' = [c \in muted \mapsto \text{TRUE}]\ @@\ scheduled$

$PreRun(cown)\ \triangleq$
$\land\ scheduled[cown]$
$\land\ \neg running[cown]$
$\land\ \neg EmptyQueue(cown)$
$\land\ cown = Max(Head(queue[cown]))$
———
$\land$ UNCHANGED $\langle fuel,\ queue,\ scheduled,\ mutor,\ priority\rangle$
# Set *max cown* in current message to running
$\land\ running' = (cown :> \text{TRUE})\ @@\ running$
$\land\ blocker'\ = [c \in Head(queue[cown]) \mapsto Null]\ @@\ blocker$

$Send(cown)\ \triangleq$
$\land\ running[cown]$
$\land\ fuel > 0$
# ———
$\land$ UNCHANGED $\langle running,\ blocker\rangle$
$\land\ fuel' = fuel - 1$
$\land\ \exists\ receivers \in Subsets(Cowns,\ 1,\ MaxMessageSize) :$
LET
$next\ \triangleq\ Min(receivers)$
$senders\ \triangleq\ Head(queue[cown])$
$mutors\ \triangleq\ \{c \in receivers : Overloaded(c)\}$
IN
# Place message for receivers in the first receiver's queue.
$\land\ queue' = Enqueue(next,\ receivers)\ @@\ queue$
$\land$ IF $(\exists\ c \in receivers : Prioritized(c)) \lor Overloaded(next)$ THEN
# *Prioritize* next.
$\land\ priority' = (next :> 1)\ @@\ priority$
# *Reschdeule* next if it was muted.
$\land\ scheduled' = (next :> (Muted(next) \lor scheduled[next]))\ @@\ scheduled$

```
        # Set mutor if any receiver is overloaded and there are no receivers in the set of senders.
        ∧ IF
            ∧ mutors ≠ {}
            ∧ mutor[cown] = Null
            ∧ (senders ∩ receivers) = {}
            THEN mutor' = (cown :> Min(mutors)) @@ mutor
            ELSE UNCHANGED ⟨mutor⟩
        ELSE
        ∧ UNCHANGED ⟨scheduled, mutor, priority⟩

PostRun(cown) ≜
  ∧ running[cown]
  ———
  ∧ UNCHANGED ⟨fuel, blocker⟩
  ∧ running' = (cown :> FALSE) @@ running
  ∧ mutor' = (cown :> Null) @@ mutor
  ∧ LET msg ≜ Head(queue[cown]) IN
      # Mute if mutor is set and no running cowns are overloaded.
    ∧ IF (mutor[cown] ≠ Null) ∧ (∀ c ∈ msg : ¬Overloaded(c)) THEN
        ∧ priority' = [c ∈ msg ↦ −1] @@ priority
        ∧ scheduled' = [c ∈ msg ↦ FALSE] @@ scheduled
        # Send unmute message to mutor
        ∧ queue' = Enqueue(mutor[cown], msg) @@ Dequeue(cown) @@ queue
        ELSE
        ∧ UNCHANGED ⟨priority⟩
        ∧ scheduled' = [c ∈ msg ↦ TRUE] @@ scheduled
        ∧ queue' = Dequeue(cown) @@ queue

RunStep(cown) ≜
    ∨ ExternalReceive(cown) \*# Very expensive check
  ∨ Acquire(cown)
  ∨ Unmute(cown)
  ∨ PreRun(cown)
  ∨ Send(cown)
  ∨ PostRun(cown)

Next ≜ ∃ c ∈ Cowns : RunStep(c)

Spec ≜
  ∧ Init
  ∧ □[Next ∨ Terminating]_vars
  ∧ ∀ c ∈ Cowns : WF_vars(RunStep(c))

 # Properties

 # Ensure that the termination condition is reached by the model.
Termination ≜ ◇□(∀ c ∈ Cowns : EmptyQueue(c))
```

# Invariants

# Ensure that the model produces finite messages.
$MessageLimit \triangleq Cardinality(Messages) \leq (Cardinality(Cowns) + MaxMessageCount)$

# *Cowns* are acquired by one running message at a time.
$UniqueAcquisition \triangleq$
LET $msgs \triangleq Concat(\{\langle Head(queue[c])\rangle : c \in \{k \in Cowns : running[k]\}\})$
IN $Cardinality(Range(msgs)) = Len(msgs)$

# Each queue has at most one token message.
$LoneToken \triangleq \forall\, c \in Cowns : Len(SelectSeq(queue[c], \text{LAMBDA } m : m = \{\})) \leq 1$

# A running *cown* must be scheduled and be the *max cown* in the message at the head of its queue.
$RunningImplication \triangleq \forall\, c \in Cowns : running[c] \Rightarrow$
$\quad \land\, scheduled[c]$
$\quad \land\, c = Max(Head(queue[c]))$
$\quad \land \forall\, k \in Head(queue[c]) : (k < c) \Rightarrow AcquiredBy(k,\, c)$

# An acquired *cown* is not scheduled.
$AcquiredImplication \triangleq \forall\, c \in Cowns : Acquired(c) \Rightarrow$
$\quad \land \neg scheduled[c]$

# A muted *cown* is not scheduled or running.
$MutedImplication \triangleq \forall\, c \in Cowns : Muted(c) \equiv$
$\quad \land \exists\, k \in Cowns : MutedBy(c,\, k)$
$\quad \land \neg scheduled[c]$
$\quad \land \neg Running(c)$

# A muted *cown* exists in an unmute message in the queue of at least one *mutor*.
$MutedInUnmuteMsg \triangleq$
$\quad \forall\, m \in \{c \in Cowns : Muted(c)\} :$
$\quad\quad Cardinality(\{c \in Cowns : MutedBy(m,\, c)\}) > 0$

# A *cown* may be acqiured by at most one message.
$AcquiredOnce \triangleq$
$\quad \forall\, a \in \{c \in Cowns : Acquired(c)\} :$
$\quad\quad Cardinality(\{c \in Cowns : AcquiredBy(a,\, c)\}) = 1$

# An acquired *cown* is acquired by a *cown* in its blocker set.
$AcquiredByBlocker \triangleq \forall\, \langle a,\, b\rangle \in Cowns \times Cowns :$
$\quad AcquiredBy(a,\, b) \Rightarrow b \in Blockers(a)$

# An overloaded *cown* doesn't exist in a muted *cown*'s queue.
$OverloadedNotInMutedQueue \triangleq \forall\, \langle o,\, m\rangle \in Cowns \times Cowns :$
$\quad Overloaded(o) \land Muted(m) \Rightarrow o \notin \text{UNION } Range(queue[m])$

```
(*
\* https://github.com/tlaplus/Examples/blob/master/specifications/TransitiveClosure/TransitiveClosure.tla#L114

TC(R) ≜
 LET
   S ≜ {r[1] : r ∈ R} ∪ {r[2] : r ∈ R}
   RECURSIVE  TCR(_)
   TCR(T) ≜
    IF  T = {} THEN R
     ELSE
      LET
        r ≜ CHOOSE s ∈ T: TRUE
        RR ≜  TCR(T \ {r})
      IN
        RR ∪ {⟨s, t⟩ ∈ S × S : ⟨s, r⟩ ∈ RR ∧ ⟨r, t⟩ ∈ RR}
 IN
   TCR(S)

CylcicTransitiveClosure(R(_, _)) ≜
 LET s ≜ {⟨a, b⟩ ∈ Cowns × Cowns : R(a, b)}
 IN  ∃ c ∈ Cowns: ⟨c, c⟩ ∈ TC(s)
*)
```