MODULE *Sem0*

Checked with
$Pcs = \{1, 2, 3, 4\}$
$N = 5$

EXTENDS *Integers*
CONSTANTS *Pcs*, *N*
ASSUME $N \in Nat \land N \geq 1$

State space

VARIABLES *counter*, *state*, *ret*

$$
\begin{aligned}
vars &\triangleq \langle ret, counter, state \rangle \\
Blocking &\triangleq \{\text{“blocking”}, \text{“timed”}, \text{“non-blocking”}\} \\
PcsState &\triangleq \{\text{“idle”}, \text{“exception”}, \text{“timeout”}\} \cup Blocking \\
Result &\triangleq \{\text{“success”}, \text{“failure”}\} \\
TypeInv &\triangleq \land counter \in Nat \\
&\quad\;\; \land counter \leq N \\
&\quad\;\; \land state \in [Pcs \rightarrow PcsState] \\
&\quad\;\; \land ret \in Result
\end{aligned}
$$

Next-state relation

Requesting the semaphore can be done in one of three ways
 * blocking – wait until *counter* > 0 then /succeed/
 * non-blocking – if *counter* > 0 then succeed else /fail/
 * timed – wait until *counter* > 0 or t secs elapse
  – if *counter* > 0 then /succeed/
  – else (t secs have elapsed) /timeout/
For the meaning of /success/, /failure/ and /timeout/, see
Return statement below.

$$
\begin{aligned}
Request(b, p) &\triangleq \land b \in Blocking \\
&\quad\;\; \land state[p] = \text{“idle”} \\
&\quad\;\; \land state' = [state \text{ EXCEPT } ![p] = b] \\
&\quad\;\; \land \text{UNCHANGED } \langle counter, ret \rangle \\
ReturnSuccess(p) &\triangleq \land state[p] \in Blocking \\
&\quad\;\; \land counter > 0 \\
&\quad\;\; \land counter' = counter - 1 \\
&\quad\;\; \land ret' = \text{“success”} \\
&\quad\;\; \land state' = [state \text{ EXCEPT } ![p] = \text{“idle”}] \\
ReturnTimeout(p) &\triangleq \land state[p] = \text{“timeout”} \\
&\quad\;\; \land state' = [state \text{ EXCEPT } ![p] = \text{“exception”}] \\
&\quad\;\; \land \text{UNCHANGED } \langle counter, ret \rangle \\
ReturnFail(p) &\triangleq \land state[p] = \text{“non-blocking”} \\
&\quad\;\; \land counter = 0 \\
&\quad\;\; \land ret' = \text{“failure”} \\
&\quad\;\; \land state' = [state \text{ EXCEPT } ![p] = \text{“idle”}] \\
&\quad\;\; \land \text{UNCHANGED } counter
\end{aligned}
$$

Return statement: they can result in one of three

* Success – counter is decreased
* Failure – counter is unchanged
* *Timeout* – An exception is thrown, counter is unchanged

$$
\begin{array}{lll}
\mathit{Return}(p) & \triangleq & \lor\ \mathit{ReturnSuccess}(p) \\
 & & \lor\ \mathit{ReturnFail}(p) \\
 & & \lor\ \mathit{ReturnTimeout}(p) \\
\mathit{Release}(p) & \triangleq & \land\ \mathit{state}[p] = \text{“idle”} \\
 & & \land\ \mathit{counter} < N \\
 & & \land\ \mathit{counter}' = \mathit{counter} + 1 \\
 & & \land\ \text{UNCHANGED}\ \langle \mathit{state},\ \mathit{ret} \rangle \\
\mathit{Timeout}(p) & \triangleq & \land\ \mathit{state}[p] = \text{“timed”} \\
 & & \land\ \mathit{state}' = [\mathit{state}\ \text{EXCEPT}\ ![p] = \text{“timeout”}] \\
 & & \land\ \text{UNCHANGED}\ \langle \mathit{counter},\ \mathit{ret} \rangle
\end{array}
$$

Process specification

$$
\begin{array}{lll}
\mathit{OneProc}(p) & \triangleq & \lor\ (\exists\, b \in \mathit{Blocking} : \mathit{Request}(b,\ p)) \\
 & & \lor\ \mathit{Return}(p) \\
 & & \lor\ \mathit{Release}(p) \\
 & & \lor\ \mathit{Timeout}(p)
\end{array}
$$

System Specification

$$
\begin{array}{lll}
\mathit{Waiting}(p) & \triangleq & \mathit{state}[p] \in \mathit{Blocking} \\
\mathit{DeadLock} & \triangleq & \lor\ \land\ (\forall\, p \in \mathit{Pcs} : \mathit{Waiting}(p) \lor \mathit{state}[p] = \text{“exception”}) \\
 & & \quad\ \land\ \mathit{counter} = 0 \\
 & & \quad\ \land\ \text{UNCHANGED}\ \mathit{vars} \\
 & & \lor\ \land\ (\forall\, p \in \mathit{Pcs} : \mathit{state}[p] = \text{“exception”}) \\
 & & \quad\ \land\ \text{UNCHANGED}\ \mathit{vars}
\end{array}
$$

The *DeadLock* event states the only conditions under which the semaphore can cause a deadlock among a set of processes. The model-checker looks for deadlocks and adding this event tells it that this is a known issue and the model checker won't treat it as a fault.

$$
\begin{array}{lll}
\mathit{Init} & \triangleq & \land\ \mathit{counter} = 1 \\
 & & \land\ \mathit{ret} = \text{“success”} \\
 & & \land\ \mathit{state} = [p \in \mathit{Pcs} \mapsto \text{“idle”}] \\
\mathit{Next} & \triangleq & \lor\ (\exists\, p\ \ \in \mathit{Pcs} : \mathit{OneProc}(p)) \\
 & & \lor\ \mathit{DeadLock} \\
\mathit{Live} & \triangleq & \land\ (\forall\, p \in \mathit{Pcs} : \mathrm{SF}_{\mathit{vars}}(\mathit{Return}(p))) \\
 & & \land\ (\forall\, p \in \mathit{Pcs} : \mathrm{WF}_{\mathit{vars}}(\mathit{Timeout}(p))) \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \mathit{Live}
\end{array}
$$

Properties

$$
\mathit{BoundedWait} \triangleq \Box\Diamond(\mathit{counter} > 0) \Rightarrow (\forall\, p \in \mathit{Pcs} : \Box\Diamond(\neg \mathit{Waiting}(p)))
$$

As long as processes keep releasing the semaphore, no process waits forever

$$NonBlocking \triangleq (\forall p \in Pcs : \Box\Diamond(state[p] \neq \text{"non-blocking"} \land state[p] \neq \text{"timed"}))$$

No matter the circumstances, no process stays blocked
in non-blocking mode or in timed mode

$$\begin{aligned} Disj(p) \triangleq\ & \land \neg(\text{ENABLED } ReturnSuccess(p) \land \text{ENABLED } ReturnTimeout(p)) \\ & \land \neg(\text{ENABLED } ReturnSuccess(p) \land \text{ENABLED } ReturnFail(p)) \\ & \land \neg(\text{ENABLED } ReturnFail(p) \quad\ \land \text{ENABLED } ReturnTimeout(p)) \end{aligned}$$

$$Disjoint \triangleq (\forall p \in Pcs : Disj(p))$$

The outcome of *Return* is uniquely specified by the *Next*
state relation

```
******************************************************************************
    Request(b, p)
      // with $b \in \{\text{"blocking"}, \text{"non-blocking"}, \text{"timed"}\}$
      // in the python code, b would also specify a timeout
      // but we abstract away from time durations
    "waiting"
    $r \leftarrow Return(p)$
      //$r \in \{\text{"success"}, \text{"failure"}\}$
      //$state \in \{\text{"idle"}, \text{"exception"}\}$
    Release(p)
******************************************************************************
```

* Modification History
* Last modified Sat *Feb* 13 21:50:39 *EST* 2016 by *Simon*
* Created *Fri Feb* 12 16:09:23 *EST* 2016 by *Simon*