MODULE *backpressure*

Note: Each while iteration is an atomic step.

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

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

$Subsets(s,\, min,\, max) \triangleq$
$\{cs \in \text{SUBSET } s : Cardinality(cs) \geq min \land Cardinality(cs) \leq max\}$

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$
$Min(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s \setminus \{x\} : y > x$

--algorithm *backpressure*

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

define
$BehaviourCount \triangleq 3$
end define ;

fair process *behaviour* $\in 1 \,..\, BehaviourCount$
variables
$required \in Subsets(Cowns,\, 1,\, 3)$, $next$, $acquired = \{\}$, $mutor$, $muting = \{\}$
begin
Send:
$refcount :=$
$[c \in Cowns \mapsto \text{IF } c \in required \text{ THEN } refcount[c] + 1 \text{ ELSE } refcount[c]]$;
$rc\_barrier := rc\_barrier + 1$;
Unmute:
while $\exists\, r \in required : r \notin unmutable$ **do**
$next := Min(\{r \in required : r \notin unmutable\})$;
if $next \in muted$ **then**
$muted := muted \setminus \{next\}$;
$available := available \cup \{next\}$;
end if ;
$unmutable := unmutable \cup \{next\}$;
end while ;
Acquire:
while $required \neq \{\}$ **do**
$next := Min(required)$;
await $next \in available$;
$acquired := acquired \cup \{next\}$;
$required := required \setminus \{next\}$;
$available := available \setminus \{next\}$;
end while ;
Action:
assert $required = \{\}$;
with $overloading \in Subsets(acquired \setminus muted,\, 0,\, 3)$ **do**
$overloaded := overloaded \cup overloading$;
end with ;
if $(overloaded \neq \{\}) \land (acquired \cap overloaded = \{\})$ **then**
either
with $mutor\_ \in overloaded$ **do** $mutor := mutor\_$**end with** ;
$muting := (acquired \setminus unmutable)$;

```
    or
      skip ;
    end either ;
  end if ;
Complete:
  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 = {} ;
end process ;

fair process mute_map_scan = 0
  variables next ;
begin
BarrierWait:
  await rc_barrier = BehaviourCount ;
MutorWait:
  while ∃ c ∈ Cowns : mute_map[c] ≠ {} do
    await ∃ c ∈ Cowns : (mute_map[c] ≠ {}) ∧ (refcount[c] = 0) ;
UnmuteSet:
    next := CHOOSE c ∈ Cowns : (mute_map[c] ≠ {}) ∧ (refcount[c] = 0) ;
    muted := muted \ mute_map[next] ;
    available := available ∪ mute_map[next] ;
    mute_map[next] := {} ;
  end while ;
end process ;

end algorithm ;
```

BEGIN *TRANSLATION* − the hash of the *PCal* code: PCal-9aaefd9f6bfbac1619db41fed353a4d7

Process variable next of process *behaviour* at line 32 col 38 changed to *next_*

CONSTANT *defaultInitValue*

VARIABLES *available*, *overloaded*, *muted*, *unmutable*, *mute_map*, *refcount*, *rc_barrier*, *pc*

define statement

$BehaviourCount \triangleq 3$

VARIABLES *required*, *next_*, *acquired*, *mutor*, *muting*, *next*

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

$ProcSet \triangleq (1 \,..\, BehaviourCount) \cup \{0\}$

$Init \triangleq$ Global variables
$\wedge\ available = Cowns$
$\wedge\ overloaded = \{\}$
$\wedge\ muted = \{\}$
$\wedge\ unmutable = \{\}$
$\wedge\ mute\_map = [c \in Cowns \mapsto \{\}]$
$\wedge\ refcount = [c \in Cowns \mapsto 0]$
$\wedge\ rc\_barrier = 0$
Process *behaviour*
$\wedge\ required \in [1 \,..\, BehaviourCount \to Subsets(Cowns, 1, 3)]$
$\wedge\ next\_ = [self \in 1 \,..\, BehaviourCount \mapsto defaultInitValue]$
$\wedge\ acquired = [self \in 1 \,..\, BehaviourCount \mapsto \{\}]$

$\land\ mutor = [self \in 1 \,..\, BehaviourCount \mapsto defaultInitValue]$
$\land\ muting = [self \in 1 \,..\, BehaviourCount \mapsto \{\}]$
Process *mute_map_scan*
$\land\ next = defaultInitValue$
$\land\ pc = [self \in ProcSet \mapsto$ CASE $self \in 1 \,..\, BehaviourCount \to$ "Send"
$\quad\ \square\ \ self = 0 \to$ "BarrierWait"$]$

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

$Unmute(self) \triangleq$ $\land\ pc[self] =$ "Unmute"
$\land$ IF $\exists\, r \in required[self] : r \notin unmutable$
THEN $\land\ next\_' = [next\_$ EXCEPT $![self] = Min(\{r \in required[self] : r \notin unmutable\})]$
$\land$ IF $next\_'[self] \in muted$
THEN $\land\ muted' = muted \setminus \{next\_'[self]\}$
$\land\ available' = (available \cup \{next\_'[self]\})$
ELSE $\land$ TRUE
$\land$ UNCHANGED $\langle available, muted\rangle$
$\land\ unmutable' = (unmutable \cup \{next\_'[self]\})$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Unmute"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "Acquire"$]$
$\land$ UNCHANGED $\langle available, muted, unmutable, next\_\rangle$
$\land$ UNCHANGED $\langle overloaded, mute\_map, refcount, rc\_barrier,$
$required, acquired, mutor, muting, next\rangle$

$Acquire(self) \triangleq$ $\land\ pc[self] =$ "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 $\langle overloaded, muted, unmutable, mute\_map,$
$refcount, rc\_barrier, mutor, muting, next\rangle$

$Action(self) \triangleq$ $\land\ pc[self] =$ "Action"
$\land\ Assert(required[self] = \{\},$
"Failure of assertion at line 56, column 3.")
$\land\ \exists\, overloading \in Subsets(acquired[self] \setminus muted, 0, 3) :$
$overloaded' = (overloaded \cup overloading)$
$\land$ 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 $\langle available, muted, unmutable, mute\_map,$
$refcount, rc\_barrier, required, next\_,$
$acquired, next\rangle$

$Complete(self) \triangleq$ $\land\ pc[self] =$ "Complete"
$\land$ 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 78, column 3.")
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Done"$]$
$\land$ UNCHANGED $\langle overloaded, unmutable, rc\_barrier, required,$
$next\_, mutor, next\rangle$

$behaviour(self) \triangleq Send(self) \lor Unmute(self) \lor Acquire(self)$
$\lor Action(self) \lor Complete(self)$

$BarrierWait \triangleq$ $\land\ pc[0] =$ "BarrierWait"
$\land\ rc\_barrier = BehaviourCount$
$\land\ pc' = [pc$ EXCEPT $![0] =$ "MutorWait"$]$
$\land$ UNCHANGED $\langle available, overloaded, muted, unmutable,$
$mute\_map, refcount, rc\_barrier, required, next\_,$
$acquired, mutor, muting, next\rangle$

$MutorWait \triangleq$ $\land\ pc[0] =$ "MutorWait"
$\land$ IF $\exists\, c \in Cowns : mute\_map[c] \neq \{\}$
THEN $\land\ \exists\, c \in Cowns : (mute\_map[c] \neq \{\}) \land (refcount[c] = 0)$
$\land\ pc' = [pc$ EXCEPT $![0] =$ "UnmuteSet"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![0] =$ "Done"$]$
$\land$ UNCHANGED $\langle available, overloaded, muted, unmutable, mute\_map,$
$refcount, rc\_barrier, required, next\_, acquired,$
$mutor, muting, next\rangle$

$UnmuteSet \triangleq$ $\land\ pc[0] =$ "UnmuteSet"
$\land\ next' = (\text{CHOOSE } c \in Cowns : (mute\_map[c] \neq \{\}) \land (refcount[c] = 0))$
$\land\ muted' = muted \setminus mute\_map[next']$
$\land\ available' = (available \cup mute\_map[next'])$
$\land\ mute\_map' = [mute\_map$ EXCEPT $![next'] = \{\}]$
$\land\ pc' = [pc$ EXCEPT $![0] =$ "MutorWait"$]$
$\land$ UNCHANGED $\langle overloaded, unmutable, refcount, rc\_barrier,$
$required, next\_, acquired, mutor, muting\rangle$

$mute\_map\_scan \triangleq BarrierWait \lor MutorWait \lor UnmuteSet$

Allow infinite stuttering to prevent deadlock on termination.
$Terminating \triangleq$ $\land \forall\, self \in ProcSet : pc[self] =$ "Done"
$\land$ UNCHANGED $vars$

$Next \triangleq$ $mute\_map\_scan$
$\lor (\exists\, self \in 1 \,..\, BehaviourCount : behaviour(self))$
$\lor Terminating$

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

$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] =$ "Done"$)$

END *TRANSLATION* – the hash of the generated TLA code (remove to silence divergence warnings): TLA-64c1ea2a00c06bb04c44c3cca8bb2c0a

$MutedInv \triangleq available \cap muted = \{\}$
$UnmutableInv \triangleq (overloaded \cup unmutable) \cap muted = \{\}$

$RefcountInv \triangleq \forall\, c \in Cowns : refcount[c] \geq 0$
$MuteMapInv \triangleq \forall\, m \in muted : m \in \text{UNION } Range(mute\_map)$
$MuteSetInv \triangleq \forall\, k \in \text{DOMAIN } mute\_map : (mute\_map[k] = \{\}) \lor (k \in overloaded)$

$TypeCorrect \triangleq MutedInv \land UnmutableInv \land RefcountInv \land MuteMapInv \land MuteSetInv$

$RefcountDrop \triangleq \Diamond\Box(\forall\, c \in Cowns : refcount[c] = 0)$

$Correct \triangleq Termination$