MODULE *dekkers_2*

Dekker's Algorithm built from scratch using only *Wikipedia*'s pseudocode [1] as reference.

[1] https://*en.wikipedia.org*/wiki/*Dekker*%27*s_algorithm*

EXTENDS *TLC*, *Integers*

$Threads \triangleq \{1, 2\}$

VARIABLES
want_to_enter,
turn,
pc

$vars \triangleq \langle want\_to\_enter, turn, pc \rangle$

Utilities

$Go(thread, state) \triangleq pc' = [pc \text{ EXCEPT } ![thread] = state]$

$Signal(thread, bool) \triangleq want\_to\_enter' = [want\_to\_enter \text{ EXCEPT } ![thread] = bool]$

$Init \triangleq$
$\land\ want\_to\_enter \in [Threads \to \{\text{FALSE}\}]$
$\land\ turn \in Threads$
$\land\ pc \in [Threads \to \{\text{"init"}\}]$

$WantToEnter(thread) \triangleq$
Thread sets its boolean flag and tries to enter the critical section
$\land\ pc[thread] = \text{"init"}$
$\land\ Signal(thread, \text{TRUE})$
$\land\ Go(thread, \text{"flag\_set"})$
$\land$ UNCHANGED *turn*

$WaitingOther(thread) \triangleq$
Thread checks for others intention to access the critical section. If no one is waiting it proceeds, if there are other threads with the same intention it waits it's turn
$\land\ pc[thread] = \text{"flag\_set"}$
$\land$ IF $\forall\, t \in Threads \setminus \{thread\} : want\_to\_enter[t]$
THEN $\land\ Go(thread, \text{"wait\_turn"})$
ELSE $\land\ Go(thread, \text{"enter\_critical"})$
$\land$ UNCHANGED $\langle want\_to\_enter, turn \rangle$

$WaitTurn(thread) \triangleq$

The thread checks to see if it can go ahead, if so it goes back to the waiting others step. If it can't, it unsets its flag and busy-waits

$\land\ pc[thread] = \text{“wait\_turn”}$
$\land$ IF $turn = thread$
THEN $\land\ Go(thread,\ \text{“flag\_set”})$
$\land$ UNCHANGED $\langle turn,\ want\_to\_enter \rangle$
ELSE $\land\ Signal(thread,\ \text{FALSE})$
$\land\ Go(thread,\ \text{“busy\_wait”})$
$\land$ UNCHANGED $turn$

$BusyWait(thread) \triangleq$

Thread busy waits for its turn, once it arrives it sets its flag and goes back to the initial wait state

$\land\ pc[thread] = \text{“busy\_wait”}$
$\land\ turn = thread$
$\land\ Signal(thread,\ \text{TRUE})$
$\land\ Go(thread,\ \text{“flag\_set”})$
$\land$ UNCHANGED $turn$

$CriticalSection(thread) \triangleq$

The critical section of the code. The $\land$ TRUE simulates real critical work

$\land\ pc[thread] = \text{“enter\_critical”}$
$\land$ TRUE perform critical work
$\land\ Go(thread,\ \text{“exit\_critical”})$
$\land$ UNCHANGED $\langle want\_to\_enter,\ turn \rangle$

$ExitCritical(thread) \triangleq$

Once done, the thread unsets it's intention flag and changes the turn to a $thread! =$ than itself. Then goes back to the initial state.

$\land\ pc[thread] = \text{“exit\_critical”}$
$\land\ Signal(thread,\ \text{FALSE})$
$\land \exists\ t \in Threads \setminus \{thread\} : turn' = t$
$\land\ Go(thread,\ \text{“init”})$

$ThreadOps(t) \triangleq$
$\lor\ WantToEnter(t)$
$\lor\ WaitingOther(t)$
$\lor\ WaitTurn(t)$
$\lor\ BusyWait(t)$
$\lor\ CriticalSection(t)$
$\lor\ ExitCritical(t)$

$Next \triangleq \exists\, t \in Threads : ThreadOps(t)$

$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge \forall\, t \in Threads : \mathrm{WF}_{vars}(ThreadOps(t))$

$TypeOk \triangleq$

$\wedge$ $Threads = \{1, 2\}$
$\wedge$ $turn \in \{1, 2\}$
$\wedge$ $want\_to\_enter \in$ BOOLEAN $\times$ BOOLEAN
$\wedge$ LET $valid\_state \triangleq \{$"init", "flag_set", "enter_critical", "exit_critical", "wait_turn", "busy_wait"$\}$
IN $\wedge\, pc \in valid\_state \times valid\_state$

$Safety \triangleq$

Only one thread at a time is allowed in the critical section.

$\vee \forall\, t \in Threads : pc[t] \neq$ "enter_critical"
$\vee \exists\, t \in Threads :$
$\wedge\, pc[t] =$ "enter_critical"
$\wedge \forall\, u \in Threads \setminus \{t\} : pc[u] \neq$ "enter_critical"

$Liveness \triangleq$

All threads must enter the critical section at least once

$\wedge \forall\, t \in Threads : \Diamond(pc[t] =$ "enter_critical"$)$
