MODULE *backpressure*

Assumptions :
- Fairness (weakly fair *behaviour* process)
- *Cowns* cannot become overloaded while muted.
- Mute map entries will eventually be removed and unmuted.
- Modeled by having overloaded cowns eventually become not overloaded.

Note : - Each while iteration is an atomic step.

EXTENDS TLC, $Integers$, $FiniteSets$

CONSTANT $null$
$Cowns \triangleq 1 \,..\, 2$
$Behaviours \triangleq 1 \,..\, 4$

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$
$Min(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s \setminus \{x\} : y > x$
$Intersection(a,\, b) \triangleq a \cap b \neq \{\}$
$Subsets(s,\, min,\, max) \triangleq$
$\{cs \in \text{SUBSET } s : Cardinality(cs) \geq min \wedge Cardinality(cs) \leq max\}$

--algorithm *backpressure*

variables
$available = Cowns$,
$overloaded = \{\}$,
$muted = \{\}$,
$unmutable = \{\}$,
$mute\_map = [c \in Cowns \mapsto \{\}]$,
$refcount \;\; = [c \in Cowns \mapsto 0]$,
$rc\_barrier = 0$ **;**

define
$MutedInv \triangleq (available \cup overloaded) \cap muted = \{\}$
$UnmutableInv \triangleq unmutable \cap muted = \{\}$
$RefcountInv \triangleq \forall\, c \in Cowns : refcount[c] \geq 0$
$MuteMapInv \triangleq \forall\, m \in muted : m \in \text{UNION } Range(mute\_map)$
$TypeInvariant \triangleq MutedInv \wedge RefcountInv \wedge MuteMapInv$

$RefcountDrop \triangleq \Diamond\Box(\forall\, c \in Cowns : refcount[c] = 0)$
$WillUnmute \triangleq$
$\Box\Diamond(\forall\, k \in \text{DOMAIN } mute\_map : mute\_map[k] = \{\} \vee k \in overloaded)$
$TemporalProp \triangleq RefcountDrop \wedge WillUnmute$

$IncRef(inc) \triangleq [c \in Cowns \mapsto \text{IF } c \in inc \text{ THEN } refcount[c] + 1 \text{ ELSE } refcount[c]]$
$DecRef(dec) \triangleq [c \in Cowns \mapsto \text{IF } c \in dec \text{ THEN } refcount[c] - 1 \text{ ELSE } refcount[c]]$
$TriggersUnmute(mutor) \triangleq mutor \notin overloaded \vee refcount[mutor] = 0$
end define ;

fair process *behaviour* $\in$ *Behaviours*
variables
  $required \in Subsets(Cowns, 0, 3)$,
  $acquired = \{\}$, $next = null$,
  $muting = \{\}$, $mutor = null$,
  $unmute\_set = null$,
begin
Create:
  $refcount := IncRef(required)$;
  $rc\_barrier := rc\_barrier + 1$;
    Empty required set used to represent fewer behaviours in the system.
  if $required = \{\}$ **then goto** *Done* ; **end if** ;

RCBarrier:
  await $rc\_barrier = Cardinality(Behaviours)$;

Acquire:
  $next := Min(required)$;
  if $Intersection(overloaded,\ acquired \cup required) \wedge (next \in muted)$ **then**
      Make *unmutable* and schedule
    $unmutable := unmutable \cup \{next\}$;
    $muted := muted \setminus \{next\}$;
    $available := available \cup \{next\}$;
   else
      Acquire cown
    await $next \in available$;
    $acquired := acquired \cup \{next\}$;
    $required := required \setminus \{next\}$;
    $available := available \setminus \{next\}$;
  end if ;
  if $required \neq \{\}$ **then**
    goto *Acquire* ;
  end if ;

Action:
  assert $acquired \cap muted = \{\}$;
  if $overloaded \neq \{\} \wedge \neg Intersection(acquired,\ overloaded)$ **then**
    either
      with $mutor\_ \in overloaded$ **do** $mutor := mutor\_$ **end with** ;
      $muting := acquired \setminus unmutable$;
    or
      skip ;
    end either ;
  end if ;

Complete:

Arbitrarily toggle overloaded state of some acquired cowns.
with $overloading \in Subsets(acquired \setminus muting,\ 0,\ 3)$ **do**
with $unoverloading \in Subsets(acquired \cap overloaded,\ 0,\ 3)$ **do**
$overloaded := (overloaded \cup overloading) \setminus unoverloading$;
end with ;
end with ;

if $mutor \neq null$ **then**
$muted := muted \cup muting$;
$mute\_map[mutor] := mute\_map[mutor] \cup muting$;
end if ;

$available := available \cup (acquired \setminus muting)$;
$refcount := DecRef(acquired)$;

MuteMapScan:
$unmute\_set :=$
UNION $Range([c \in \{k \in Cowns : TriggersUnmute(k)\} \mapsto mute\_map[c]])$;
$mute\_map := [c \in Cowns \mapsto$ IF $TriggersUnmute(c)$ THEN $\{\}$ ELSE $mute\_map[c]]$;
$muted := muted \setminus unmute\_set$;
$available := available \cup unmute\_set$;
end process ;

end algorithm ;