MODULE *Peterson*

EXTENDS *Integers*

VARIABLES

turn, This is one of the three items that mutate in *Peterson*'s algorithm. It is what keeps track of which process's turn it is to access the shared resource

processState, This is a function mapping each process (represented by an integer 0 or 1) to its current state (a string).

flag This contains the remaining two of the three items that mutate in *Peterson*′ algorithm. It is a function mapping each process (represented by an integer 0 or 1) to a boolean flag indicating whether the process would like to access the shared resource.

$vars \triangleq \langle turn, processState, flag \rangle$ This is simply a convenient grouping of variables to use later on when defining temporal properties to save on tedious typing.

$TypeOK \triangleq$
- $\land \quad \forall\, p \in \{0, 1\} : flag[p] \in \{\text{TRUE, FALSE}\}$
- $\land \quad turn \in \{0, 1\}$
- $\land \quad \forall\, p \in \{0, 1\} : processState[p] \in \{\text{“idle”, “sentRequest”, “waiting”, “critical”}\}$

$TypeOk2 \triangleq$
- $\land\, turn \in \{0, 1\}$
- $\land\, flag \in [\{0, 1\} \to \{\text{TRUE, FALSE}\}]$
- $\land\, processState \in [\{0, 1\} \to \{\text{“idle”, “sentRequest”, “waiting”, “critical”}\}]$

This is one possible initial state of *Peterson*'s algorithm. It is the most “natural” one, that is the one where both processes are idle and have not yet decided to access the resource.

$Init \triangleq$
- $\land\, flag = [i \in \{0, 1\} \mapsto \text{FALSE}]$
- $\land\, turn \in \{0, 1\}$
- $\land\, processState = [i \in \{0, 1\} \mapsto \text{“idle”}]$

This predicate describes how a process can send a request to access a resource, which essentially amounts to setting its request flag to true.

$ProcessRequestFlag(p) \triangleq$
- $\land\, processState[p] = \text{“idle”}$
- $\land\, flag' = [flag \text{ EXCEPT } ![p] = \text{TRUE}]$
- $\land\, processState' = [processState \text{ EXCEPT } ![p] = \text{“sentRequest”}]$
- $\land \text{ UNCHANGED } \langle turn \rangle$

$ProcessBeginWaiting(p) \triangleq$
- $\land\, processState[p] = \text{“sentRequest”}$
- $\land\, turn' = 1 - p$
- $\land\, processState' = [processState \text{ EXCEPT } ![p] = \text{“waiting”}]$
- $\land \text{ UNCHANGED } \langle flag \rangle$

$ProcessEnterCritical(p) \triangleq$
$\quad \land\ processState[p] = \text{“waiting”}$
$\quad \land\ (flag[(1-p)] = \text{FALSE} \lor turn = p)$
$\quad \land\ processState' = [processState \text{ EXCEPT } ![p] = \text{“critical”}]$
$\quad \land\ \text{UNCHANGED } \langle flag,\ turn \rangle$

$ProcessExitCritical(p) \triangleq$
$\quad \land\ processState[p] = \text{“critical”}$
$\quad \land\ processState' = [processState \text{ EXCEPT } ![p] = \text{“idle”}]$
$\quad \land\ flag' = [flag \text{ EXCEPT } ![p] = \text{FALSE}]$
$\quad \land\ \text{UNCHANGED } \langle turn \rangle$

$Next \triangleq$
$\quad \exists\, p \in \{0, 1\} :$
$\qquad \lor\ ProcessRequestFlag(p)$
$\qquad \lor\ ProcessBeginWaiting(p)$
$\qquad \lor\ ProcessEnterCritical(p)$
$\qquad \lor\ ProcessExitCritical(p)$

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

$SpecWithFairness \triangleq Spec \land \text{WF}_{vars}(Next) \land \forall\, p \in \{0, 1\} : \text{WF}_{vars}(ProcessRequestFlag(p))$

$MutualExclusion \triangleq \neg(processState[0] = \text{“critical”} \land processState[1] = \text{“critical”})$

THEOREM $Spec \Rightarrow \Box MutualExclusion$

This is a basic liveness requirement that corresponds to what is called “Progress” in the *Wikipedia* article. Both processes should eventually be able to enter their critical sections.

$WillEventuallyEnterCritical \triangleq \Diamond(processState[0] = \text{“critical”}) \land \Diamond(processState[1] = \text{“critical”})$

THEOREM $SpecWithFairness \Rightarrow WillEventuallyEnterCritical$

THIS INVARIANT DOES NOT *HOLD* AND SHOULD NOT *HOLD*! It's merely instructive of something a reader may intuitively believe about this algorithm that turns out to be false.

See the note in *ProcessEnterCritical*.

$CanOnlyBeCriticalIfTurn \triangleq \forall\, p \in \{0, 1\} : processState[p] = \text{“critical”} \Rightarrow turn = p$

Finally we note simply that our variables should always stay within the bounds we enumerated earlier.

THEOREM $Spec \Rightarrow \Box\, TypeOK$

$Inv \triangleq \land\ TypeOK$
$\qquad \land \forall\, p,\ q \in \{0, 1\} : processState[p] = \text{“critical”} \Rightarrow (flag[q] = \text{FALSE} \lor turn = p)$
$\qquad \land\ MutualExclusion$

$Inv2 \triangleq \forall\, p,\ q : processState[p] \neq \text{“critical”}$
$Inv3 \triangleq \neg\exists\, p : processState[p] = \text{“critical”}$

THEOREM $InitProperty \triangleq Init \Rightarrow Inv$
BY DEF $Init$, Inv, $ProcessRequestFlag$, $ProcessBeginWaiting$, $ProcessEnterCritical$, $ProcessExitCritical$, T

THEOREM $processState = [i \in \{0, 1\} \mapsto \text{“idle”}] \Rightarrow Inv$
???

THEOREM $InductiveProperty \triangleq Inv \land Next \Rightarrow Inv'$
BY DEF Inv, $Next$, $MutualExclusion$, $TypeOK$

$SpecIndCheck \triangleq TypeOk2 \land Inv \land \Box[Next]_{vars}$

* Modification History
* Last modified *Wed* May 28 12:21:03 *EDT* 2025 by *johnnguyen*
* Created *Wed* May 28 01:17:56 *EDT* 2025 by *johnnguyen*