```
──────────────── MODULE backpressure ────────────────
EXTENDS FiniteSets, Integers, Sequences, TLC

Null ≜ 0
Cowns ≜ 1 . . 3 # TODO: 4
MaxMessageCount ≜ 3
MaxMessageSize ≜ 3
OverloadThreshold ≜ 2
PriorityLevels ≜ {2, 1, 0}

Pick(s) ≜ CHOOSE x ∈ s : TRUE
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}
Subsets(s, min, max) ≜
  {x ∈ SUBSET s : (Cardinality(x) ≥ min) ∧ (Cardinality(x) ≤ max)}
RECURSIVE Concat(_)
Concat(s) ≜ IF s = {} THEN ⟨⟩ ELSE LET x ≜ Pick(s)IN   x ∘ Concat(s \ {x})

VARIABLES fuel, queue, scheduled, running, mutor
vars ≜ ⟨fuel, queue, scheduled, running, mutor⟩

Messages ≜ UNION {Range(queue[c]) : c ∈ Cowns}
EmptyQueue(c) ≜ Len(queue[c]) = 0
Overloaded(c)   ≜ Len(queue[c]) ≥ OverloadThreshold
Enqueue(c, m)  ≜ c :> Append(queue[c], m)
Dequeue(c) ≜ c :> Tail(queue[c])

Init ≜
   ∧ fuel = MaxMessageCount
   ∧ queue = [c ∈ Cowns ↦ ⟨{c}⟩]
   ∧ scheduled = [c ∈ Cowns ↦ TRUE]
   ∧ running = [c ∈ Cowns ↦ FALSE]
   ∧ mutor = [c ∈ Cowns ↦ Null]

Terminating ≜
   ∧ ∀ c ∈ Cowns : EmptyQueue(c)
   ─────
   ∧ UNCHANGED vars

ExternalReceive(cown) ≜
   ∧ fuel > 0
   ─────
   ∧ UNCHANGED ⟨scheduled, running, mutor⟩
   ∧ fuel′ = fuel − 1
    # Receive a message from an external source
  $\land \exists\, others \in Subsets(\{c \in Cowns : c > cown\}, 0, MaxMessageSize - 1) :$
    $queue' = Enqueue(cown, \{cown\} \cup others) \mathbin{@@} queue$

$Acquire(cown) \triangleq$
  $\land\ scheduled[cown]$
  $\land \neg running[cown]$
  $\land \neg EmptyQueue(cown)$
  $\land\ cown \in Head(queue[cown])$
  $\land\ cown < Max(Head(queue[cown]))$
  ———
  $\land$ UNCHANGED $\langle fuel, scheduled, running, mutor \rangle$
   # Forward the message to the next $cown$.
  $\land$ LET
      $msg \triangleq Head(queue[cown])$
      $next \triangleq Min(\{c \in msg : c > cown\})$
    IN
      $queue' = Enqueue(next, msg) \mathbin{@@} Dequeue(cown) \mathbin{@@} queue$

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

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

$Send(cown) \triangleq$
  $\land\ running[cown]$
  $\land\ fuel > 0$
  ———
  $\land$ UNCHANGED $\langle scheduled, running \rangle$
  $\land\ fuel' = fuel - 1$
  # Select set of receivers
  $\land \exists\, receivers \in Subsets(Cowns, 1, MaxMessageSize)$ :
   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.
    $\land queue' = Enqueue(next, receivers)$ @@ $queue$
     # Set mutor if any receiver is overloaded and there are no receivers in the set of senders.
    $\land$ IF
      $\land mutors \neq \{\}$
      $\land mutor[cown] = Null$
      $\land (senders \cap receivers) = \{\}$
      THEN $mutor' = (cown :> Min(mutors))$ @@ $mutor$
      ELSE UNCHANGED $\langle mutor \rangle$

$PostRun(cown) \triangleq$
  $\land running[cown]$
   ———
  $\land$ UNCHANGED $\langle fuel \rangle$
  $\land running' = (cown :>$ FALSE$)$ @@ $running$
  $\land mutor' = (cown :> Null)$ @@ $mutor$
  $\land$ LET $msg \triangleq Head(queue[cown])$ IN
     # Mute if mutor is set.
   IF $(mutor[cown] \neq Null) \land (\forall\, c \in msg : \neg Overloaded(c))$ THEN
      $\land scheduled' = [c \in msg \mapsto$ FALSE$]$ @@ $scheduled$
       # Send unmute message to mutor
      $\land queue' = Enqueue(mutor[cown], msg)$ @@ $Dequeue(cown)$ @@ $queue$
    ELSE
      $\land$ UNCHANGED $\langle scheduled \rangle$
      $\land queue' = Dequeue(cown)$ @@ $queue$

$RunStep(cown) \triangleq$
    $\lor ExternalReceive(cown)$ \*# Very expensive check
  $\lor Acquire(cown)$
  $\lor Unmute(cown)$
  $\lor PreRun(cown)$
  $\lor Send(cown)$
  $\lor PostRun(cown)$

$Next \triangleq \exists\, c \in Cowns : RunStep(c)$

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

```
    ∧ ∀ 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 ≜ Cardinality(Messages) ≤ (Cardinality(Cowns) + MaxMessageCount)

 # Cowns are acquired by one running message at a time.
UniqueAcquisition ≜
  LET msgs ≜ Concat({⟨Head(queue[c])⟩ : c ∈ {k ∈ Cowns : running[k]}})
  IN  Cardinality(Range(msgs)) = Len(msgs)

 # Each queue has at most one token message.
LoneToken ≜ ∀ c ∈ Cowns : Len(SelectSeq(queue[c], LAMBDA m : m = {})) ≤ 1

 # A running cown must be scheduled and be the max cown in the message at the head of its queue.
RunningImplication ≜
  ∀ c ∈ Cowns : running[c] ⇒ scheduled[c] ∧ (c = Max(Head(queue[c])))
```