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*

$Cowns \triangleq 1 \,..\, 4$

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

--algorithm *backpressure*

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

define
$BehaviourCount \triangleq 3$

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

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

$ZeroRC(c) \triangleq rc\_barrier = BehaviourCount \land refcount[c] = 0$
$TriggersUnmute(mutor) \triangleq mutor \in overloaded \lor ZeroRC(mutor)$
end define ;

fair process $behaviour \in 1 \,..\, BehaviourCount$
variables
$required \in Subsets(Cowns,\, 0,\, 3)$,
$next$, $acquired = \{\}$, $mutor$, $muting = \{\}$, $unmute\_set$
begin
Send:
$refcount :=$
$[c \in Cowns \mapsto \text{IF } c \in required \text{ THEN } refcount[c] + 1 \text{ ELSE } refcount[c]]$ **;**
$rc\_barrier := rc\_barrier + 1$ **;**
Empty required set used to represent fewer behaviours in the system.
if $required = \{\}$ **then goto** *Done* **; end if ;**

Unmute:
while $(\exists\, r \in required : r \notin unmutable)$
$\land\, (overloaded \cap required \neq \{\})$

```
    do
      next := Min({r ∈ required : r ∉ unmutable}) ;
      unmutable := unmutable ∪ {next} ;
      if next ∈ muted then
        muted := muted \ {next} ;
        available := available ∪ {next} ;
      end if ;
    end while ;

Acquire:
    while required ≠ {} do
      next := Min(required) ;
      await next ∈ available ;
      acquired := acquired ∪ {next} ;
      required := required \ {next} ;
      available := available \ {next} ;
    end while ;

Action:
    assert required = {} ;
    assert acquired ∩ muted = {} ;
    if (overloaded ≠ {}) ∧ (acquired ∩ overloaded = {}) then
      either
        with mutor_ ∈ overloaded do mutor := mutor_end with ;
        muting := acquired \ unmutable ;
      or
        skip ;
      end either ;
    end if ;

Complete:
      Arbitrarily toggle overloaded state of some acquired cowns.
    with overloading ∈ Subsets(acquired \ muting, 0, 3) do
      with unoverloading ∈ Subsets(acquired ∩ overloaded, 0, 3) do
        overloaded := (overloaded ∪ overloading) \ unoverloading ;
      end with ;
    end with ;

    if mutor ≠ defaultInitValue then
      muted := muted ∪ muting ;
      mute_map[mutor] := mute_map[mutor] ∪ muting ;
    end if ;

    available := available ∪ (acquired \ muting) ;
    muting := {} ;
    refcount :=
      [c ∈ Cowns ↦ IF c ∈ acquired THEN refcount[c] − 1 ELSE refcount[c]] ;
    acquired := {} ;
    assert acquired ∪ required = {} ;

MuteMapScan:
    unmute_set :=
      UNION Range([c ∈ {k ∈ Cowns : TriggersUnmute(k)} ↦ mute_map[c]]) ;
    mute_map := [c ∈ Cowns ↦ IF TriggersUnmute(c) THEN {} ELSE mute_map[c]] ;
    muted := muted \ unmute_set ;
    available := available ∪ unmute_set ;
end process ;

end algorithm ;

  BEGIN TRANSLATION − the hash of the PCal code: PCal-bc8e2f6d1cef2a2ee2ced52555aa4e69
CONSTANT defaultInitValue
VARIABLES available, overloaded, muted, unmutable, mute_map, refcount,
```

$rc\_barrier$, pc

define statement

$BehaviourCount \triangleq 3$

$MutedInv \triangleq available \cap muted = \{\}$

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

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

$ZeroRC(c) \triangleq rc\_barrier = BehaviourCount \land refcount[c] = 0$
$TriggersUnmute(mutor) \triangleq mutor \in overloaded \lor ZeroRC(mutor)$

VARIABLES $required$, $next$, $acquired$, $mutor$, $muting$, $unmute\_set$

$vars \triangleq \langle available, overloaded, muted, unmutable, mute\_map, refcount,$
$rc\_barrier, pc, required, next, acquired, mutor, muting,$
$unmute\_set\rangle$

$ProcSet \triangleq (1 \,..\, BehaviourCount)$

$Init \triangleq$ Global variables
$\land\ available = Cowns$
$\land\ overloaded = \{\}$
$\land\ muted = \{\}$
$\land\ unmutable = \{\}$
$\land\ mute\_map = [c \in Cowns \mapsto \{\}]$
$\land\ refcount = [c \in Cowns \mapsto 0]$
$\land\ rc\_barrier = 0$
Process *behaviour*
$\land\ required \in [1 \,..\, BehaviourCount \to Subsets(Cowns, 0, 3)]$
$\land\ next = [self \in 1 \,..\, BehaviourCount \mapsto defaultInitValue]$
$\land\ acquired = [self \in 1 \,..\, BehaviourCount \mapsto \{\}]$
$\land\ mutor = [self \in 1 \,..\, BehaviourCount \mapsto defaultInitValue]$
$\land\ muting = [self \in 1 \,..\, BehaviourCount \mapsto \{\}]$
$\land\ unmute\_set = [self \in 1 \,..\, BehaviourCount \mapsto defaultInitValue]$
$\land\ pc = [self \in ProcSet \mapsto \text{“Send”}]$

$Send(self) \triangleq \land\ pc[self] = \text{“Send”}$
$\land\ refcount' = [c \in Cowns \mapsto \text{IF } c \in required[self] \text{ THEN } refcount[c] + 1 \text{ ELSE } refcount[c]]$
$\land\ rc\_barrier' = rc\_barrier + 1$
$\land\ \text{IF } required[self] = \{\}$
$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}]$
$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Unmute”}]$
$\land\ \text{UNCHANGED } \langle available, overloaded, muted, unmutable,$
$mute\_map, required, next, acquired, mutor,$
$muting, unmute\_set\rangle$

$Unmute(self) \triangleq \land\ pc[self] = \text{“Unmute”}$
$\land\ \text{IF } (\exists\, r \in required[self] : r \notin unmutable)$
$\land\ (overloaded \cap required[self] \neq \{\})$
$\text{THEN } \land\ next' = [next \text{ EXCEPT } ![self] = Min(\{r \in required[self] : r \notin unmutable\})]$
$\land\ unmutable' = (unmutable \cup \{next'[self]\})$
$\land\ \text{IF } next'[self] \in muted$
$\text{THEN } \land\ muted' = muted \setminus \{next'[self]\}$
$\land\ available' = (available \cup \{next'[self]\})$
$\text{ELSE } \land\ \text{TRUE}$

$\land$ UNCHANGED $\langle available, muted\rangle$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Unmute"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "Acquire"$]$
$\land$ UNCHANGED $\langle available, muted, unmutable, next\rangle$
$\land$ UNCHANGED $\langle overloaded, mute\_map, refcount, rc\_barrier,$
$required, acquired, mutor, muting, unmute\_set\rangle$

$Acquire(self) \triangleq$ $\land\ pc[self] =$ "Acquire"
$\land$ IF $required[self] \neq \{\}$
THEN $\land\ next' = [next$ EXCEPT $![self] = Min(required[self])]$
$\land\ next'[self] \in available$
$\land\ acquired' = [acquired$ EXCEPT $![self] = acquired[self] \cup \{next'[self]\}]$
$\land\ required' = [required$ EXCEPT $![self] = required[self] \setminus \{next'[self]\}]$
$\land\ available' = available \setminus \{next'[self]\}$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Acquire"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "Action"$]$
$\land$ UNCHANGED $\langle available, required, next,$
$acquired\rangle$
$\land$ UNCHANGED $\langle overloaded, muted, unmutable, mute\_map,$
$refcount, rc\_barrier, mutor, muting,$
$unmute\_set\rangle$

$Action(self) \triangleq$ $\land\ pc[self] =$ "Action"
$\land\ Assert(required[self] = \{\},$
"Failure of assertion at line 78, column 3.")
$\land\ Assert(acquired[self] \cap muted = \{\},$
"Failure of assertion at line 79, column 3.")
$\land$ IF $(overloaded \neq \{\}) \land (acquired[self] \cap overloaded = \{\})$
THEN $\land\ \lor\ \land \exists\, mutor\_ \in overloaded :$
$mutor' = [mutor$ EXCEPT $![self] = mutor\_]$
$\land\ muting' = [muting$ EXCEPT $![self] = acquired[self] \setminus unmutable]$
$\lor\ \land$ TRUE
$\land$ UNCHANGED $\langle mutor, muting\rangle$
ELSE $\land$ TRUE
$\land$ UNCHANGED $\langle mutor, muting\rangle$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Complete"$]$
$\land$ UNCHANGED $\langle available, overloaded, muted, unmutable,$
$mute\_map, refcount, rc\_barrier, required, next,$
$acquired, unmute\_set\rangle$

$Complete(self) \triangleq$ $\land\ pc[self] =$ "Complete"
$\land \exists\, overloading \in Subsets(acquired[self] \setminus muting[self], 0, 3) :$
$\exists\, unoverloading \in Subsets(acquired[self] \cap overloaded, 0, 3) :$
$overloaded' = (overloaded \cup overloading) \setminus unoverloading$
$\land$ IF $mutor[self] \neq defaultInitValue$
THEN $\land\ muted' = (muted \cup muting[self])$
$\land\ mute\_map' = [mute\_map$ EXCEPT $![mutor[self]] = mute\_map[mutor[self]] \cup muting[self]]$
ELSE $\land$ TRUE
$\land$ UNCHANGED $\langle muted, mute\_map\rangle$
$\land\ available' = (available \cup (acquired[self] \setminus muting[self]))$
$\land\ muting' = [muting$ EXCEPT $![self] = \{\}]$
$\land\ refcount' = [c \in Cowns \mapsto$ IF $c \in acquired[self]$ THEN $refcount[c] - 1$ ELSE $refcount[c]]$
$\land\ acquired' = [acquired$ EXCEPT $![self] = \{\}]$
$\land\ Assert(acquired'[self] \cup required[self] = \{\},$
"Failure of assertion at line 107, column 3.")
$\land\ pc' = [pc$ EXCEPT $![self] =$ "MuteMapScan"$]$
$\land$ UNCHANGED $\langle unmutable, rc\_barrier, required, next, mutor,$
$unmute\_set\rangle$

$MuteMapScan(self) \triangleq$ $\land\ pc[self] =$ "MuteMapScan"
$\land\ unmute\_set' = [unmute\_set$ EXCEPT $![self] =$

$$\text{UNION } \mathit{Range}([c \in \{k \in \mathit{Cowns} : \mathit{TriggersUnmute}(k)\} \mapsto \mathit{mute\_map}[c]])]$$
$$\land\ \mathit{mute\_map}' = [c \in \mathit{Cowns} \mapsto \text{IF } \mathit{TriggersUnmute}(c) \text{ THEN } \{\} \text{ ELSE } \mathit{mute\_map}[c]]$$
$$\land\ \mathit{muted}' = \mathit{muted} \setminus \mathit{unmute\_set}'[\mathit{self}]$$
$$\land\ \mathit{available}' = (\mathit{available} \cup \mathit{unmute\_set}'[\mathit{self}])$$
$$\land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\mathit{self}] = \text{“Done”}]$$
$$\land\ \text{UNCHANGED } \langle \mathit{overloaded}, \mathit{unmutable}, \mathit{refcount}, \mathit{rc\_barrier}, \mathit{required}, \mathit{next}, \mathit{acquired}, \mathit{mutor}, \mathit{muting}\rangle$$

$$\mathit{behaviour}(\mathit{self}) \triangleq \mathit{Send}(\mathit{self}) \lor \mathit{Unmute}(\mathit{self}) \lor \mathit{Acquire}(\mathit{self}) \lor \mathit{Action}(\mathit{self}) \lor \mathit{Complete}(\mathit{self}) \lor \mathit{MuteMapScan}(\mathit{self})$$

Allow infinite stuttering to prevent deadlock on termination.

$$\mathit{Terminating} \triangleq \land \forall\, \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{“Done”} \land\ \text{UNCHANGED } \mathit{vars}$$

$$\mathit{Next} \triangleq (\exists\, \mathit{self} \in 1 \mathinner{..} \mathit{BehaviourCount} : \mathit{behaviour}(\mathit{self})) \lor \mathit{Terminating}$$

$$\mathit{Spec} \triangleq \land \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \forall\, \mathit{self} \in 1 \mathinner{..} \mathit{BehaviourCount} : \mathrm{WF}_{\mathit{vars}}(\mathit{behaviour}(\mathit{self}))$$

$$\mathit{Termination} \triangleq \Diamond(\forall\, \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{“Done”})$$

END *TRANSLATION* – the hash of the generated TLA code (remove to silence divergence warnings): TLA-bd24460ac3e3dbc5294ac3189bb484af