```
MODULE backpressure

EXTENDS FiniteSets, Integers, Sequences, TLC

Null ≜ 0
Cowns ≜ 1 . . 4
BehaviourLimit ≜ 4
OverloadThreshold ≜ 2
PriorityLevels ≜ { − 1, 0, 1}

Min(s) ≜ CHOOSE x ∈ s : ∀ y ∈ s \ {x} : y > x
Max(s) ≜ CHOOSE x ∈ s : ∀ y ∈ s \ {x} : y < x

Range(f) ≜ {f[x] : x ∈ DOMAIN f}

VARIABLES fuel, queue, scheduled, running, priority, blocker, mutor, mute
vars ≜ ⟨fuel, queue, scheduled, running, priority, blocker, mutor, mute⟩

Sleeping(c) ≜ scheduled[c] ∧ (Len(queue[c]) = 0)

Available(c) ≜ scheduled[c] ∧ (Len(queue[c]) > 0)

Overloaded(c) ≜ Len(queue[c]) > OverloadThreshold

CurrentMessage(c) ≜ IF Len(queue[c]) > 0 THEN Head(queue[c]) ELSE {}

LowPriority(cs) ≜ {c ∈ cs : priority[c] = − 1}

HighPriority(cs) ≜ {c ∈ cs : priority[c] = 1}

RequiresPriority(c) ≜
  ∨ Overloaded(c)
  ∨ ∃ m ∈ Range(queue[c]) : ∃ k ∈ m \ {c} : priority[k] = 1

RECURSIVE Blockers(_)
Blockers(c) ≜
  IF blocker[c] = Null THEN {} ELSE {blocker[c]} ∪ Blockers(blocker[c])
Prioritizing(cs) ≜
  LET unprioritized ≜ {c ∈ cs : priority[c] < 1}IN
  unprioritized ∪ UNION {Blockers(c) : c ∈ unprioritized}

ValidMutor(c) ≜
  ∨ (priority[c] = 1) ∧ Overloaded(c)
  ∨ (priority[c] = − 1)

Init ≜
  ∧ fuel = BehaviourLimit
  ∧ queue = [c ∈ Cowns ↦ ⟨{c}⟩]
  ∧ scheduled = [c ∈ Cowns ↦ TRUE]
  ∧ running = [c ∈ Cowns ↦ FALSE]
  $\land\ priority = [c \in Cowns \mapsto 0]$
  $\land\ blocker = [c \in Cowns \mapsto Null]$
  $\land\ mutor = [c \in Cowns \mapsto Null]$
  $\land\ mute = [c \in Cowns \mapsto \{\}]$

$Terminating\ \triangleq$
    $TODO$: $\land \forall\, c \in Cowns$: $Len(queue[c]) = 0$
    $\land\ Assert(\forall\, c \in Cowns : Sleeping(c)$, "$Termination\ with\ unscheduled\ cowns$")
  $\land \forall\, c \in Cowns : Sleeping(c)$
  $\land$ UNCHANGED $vars$

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

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

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

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

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

$\mathit{Unmute} \triangleq$
  LET $\mathit{invalid\_keys} \triangleq \{c \in$ DOMAIN $\mathit{mute} : \mathit{priority}[c] = 0\}$IN
  LET $\mathit{unmuting} \triangleq$ UNION $\mathit{Range}([k \in \mathit{invalid\_keys} \mapsto \mathit{LowPriority}(\mathit{mute}[k])])$IN
  $\land\ \mathit{unmuting} \neq \{\}$
  $\land\ \mathit{priority}' = [c \in \mathit{unmuting} \mapsto 0]\ @@\ \mathit{priority}$
  $\land\ \mathit{mute}' = [c \in \mathit{invalid\_keys} \mapsto \{\}]\ @@\ \mathit{mute}$
  $\land\ \mathit{scheduled}'\ = [c \in \mathit{unmuting} \mapsto \text{TRUE}]\ @@\ \mathit{scheduled}$
  $\land\ \text{UNCHANGED}\ \langle \mathit{fuel}, \mathit{queue}, \mathit{running}, \mathit{blocker}, \mathit{mutor} \rangle$

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

$\mathit{Next} \triangleq \mathit{Terminating} \lor \exists\, c \in \mathit{Cowns} : \mathit{Run}(c) \lor \mathit{Unmute}$

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

 Utility Functions

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

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

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

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

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

 https://$\mathit{github.com}$/tlaplus/Examples/blob/master/specifications/$\mathit{TransitiveClosure}$/$\mathit{TransitiveClosure.tla}\#L114$
$\mathit{TC}(R) \triangleq$
   $\text{LET}$
     $S \triangleq \{r[1] : r \in R\} \cup \{r[2] : r \in R\}$
     $\text{RECURSIVE}\ \mathit{TCR}(\_)$
     $\mathit{TCR}(T) \triangleq$
       $\text{IF}\ T = \{\}\ \text{THEN}\ R$
        $\text{ELSE}$
         $\text{LET}$
           $r \triangleq \text{CHOOSE}\ s \in T : \text{TRUE}$
           $\mathit{RR} \triangleq \mathit{TCR}(T \setminus \{r\})$
        IN
          $RR \cup \{\langle s, t \rangle \in S \times S : \langle s, r \rangle \in RR \wedge \langle r, t \rangle \in RR\}$
    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.

$RunningIsScheduled \triangleq$

$\forall c \in Cowns : running[c] \Rightarrow scheduled[c] \wedge (c = Max(CurrentMessage(c)))$

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*.

$Nonblocking \triangleq$

```
  $\forall c \in Cowns : \forall m \in Range(queue[c]) :$
    $\forall \langle l, h \rangle \in LowPriority(m) \times HighPriority(m) : (c \leq h) \vee (c < l)$
```

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.

$UnscheduledByMuteOrAcquire \triangleq$

$\forall c \in Cowns : \neg((priority[c] = -1) \vee Acquired(c)) \equiv scheduled[c]$

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*.

$$
\begin{array}{l}
\mathit{AcquiredOnce} \triangleq \\
\quad \forall \langle a,\ b,\ c \rangle \in \mathit{Cowns} \times \mathit{Cowns} \times \mathit{Cowns} : \\
\qquad (\mathit{AcquiredBy}(a,\ b) \land \mathit{AcquiredBy}(a,\ c)) \Rightarrow (b = c)
\end{array}
$$

A message in a *cown*'s queue must contain the *cown*.

$$
\begin{array}{l}
\mathit{SelfInCurrentMessage} \triangleq \\
\quad \forall\, c \in \mathit{Cowns} : (\mathit{Len}(\mathit{queue}[c]) > 0) \Rightarrow (c \in \mathit{CurrentMessage}(c))
\end{array}
$$

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

$$
\begin{array}{l}
\mathit{HighPriorityInUnblockedQueue} \triangleq \\
\quad \forall\, c \in \mathit{HighPriority}(\mathit{Cowns}) : \\
\qquad \exists\, k \in \mathit{HighPriority}(\mathit{Cowns}) : c \in \textsc{union}\ \mathit{Range}(\mathit{queue}[k])
\end{array}
$$

Warning: not enforced by implementation.

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

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

$$
\begin{array}{l}
\mathit{HighPriorityHasWork} \triangleq \forall\, c \in \mathit{HighPriority}(\mathit{Cowns}) : \\
\quad \lor\ \mathit{Len}(\mathit{queue}[c]) > 0 \\
\quad \lor\ \mathit{Acquired}(c)
\end{array}
$$

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

$$
\begin{array}{l}
\mathit{MuteSetsDisjoint} \triangleq \forall \langle a,\ b \rangle \in \mathit{Cowns} \times \mathit{Cowns} : \\
\quad ((\mathit{mute}[a] \cap \mathit{mute}[b]) \qquad \neq \{\}) \Rightarrow (a = b)
\end{array}
$$

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

$$
\mathit{AcyclicTCMute} \triangleq \neg \mathit{CylcicTransitiveClosure}(\mathit{MutedBy})
$$