MODULE *IPCProtocol*

In *Linux* versions before 2.6.11, the capacity of a pipe was the same as the system page size (*e.g.*, 4096 bytes on *i*386). Since *Linux* 2.6.11, the pipe capacity is 16 pages (*i.e.*, 65,536 bytes in a system with a page size of 4096 bytes).

$N = 20$
$Messsage = \{$"*ping*", "*pong*", "*queryPort*", "*replyPort*"$\}$
$MessagePairs = \{[msgIn \mapsto$ "*ping*", $msgOut \mapsto$ "*pong*"$], [msgIn \mapsto$ "*queryPort*", $msgOut \mapsto$ "*replyPort*"$]\}$

EXTENDS *Naturals*, *Sequences*

VARIABLES *inQueue*, *outQueue*

CONSTANT *Message*, *MessagePairs*, *N*
ASSUME $(N \in Nat) \land (N > 0)$ Both queues have the same number of messages
ASSUME $(\forall\, msgPair \in MessagePairs :$ $\land\, msgPair.msgIn \in Message$
$\land\, msgPair.msgOut \in Message$
$\land\, msgPair.msgIn \neq msgPair.msgOut)$

A simple type invariant
$TypeOK \triangleq \land \forall\, msgPair \in MessagePairs : msgPair.msgIn \neq msgPair.msgOut$

Util function
$Last(s) \triangleq s[Len(s)]$

---

* We would usually use the existing structures, but the problem with these is that
* they have a step that chooses a random message when we advance their state using next.
InQueue $\triangleq$ INSTANCE *BoundedFIFO* WITH in $\leftarrow$ *inQueueIn*, out $\leftarrow$ *inQueueOut*, $q \leftarrow$ *inQueue*
OutQueue $\triangleq$ INSTANCE *BoundedFIFO* WITH in $\leftarrow$ *outQueueIn*, out $\leftarrow$ *outQueueOut*, $q \leftarrow$ *outQueue*

Make sure that once the message goes in, eventually it must go out as it's pair response
$MsgIncl \triangleq$
$\forall\, msgPair \in MessagePairs :$
$(Len(inQueue) > 0) \Rightarrow Last(inQueue) = msgPair.msgIn \rightsquigarrow Head(outQueue) = msgPair.msgOut$

When theys start they are empty
$Init \triangleq \land\, inQueue = \langle\rangle$
$\land\, outQueue = \langle\rangle$

If the input queue is empty, we presume that the sender awaits for the response.
$BNext \triangleq \land$ IF $Len(inQueue) = 0$
Append a new message to the in queue
THEN $\exists\, msgPair \in MessagePairs :$ $\land\, inQueue' = Append(inQueue, msgPair.msgIn)$
$\land$ UNCHANGED $(outQueue)$

Advances a step, removes a message from in queue and appends it's pair to out queue

ELSE $\exists\, msgPair \in MessagePairs :$ $\land\, inQueue' = Tail(inQueue)$
$\land\, outQueue' = Append(outQueue,\, msgPair.msgOut)$

$\land\, Len(inQueue) < N$ Bounded $inQueue$, this gets discharged, but let's keep our invariants

$\land\, Len(outQueue) < N$ Bounded $outQueue$

$Spec \triangleq \land\, MsgIncl$
$\land\, Init$
$\land\, \Box[BNext]_{\langle inQueue,\, outQueue\rangle}$

THEOREM $Spec \Rightarrow \Box\, TypeOK$
