MODULE *orchestrator*

EXTENDS *TLC*, *Naturals*, *Integers*, *Sequences*

CONSTANT *Workers*, *Manager*, *Clients*

$$\begin{aligned} \mathit{Messages} \triangleq\ & [\mathit{type} : \{\text{“task”}\},\ s : \mathit{Manager},\ r : \mathit{Workers}] \cup \\ & [\mathit{type} : \{\text{“working”},\ \text{“completed”},\ \text{“waiting”}\},\ s : \mathit{Workers},\ r : \mathit{Manager}] \cup \\ & [\mathit{type} : \{\text{“inprogress”},\ \text{“finished”}\},\ s : \mathit{Manager},\ r : \mathit{Clients}] \cup \\ & [\mathit{type} : \{\text{“doWork”}\},\ s : \mathit{Clients},\ r : \mathit{Manager}] \end{aligned}$$

$$\mathit{Actors} \triangleq \{\mathit{Workers} \cup \mathit{Manager} \cup \mathit{Clients}\}$$

--algorithm *orchestrator*

variables $\mathit{msgs} = \{\},\ \mathit{wState} = [w \in \mathit{Workers} \mapsto \text{“waiting”}],$
$\mathit{mState} = [m \in \mathit{Manager} \mapsto \text{“ready”}],$
$\mathit{cState} = [c \in \mathit{Clients} \mapsto \text{“idle”}],$
$\mathit{queues} = [q \in \mathit{Actors} \mapsto \langle\rangle]$ **;**

macro $\mathit{send}(\mathit{id},\ \mathit{msg})$**begin**
$\mathit{queues} := \mathit{Append}(\mathit{queues}[\mathit{id}],\ \mathit{msg})$ **;**
end macro ;

macro $\mathit{receive}(\mathit{msg})$**begin**
await $\mathit{Len}(\mathit{queues}[\mathit{self}] > 0)$ **;**
$\mathit{msg} := \mathit{Head}(\mathit{queues}[\mathit{self}])$ **;**
$\mathit{queues} := \mathit{Tail}(\mathit{queues}[\mathit{self}])$ **;**
end macro ;

process $\mathit{worker} \in \mathit{Workers}$
variable $\mathit{workQueue} = \langle\rangle$ **;**
begin
$\mathit{WaitForWork}$:
skip ;
$\mathit{PerformWork}$:
skip ;

end process ;

process $\mathit{client} \in \mathit{Clients}$
variable $\mathit{msg} = \langle\rangle$ **;**
begin
$\mathit{SendTaskToManager}$:
if $\mathit{msg} = \langle\rangle$ **then**
with $m \in \mathit{Manager}$ **do**
$\mathit{send}(\mathit{self},\ [\mathit{type} \mapsto \text{“doWork”},\ s \mapsto \mathit{self},\ r \mapsto m])$ **;**
end with ;

```
        end if ;
    ReceiveTaskFinish:
        with m ∈ Manager do
            if
                  ∧ msgs.type  = "finished"
                  ∧ mState[m] = "done"
             then
                receive(msg) ;
             else
                goto SendTaskToManager ;
            end if ;
        end with ;
end process ;

process manager ∈ Manager
begin
    NotifyClientOfCompleteJob:
       if msgs.type = "completed" then
            with c ∈ Clients do
                send(self, [type ↦ "finished", s ↦ self, r ↦ c]) ;
            end with ;
       end if ;
    ReceiveTaskFromClient:
        skip ;
    GiveTaskToWorker:
        skip ;

end process ;


end algorithm ;
```

BEGIN TRANSLATION ($chksum(pcal) =$ "901ddb8f" $\wedge\ chksum(tla) =$ "580405ba")

VARIABLES $msgs$, $wState$, $mState$, $cState$, $queues$, pc, $workQueue$, msg

$vars \triangleq \langle msgs,\ wState,\ mState,\ cState,\ queues,\ pc,\ workQueue,\ msg \rangle$

$ProcSet \triangleq (Workers) \cup (Clients) \cup (Manager)$

$Init \triangleq$ Global variables

$\wedge\ msgs = \{\}$

$\wedge\ wState = [w \in Workers \mapsto$ "waiting"$]$

$\wedge\ mState = [m \in Manager \mapsto$ "ready"$]$

$\wedge\ cState = [c \in Clients \mapsto$ "idle"$]$

$\wedge\ queues = [q \in Actors \mapsto \langle\rangle]$

Process worker

$\wedge\ workQueue = [self \in Workers \mapsto \langle\rangle]$

Process client

$\land\ msg = [self \in Clients \mapsto \langle\rangle]$
$\land\ pc = [self \in ProcSet \mapsto$ CASE $self \in Workers \to$ "WaitForWork"
$\square\ self \in Clients \to$ "SendTaskToManager"
$\square\ self \in Manager \to$ "NotifyClientOfCompleteJob"$]$

$WaitForWork(self) \triangleq$ $\land\ pc[self] =$ "WaitForWork"
$\land$ TRUE
$\land\ pc' = [pc$ EXCEPT $![self] =$ "PerformWork"$]$
$\land$ UNCHANGED $\langle msgs, wState, mState, cState, queues,$
$workQueue, msg\rangle$

$PerformWork(self) \triangleq$ $\land\ pc[self] =$ "PerformWork"
$\land$ TRUE
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Done"$]$
$\land$ UNCHANGED $\langle msgs, wState, mState, cState, queues,$
$workQueue, msg\rangle$

$worker(self) \triangleq WaitForWork(self) \lor PerformWork(self)$

$SendTaskToManager(self) \triangleq$ $\land\ pc[self] =$ "SendTaskToManager"
$\land$ IF $msg[self] = \langle\rangle$
THEN $\land \exists\, m \in Manager :$
$queues' = Append(queues[self], ([type \mapsto$ "doWork", $s \mapsto s$
ELSE $\land$ TRUE
$\land$ UNCHANGED $queues$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "ReceiveTaskFinish"$]$
$\land$ UNCHANGED $\langle msgs, wState, mState, cState,$
$workQueue, msg\rangle$

$ReceiveTaskFinish(self) \triangleq$ $\land\ pc[self] =$ "ReceiveTaskFinish"
$\land \exists\, m \in Manager :$
IF $\land\ msgs.type =$ "finished"
$\land\ mState[m] =$ "done"
THEN $\land\ Len(queues[self] > 0)$
$\land\ msg' = [msg$ EXCEPT $![self] = Head(queues[self])]$
$\land\ queues' = Tail(queues[self])$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Done"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "SendTaskToManager"$]$
$\land$ UNCHANGED $\langle queues, msg\rangle$
$\land$ UNCHANGED $\langle msgs, wState, mState, cState,$
$workQueue\rangle$

$client(self) \triangleq SendTaskToManager(self) \lor ReceiveTaskFinish(self)$

$NotifyClientOfCompleteJob(self) \triangleq$ $\land\ pc[self] =$ "NotifyClientOfCompleteJob"
$\land$ IF $msgs.type =$ "completed"
THEN $\land \exists\, c \in Clients :$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad queues' = Append(queues[self],\ ([type \mapsto \text{“finished”},\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{ELSE} \quad \land \text{TRUE}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \text{UNCHANGED } queues\\
\qquad\qquad\qquad\qquad\qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“ReceiveTaskFromClient”}]\\
\qquad\qquad\qquad\qquad\qquad \land \text{UNCHANGED } \langle msgs,\ wState,\ mState,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad cState,\ workQueue,\ msg\rangle
\end{array}
$$

$$
\begin{array}{l}
ReceiveTaskFromClient(self) \triangleq \land\ pc[self] = \text{“ReceiveTaskFromClient”}\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \text{TRUE}\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“GiveTaskToWorker”}]\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \text{UNCHANGED } \langle msgs,\ wState,\ mState,\ cState,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad queues,\ workQueue,\ msg\rangle
\end{array}
$$

$$
\begin{array}{l}
GiveTaskToWorker(self) \triangleq \land\ pc[self] = \text{“GiveTaskToWorker”}\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \text{TRUE}\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}]\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \text{UNCHANGED } \langle msgs,\ wState,\ mState,\ cState,\ queues,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad workQueue,\ msg\rangle
\end{array}
$$

$$
\begin{array}{l}
manager(self) \triangleq\ NotifyClientOfCompleteJob(self)\\
\qquad\qquad\qquad\qquad\quad \lor\ ReceiveTaskFromClient(self)\\
\qquad\qquad\qquad\qquad\quad \lor\ GiveTaskToWorker(self)
\end{array}
$$

Allow infinite stuttering to prevent deadlock on termination.

$$
\begin{array}{l}
Terminating \triangleq \land \forall\, self \in ProcSet : pc[self] = \text{“Done”}\\
\qquad\qquad\qquad\quad \land \text{UNCHANGED } vars
\end{array}
$$

$$
\begin{array}{l}
Next \triangleq\ (\exists\, self \in Workers : worker(self))\\
\qquad\qquad \lor\ (\exists\, self \in Clients\ \ : client(self))\\
\qquad\qquad \lor\ (\exists\, self \in Manager : manager(self))\\
\qquad\qquad \lor\ Terminating
\end{array}
$$

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

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

END TRANSLATION

$$
\begin{array}{l}
TypeOK \triangleq\\
\qquad \land\ wState \in [Workers \to \{\text{“waiting”},\ \text{“working”}\}]\\
\qquad \land\ mState \in [Manager \to \{\text{“ready”},\ \text{“busy”},\ \text{“jobComplete”}\}]\\
\qquad \land\ cState\ \in [Clients\ \ \to \{\text{“assignTask”},\ \text{“idle”}\}]\\
\qquad \land\ msgs \subseteq Messages
\end{array}
$$

* Modification History

\ * Last modified *Mon Mar* 11 10:40:22 *CET* 2024 by lee
\ * Created *Fri Mar* 08 22:22:11 *CET* 2024 by lee