MODULE *event_notifiers*

Formal specification of the *event_notifiers* datastructure.

It allows a single consumer to be put to sleep and being woken up by multiple producers so that the consumer is able to consume the incoming messages (steal requests).

This is a companion datastructure to a lock-free *MPSC* (Multi-Producer Single-Consumer) queue

It is a 2-step algorithm "prepareParking" + "park", that allows the potential sleeper to send a message in-between to its parent. The commit phase is aborted when any producer signals an incoming message (in the queue).

There should be no deadlock, *i.e.* an incoming message being signaled but the consumer stays sleeping as in the runtime the main thread may be the only one awake and wouldn't be able to awaken its children in that case.

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

CONSTANTS *NumThreads*, *ConsumerTID*, *MaxTasks*
ASSUME $NumThreads > 1$
ASSUME $ConsumerTID > 0$
ASSUME $ConsumerTID < NumThreads$

Threads are organized in an implicit binary tree
with Parent at N div 2 and child at $2N+1$ and $2N+2$ with root at 0

$MaxID \triangleq NumThreads - 1$
$ParentTID \triangleq ConsumerTID \div 2$
$producers \triangleq (0 \,..\, MaxID) \setminus \{ConsumerTID, ParentTID\}$

PlusCal options (-termination)
--algorithm *event_notifier*

variables

$phase =$ FALSE ;	a binary timestamp
$ticket =$ FALSE ;	a ticket to sleep in a phase
$signaled =$ FALSE ;	a flag for incoming condvar signal
$condVar =$ FALSE ;	Simulate a condition variable
$msgToParent =$ "None" ;	Simulate a worksharing request to the parent
$signaledTerminate =$ FALSE ;	Simulate a termination task
$tasks \in [0 \,..\, MaxID \to 0 \,..\, MaxTasks]$;	Tasks per Worker (up to *MaxTasks*)

Simulate a work-stealing runtime

---

macro *oneLessTask*(*pid*)**begin**
  $tasks[pid] := tasks[pid] - 1$;
end macro ;

procedure *mayRequestWork*(*pid*)
  Work or *Steal*
  begin *MaySteal*:

```
            if tasks[pid] > 0 then
                Work: oneLessTask(pid) ;
             else
                    Steal + execute right away so no increment
                Steal: call notify() ;
            end if ;
RET_WS: return ;
end procedure

procedure mayShareWork(pid)
      If child is "Waiting", send it some work
    begin MayShare:
              if tasks[pid] > 0 then
Share0:           if msgToParent = "Waiting" then
Share1:             call notify() ;                wakeup the child
Share2:             msgToParent := "None" ;        dequeue the child steal request
TaskShared:         oneLessTask(pid) ;
                  end if ;
              end if ;
RET_Share:    return ;
end procedure ;

 Event notifier being specified
```

```
procedure prepareParking()
    begin
        NotSignaled:  if ¬signaled then
        TakeTicket:     ticket := phase ;
                      end if ;
        RET_Int:      return ;
end procedure

procedure park()
    begin
        NotSignaled2:  if ¬signaled then
        StillValid:      if ticket = phase then
        Wait:              await condVar ;          next line is atomic
                           condVar := FALSE ;       we don't model the spurious wakeups here
                         end if ;                   but they are not a problem anyway
                         end if ;
        Reset:           signaled := FALSE ;
        RET_Park:        return ;
end procedure ;

procedure notify()
```

```
    variables prevState
    begin
        DontUndoOther: if signaled then
        EarlyExit:        return;
                        end if ;
        Notify:         signaled := TRUE;
        InvalidTicket:  phase := ¬phase ;
        Awaken:         condVar := TRUE;
        RET_Notify:     return;
end procedure ;
```

Simulate the runtime lifetime

```
process producer ∈ producers
    begin Coworkers:
        while tasks[self] > 0 do
          call mayRequestWork(self);
        end while ;
end process ;

process parent = ParentTID
    begin ParentWork:
        either   The order of work sharing and work stealing is arbitrary
            PMayRW0: call mayRequestWork(ParentTID);
            PMaySW0: call mayShareWork(ParentTID);
        or
            PMaySW1: call mayShareWork(ParentTID);
            PMayRW1: call mayRequestWork(ParentTID);
        end either ;
          But it will for sure tell the consumer to terminate at one point
    OutOfWork:
        if tasks = [x ∈ 0 .. MaxID ↦ 0] then
            Terminate0:  signaledTerminate := TRUE;
            WakeTerm1:   call notify();
            Drop:        msgToParent := "None" ;
         else
            StillWork: goto ParentWork;
        end if ;
end process ;

process consumer = ConsumerTID
    begin ConsumerWork:
            if tasks[ConsumerTID] > 0 then
                FoundWork: oneLessTask(ConsumerTID);
             else
```

```
                we signal our intent to sleep, tell our parent and then sleep
                Sleeping0: call prepareParking() ;
                Sleeping1: msgToParent := "Waiting" ;
                Sleeping2: call park() ;
            end if ;
    isEnding:
        if signaledTerminate then
            Received a termination task
            RecvTerminate: skip ;
        else
            goto ConsumerWork ;
        end if ;
end process ;

end algorithm ;
```

BEGIN TRANSLATION

Parameter *pid* of procedure *mayRequestWork* at line 51 col 26 changed to *pid_*

CONSTANT *defaultInitValue*

VARIABLES *phase*, *ticket*, *signaled*, *condVar*, *msgToParent*, *signaledTerminate*, *tasks*, *pc*, *stack*, *pid_*, *pid*, *prevState*

$vars \triangleq \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, pc, stack, pid\_, pid, prevState \rangle$

$ProcSet \triangleq (producers) \cup \{ParentTID\} \cup \{ConsumerTID\}$

$Init \triangleq$ Global variables
$\land\ phase = \text{FALSE}$
$\land\ ticket = \text{FALSE}$
$\land\ signaled = \text{FALSE}$
$\land\ condVar = \text{FALSE}$
$\land\ msgToParent = \text{"None"}$
$\land\ signaledTerminate = \text{FALSE}$
$\land\ tasks \in [0\,..\,MaxID \to 0\,..\,MaxTasks]$
Procedure *mayRequestWork*
$\land\ pid\_ = [self \in ProcSet \mapsto defaultInitValue]$
Procedure *mayShareWork*
$\land\ pid = [self \in ProcSet \mapsto defaultInitValue]$
Procedure notify
$\land\ prevState = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ stack = [self \in ProcSet \mapsto \langle\rangle]$
$\land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in producers \to \text{"Coworkers"}$
$\qquad\qquad\qquad\qquad\qquad\qquad \square\ \ self = ParentTID \to \text{"ParentWork"}$
$\qquad\qquad\qquad\qquad\qquad\qquad \square\ \ self = ConsumerTID \to \text{"ConsumerWork"}]$

$MaySteal(self) \triangleq \land\ pc[self] = \text{"MaySteal"}$

$\land$ IF $tasks[pid\_[self]] > 0$
THEN $\land\ pc' = [pc$ EXCEPT $![self] =$ "Work"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "Steal"$]$
$\land$ UNCHANGED $\langle phase,\ ticket,\ signaled,\ condVar,$
$msgToParent,\ signaledTerminate,\ tasks,\ stack,$
$pid\_,\ pid,\ prevState\rangle$

$Work(self) \triangleq \land\ pc[self] =$ "Work"
$\land\ tasks' = [tasks$ EXCEPT $![pid\_[self]] = tasks[pid\_[self]] - 1]$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "RET_WS"$]$
$\land$ UNCHANGED $\langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,$
$signaledTerminate,\ stack,\ pid\_,\ pid,\ prevState\rangle$

$Steal(self) \triangleq \land\ pc[self] =$ "Steal"
$\land\ stack' = [stack$ EXCEPT $![self] = \langle[procedure \mapsto$ "notify",
$pc \mapsto$ "RET_WS",
$prevState \mapsto prevState[self]]\rangle$
$\circ\ stack[self]]$
$\land\ prevState' = [prevState$ EXCEPT $![self] = defaultInitValue]$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "DontUndoOther"$]$
$\land$ UNCHANGED $\langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,$
$signaledTerminate,\ tasks,\ pid\_,\ pid\rangle$

$RET\_WS(self) \triangleq \land\ pc[self] =$ "RET_WS"
$\land\ pc' = [pc$ EXCEPT $![self] = Head(stack[self]).pc]$
$\land\ pid\_' = [pid\_$ EXCEPT $![self] = Head(stack[self]).pid\_]$
$\land\ stack' = [stack$ EXCEPT $![self] = Tail(stack[self])]$
$\land$ UNCHANGED $\langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,$
$signaledTerminate,\ tasks,\ pid,\ prevState\rangle$

$mayRequestWork(self) \triangleq MaySteal(self) \lor Work(self) \lor Steal(self)$
$\lor RET\_WS(self)$

$MayShare(self) \triangleq \land\ pc[self] =$ "MayShare"
$\land$ IF $tasks[pid[self]] > 0$
THEN $\land\ pc' = [pc$ EXCEPT $![self] =$ "Share0"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "RET_Share"$]$
$\land$ UNCHANGED $\langle phase,\ ticket,\ signaled,\ condVar,$
$msgToParent,\ signaledTerminate,\ tasks,\ stack,$
$pid\_,\ pid,\ prevState\rangle$

$Share0(self) \triangleq \land\ pc[self] =$ "Share0"
$\land$ IF $msgToParent =$ "Waiting"
THEN $\land\ pc' = [pc$ EXCEPT $![self] =$ "Share1"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "RET_Share"$]$
$\land$ UNCHANGED $\langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,$
$signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,$

$prevState\rangle$

$Share1(self) \triangleq \land\ pc[self] = \text{"Share1"}$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = \langle[procedure \mapsto \text{"notify"},$
$pc \mapsto \text{"Share2"},$
$prevState \mapsto prevState[self]]\rangle$
$\circ\ stack[self]]$
$\land\ prevState' = [prevState \text{ EXCEPT } ![self] = defaultInitValue]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"DontUndoOther"}]$
$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent,$
$signaledTerminate, tasks, pid\_, pid\rangle$

$Share2(self) \triangleq \land\ pc[self] = \text{"Share2"}$
$\land\ msgToParent' = \text{"None"}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"TaskShared"}]$
$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar,$
$signaledTerminate, tasks, stack, pid\_, pid,$
$prevState\rangle$

$TaskShared(self) \triangleq \land\ pc[self] = \text{"TaskShared"}$
$\land\ tasks' = [tasks \text{ EXCEPT } ![pid[self]] = tasks[pid[self]] - 1]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"RET\_Share"}]$
$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar,$
$msgToParent, signaledTerminate, stack,$
$pid\_, pid, prevState\rangle$

$RET\_Share(self) \triangleq \land\ pc[self] = \text{"RET\_Share"}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ pid' = [pid \text{ EXCEPT } ![self] = Head(stack[self]).pid]$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar,$
$msgToParent, signaledTerminate, tasks, pid\_,$
$prevState\rangle$

$mayShareWork(self) \triangleq MayShare(self) \lor Share0(self) \lor Share1(self)$
$\lor Share2(self) \lor TaskShared(self)$
$\lor RET\_Share(self)$

$NotSignaled(self) \triangleq \land\ pc[self] = \text{"NotSignaled"}$
$\land \text{ IF } \neg signaled$
$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"TakeTicket"}]$
$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"RET\_Int"}]$
$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar,$
$msgToParent, signaledTerminate, tasks,$
$stack, pid\_, pid, prevState\rangle$

$TakeTicket(self) \triangleq \land\ pc[self] = \text{"TakeTicket"}$

$\land\ ticket' = phase$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“RET\_Int”}]$
$\land \text{ UNCHANGED } \langle phase,\ signaled,\ condVar,\ msgToParent,$
$signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,$
$prevState\rangle$

$RET\_Int(self) \triangleq \land\ pc[self] = \text{“RET\_Int”}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,$
$signaledTerminate,\ tasks,\ pid\_,\ pid,$
$prevState\rangle$

$prepareParking(self) \triangleq NotSignaled(self) \lor TakeTicket(self)$
$\lor\ RET\_Int(self)$

$NotSignaled2(self) \triangleq \land\ pc[self] = \text{“NotSignaled2”}$
$\land \text{ IF } \neg signaled$
$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“StillValid”}]$
$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Reset”}]$
$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,$
$msgToParent,\ signaledTerminate,\ tasks,$
$stack,\ pid\_,\ pid,\ prevState\rangle$

$StillValid(self) \triangleq \land\ pc[self] = \text{“StillValid”}$
$\land \text{ IF } ticket = phase$
$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Wait”}]$
$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Reset”}]$
$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,$
$msgToParent,\ signaledTerminate,\ tasks,$
$stack,\ pid\_,\ pid,\ prevState\rangle$

$Wait(self) \triangleq \land\ pc[self] = \text{“Wait”}$
$\land\ condVar$
$\land\ condVar' = \text{FALSE}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Reset”}]$
$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ msgToParent,$
$signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,$
$prevState\rangle$

$Reset(self) \triangleq \land\ pc[self] = \text{“Reset”}$
$\land\ signaled' = \text{FALSE}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“RET\_Park”}]$
$\land \text{ UNCHANGED } \langle phase,\ ticket,\ condVar,\ msgToParent,$
$signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,$
$prevState\rangle$

$$RET\_Park(self) \triangleq \land pc[self] = \text{“RET\_Park”}$$
$$\land pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$$
$$\land stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$$
$$\land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, pid\_, pid, prevState\rangle$$

$$park(self) \triangleq NotSignaled2(self) \lor StillValid(self) \lor Wait(self) \lor Reset(self) \lor RET\_Park(self)$$

$$DontUndoOther(self) \triangleq \land pc[self] = \text{“DontUndoOther”}$$
$$\land \text{IF } signaled$$
$$\text{THEN } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“EarlyExit”}]$$
$$\text{ELSE } \land pc' = [pc \text{ EXCEPT } ![self] = \text{“Notify”}]$$
$$\land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, stack, pid\_, pid, prevState\rangle$$

$$EarlyExit(self) \triangleq \land pc[self] = \text{“EarlyExit”}$$
$$\land pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$$
$$\land prevState' = [prevState \text{ EXCEPT } ![self] = Head(stack[self]).prevState]$$
$$\land stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$$
$$\land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, pid\_, pid\rangle$$

$$Notify(self) \triangleq \land pc[self] = \text{“Notify”}$$
$$\land signaled' = \text{TRUE}$$
$$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“InvalidTicket”}]$$
$$\land \text{UNCHANGED } \langle phase, ticket, condVar, msgToParent, signaledTerminate, tasks, stack, pid\_, pid, prevState\rangle$$

$$InvalidTicket(self) \triangleq \land pc[self] = \text{“InvalidTicket”}$$
$$\land phase' = \neg phase$$
$$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“Awaken”}]$$
$$\land \text{UNCHANGED } \langle ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, stack, pid\_, pid, prevState\rangle$$

$$Awaken(self) \triangleq \land pc[self] = \text{“Awaken”}$$
$$\land condVar' = \text{TRUE}$$
$$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“RET\_Notify”}]$$
$$\land \text{UNCHANGED } \langle phase, ticket, signaled, msgToParent, signaledTerminate, tasks, stack, pid\_, pid, prevState\rangle$$

$$
\begin{array}{l}
RET\_Notify(self) \triangleq \land\ pc[self] = \text{“RET\_Notify”} \\
\qquad\qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc] \\
\qquad\qquad \land\ prevState' = [prevState \text{ EXCEPT } ![self] = Head(stack[self]).prevState] \\
\qquad\qquad \land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])] \\
\qquad\qquad \land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar, \\
\qquad\qquad\qquad\qquad msgToParent,\ signaledTerminate,\ tasks, \\
\qquad\qquad\qquad\qquad pid\_,\ pid\rangle
\end{array}
$$

$$
\begin{array}{l}
notify(self) \triangleq\ DontUndoOther(self) \lor EarlyExit(self) \lor Notify(self) \\
\qquad\qquad \lor\ InvalidTicket(self) \lor Awaken(self) \\
\qquad\qquad \lor\ RET\_Notify(self)
\end{array}
$$

$$
\begin{array}{l}
Coworkers(self) \triangleq\ \land\ pc[self] = \text{“Coworkers”} \\
\qquad\qquad \land \text{ IF } tasks[self] > 0 \\
\qquad\qquad\qquad \text{THEN } \land\ \land\ pid\_' = [pid\_ \text{ EXCEPT } ![self] = self] \\
\qquad\qquad\qquad\qquad\qquad \land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto \text{“mayRequestWork”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{“Coworkers”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pid\_ \mapsto pid\_[self]]\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[self]] \\
\qquad\qquad\qquad\qquad\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“MaySteal”}] \\
\qquad\qquad\qquad \text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}] \\
\qquad\qquad\qquad\qquad\ \land \text{ UNCHANGED } \langle stack,\ pid\_\rangle \\
\qquad\qquad \land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar, \\
\qquad\qquad\qquad\qquad msgToParent,\ signaledTerminate,\ tasks,\ pid, \\
\qquad\qquad\qquad\qquad prevState\rangle
\end{array}
$$

$$
producer(self) \triangleq Coworkers(self)
$$

$$
\begin{array}{l}
ParentWork \triangleq\ \land\ pc[ParentTID] = \text{“ParentWork”} \\
\qquad\qquad \land\ \lor\ \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“PMayRW0”}] \\
\qquad\qquad\qquad\ \lor\ \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“PMaySW1”}] \\
\qquad\qquad \land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent, \\
\qquad\qquad\qquad\qquad signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid, \\
\qquad\qquad\qquad\qquad prevState\rangle
\end{array}
$$

$$
\begin{array}{l}
PMayRW0 \triangleq\ \land\ pc[ParentTID] = \text{“PMayRW0”} \\
\qquad\qquad \land\ \land\ pid\_' = [pid\_ \text{ EXCEPT } ![ParentTID] = ParentTID] \\
\qquad\qquad\qquad\ \land\ stack' = [stack \text{ EXCEPT } ![ParentTID] = \langle [procedure \mapsto \text{“mayRequestWork”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{“PMaySW0”}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pid\_ \mapsto pid\_[ParentTID]]\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[ParentTID]] \\
\qquad\qquad \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“MaySteal”}] \\
\qquad\qquad \land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent, \\
\qquad\qquad\qquad\qquad signaledTerminate,\ tasks,\ pid,\ prevState\rangle
\end{array}
$$

$$
\begin{array}{l}
PMaySW0 \triangleq\ \land\ pc[ParentTID] = \text{“PMaySW0”} \\
\qquad\qquad \land\ \land\ pid' = [pid \text{ EXCEPT } ![ParentTID] = ParentTID]
\end{array}
$$

$$
\begin{array}{l}
\quad \land\ stack' = [stack \text{ EXCEPT } ![ParentTID] = \langle [procedure \mapsto \text{“mayShareWork”},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad pc \mapsto \text{“OutOfWork”},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad pid \mapsto pid[ParentTID]]\rangle\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \circ stack[ParentTID]]\\
\land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“MayShare”}]\\
\land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent,\\
\qquad\qquad signaledTerminate, tasks, pid\_, prevState\rangle
\end{array}
$$

$$
\begin{array}{rl}
PMaySW1 \triangleq & \land\ pc[ParentTID] = \text{“PMaySW1”}\\
& \land\ \land\ pid' = [pid \text{ EXCEPT } ![ParentTID] = ParentTID]\\
& \quad \land\ stack' = [stack \text{ EXCEPT } ![ParentTID] = \langle [procedure \mapsto \text{“mayShareWork”},\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad pc \mapsto \text{“PMayRW1”},\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad pid \mapsto pid[ParentTID]]\rangle\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \circ stack[ParentTID]]\\
& \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“MayShare”}]\\
& \land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent,\\
& \qquad\qquad signaledTerminate, tasks, pid\_, prevState\rangle
\end{array}
$$

$$
\begin{array}{rl}
PMayRW1 \triangleq & \land\ pc[ParentTID] = \text{“PMayRW1”}\\
& \land\ \land\ pid\_' = [pid\_ \text{ EXCEPT } ![ParentTID] = ParentTID]\\
& \quad \land\ stack' = [stack \text{ EXCEPT } ![ParentTID] = \langle [procedure \mapsto \text{“mayRequestWork”},\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad pc \mapsto \text{“OutOfWork”},\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad pid\_ \mapsto pid\_[ParentTID]]\rangle\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \circ stack[ParentTID]]\\
& \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“MaySteal”}]\\
& \land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent,\\
& \qquad\qquad signaledTerminate, tasks, pid, prevState\rangle
\end{array}
$$

$$
\begin{array}{rl}
OutOfWork \triangleq & \land\ pc[ParentTID] = \text{“OutOfWork”}\\
& \land \text{IF } tasks = [x \quad \in 0 \mathinner{..} MaxID \mapsto 0]\\
& \qquad \text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“Terminate0”}]\\
& \qquad \text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“StillWork”}]\\
& \land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent,\\
& \qquad\qquad signaledTerminate, tasks, stack, pid\_, pid,\\
& \qquad\qquad prevState\rangle
\end{array}
$$

$$
\begin{array}{rl}
Terminate0 \triangleq & \land\ pc[ParentTID] = \text{“Terminate0”}\\
& \land\ signaledTerminate' = \text{TRUE}\\
& \land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“WakeTerm1”}]\\
& \land \text{UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent,\\
& \qquad\qquad tasks, stack, pid\_, pid, prevState\rangle
\end{array}
$$

$$
\begin{array}{rl}
WakeTerm1 \triangleq & \land\ pc[ParentTID] = \text{“WakeTerm1”}\\
& \land\ stack' = [stack \text{ EXCEPT } ![ParentTID] = \langle [procedure \mapsto \text{“notify”},\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad pc \mapsto \text{“Drop”},\\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad prevState \mapsto prevState[ParentTID]]\rangle
\end{array}
$$

$$\circ\ stack[ParentTID]]$$
$$\land\ prevState' = [prevState \text{ EXCEPT } ![ParentTID] = defaultInitValue]$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“DontUndoOther”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,\ signaledTerminate,\ tasks,\ pid\_,\ pid\rangle$$

$$Drop \triangleq \land\ pc[ParentTID] = \text{“Drop”}$$
$$\land\ msgToParent' = \text{“None”}$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“Done”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,\ prevState\rangle$$

$$StillWork \triangleq \land\ pc[ParentTID] = \text{“StillWork”}$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ParentTID] = \text{“ParentWork”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,\ signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,\ prevState\rangle$$

$$parent \triangleq ParentWork \lor PMayRW0 \lor PMaySW0 \lor PMaySW1 \lor PMayRW1 \lor OutOfWork \lor Terminate0 \lor WakeTerm1 \lor Drop \lor StillWork$$

$$ConsumerWork \triangleq \land\ pc[ConsumerTID] = \text{“ConsumerWork”}$$
$$\land \text{ IF } tasks[ConsumerTID] > 0$$
$$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{“FoundWork”}]$$
$$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{“Sleeping0”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,\ signaledTerminate,\ tasks,\ stack,\ pid\_,\ pid,\ prevState\rangle$$

$$FoundWork \triangleq \land\ pc[ConsumerTID] = \text{“FoundWork”}$$
$$\land\ tasks' = [tasks \text{ EXCEPT } ![ConsumerTID] = tasks[ConsumerTID] - 1]$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{“isEnding”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,\ signaledTerminate,\ stack,\ pid\_,\ pid,\ prevState\rangle$$

$$Sleeping0 \triangleq \land\ pc[ConsumerTID] = \text{“Sleeping0”}$$
$$\land\ stack' = [stack \text{ EXCEPT } ![ConsumerTID] = \langle [procedure \mapsto \text{“prepareParking”},\ pc \mapsto \text{“Sleeping1”}] \rangle \circ stack[ConsumerTID]]$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{“NotSignaled”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,\ msgToParent,\ signaledTerminate,\ tasks,\ pid\_,\ pid,\ prevState\rangle$$

$$Sleeping1 \triangleq \land\ pc[ConsumerTID] = \text{“Sleeping1”}$$
$$\land\ msgToParent' = \text{“Waiting”}$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{“Sleeping2”}]$$
$$\land \text{ UNCHANGED } \langle phase,\ ticket,\ signaled,\ condVar,$$

$signaledTerminate, tasks, stack, pid\_, pid,$
$prevState\rangle$

$$Sleeping2 \triangleq \land\ pc[ConsumerTID] = \text{"Sleeping2"}$$
$$\land\ stack' = [stack \text{ EXCEPT } ![ConsumerTID] = \langle[procedure \mapsto \text{"park"}, pc \mapsto \text{"isEnding"}]\rangle \circ stack[ConsumerTID]]$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{"NotSignaled2"}]$$
$$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, pid\_, pid, prevState\rangle$$

$$isEnding \triangleq \land\ pc[ConsumerTID] = \text{"isEnding"}$$
$$\land \text{ IF } signaledTerminate$$
$$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{"RecvTerminate"}]$$
$$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{"ConsumerWork"}]$$
$$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, stack, pid\_, pid, prevState\rangle$$

$$RecvTerminate \triangleq \land\ pc[ConsumerTID] = \text{"RecvTerminate"}$$
$$\land \text{ TRUE}$$
$$\land\ pc' = [pc \text{ EXCEPT } ![ConsumerTID] = \text{"Done"}]$$
$$\land \text{ UNCHANGED } \langle phase, ticket, signaled, condVar, msgToParent, signaledTerminate, tasks, stack, pid\_, pid, prevState\rangle$$

$$consumer \triangleq ConsumerWork \lor FoundWork \lor Sleeping0 \lor Sleeping1 \lor Sleeping2 \lor isEnding \lor RecvTerminate$$

Allow infinite stuttering to prevent deadlock on termination.

$$Terminating \triangleq \land\ \forall\, self \in ProcSet : pc[self] = \text{"Done"}$$
$$\land \text{ UNCHANGED } vars$$

$$Next \triangleq parent \lor consumer$$
$$\lor\ (\exists\, self \in ProcSet : \lor\ mayRequestWork(self) \lor mayShareWork(self) \lor prepareParking(self) \lor park(self) \lor notify(self))$$
$$\lor\ (\exists\, self \in producers : producer(self))$$
$$\lor\ Terminating$$

$$Spec \triangleq \land\ Init \land \Box[Next]_{vars}$$
$$\land\ \forall\, self \in producers : \land\ \text{WF}_{vars}(producer(self)) \land \text{WF}_{vars}(mayRequestWork(self)) \land \text{WF}_{vars}(notify(self))$$
$$\land\ \land\ \text{WF}_{vars}(parent)$$
$$\land\ \text{WF}_{vars}(mayRequestWork(ParentTID))$$
$$\land\ \text{WF}_{vars}(mayShareWork(ParentTID))$$

$$
\begin{array}{ll}
 & \wedge \, \mathrm{WF}_{vars}(notify(ParentTID)) \\
\wedge & \wedge \, \mathrm{WF}_{vars}(consumer) \\
 & \wedge \, \mathrm{WF}_{vars}(prepareParking(ConsumerTID)) \\
 & \wedge \, \mathrm{WF}_{vars}(park(ConsumerTID))
\end{array}
$$

$$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$$

END TRANSLATION