```
────────────────── MODULE lww_register ──────────────────
EXTENDS Integers, Sequences, TLC
CONSTANTS N

Procs ≜ 1 .. N

INITIAL ≜ [t ↦ 0, val ↦ “”]   initial value [t ↦ 0, val ↦ “”]

--algorithm lww_register
variables
  msgs = [j ∈ Procs ↦ INITIAL];   to send the messages

define
  _Compare(p1, p2) ≜ p1.t ≤ p2.t   return TRUE if timestamp_1 ≤ timestamp_2

  _Merge(p1, p2) ≜ IF _Compare(p1, p2) THEN p2 ELSE p1
end define ;

  assign a value and timestamp into local payload
macro _Assign(v)begin
  payload := [t ↦ JavaTime, val ↦ v];
  print ToString(self) ∘ “ assigned ” ∘ ToString(payload);
end macro ;

  send the payload 'p' to a random proc
macro _Send(p)begin
  if payload ≠ INITIAL then
    with j ∈ Procs \ {self} do
      msgs[j] := payload;
      print ToString(self) ∘ “ sent ” ∘ ToString(msgs[j]) ∘ “ to ” ∘ ToString(j);
    end with ;
  end if ;
end macro ;

  receive the payload
macro _Receive()begin
  if msgs[self] ≠ INITIAL then
    print ToString(self) ∘ “ received ” ∘ ToString(msgs[self]);
    if payload = INITIAL then   payload is empty, just receive the message
      payload := msgs[self];
     else   merge the payload and received message
      payload := _Merge(payload, msgs[self]);
    end if ;
    msgs[self] := INITIAL;
    print ToString(self) ∘ “ merged ” ∘ ToString(payload);
  end if ;
end macro ;
```

```
fair process Register ∈ Procs
variables
  i = 0,   count iterations
  payload = INITIAL ;   local payload
begin Main:
  while i < N do
    either Assign:
      _Assign(self) ;
    or Send:
      _Send(payload) ;
    or Receive:
      _Receive() ;
    end either  ;
    Loop:
      i := i + 1 ;
  end while  ;
end process  ;

end algorithm   ;
 BEGIN TRANSLATION
VARIABLES msgs, pc

 define statement
_Compare(p1, p2) ≜ p1.t ≤ p2.t

_Merge(p1, p2) ≜ IF _Compare(p1, p2) THEN p1 ELSE  p2

VARIABLES i, payload

vars ≜ ⟨msgs, pc, i, payload⟩

ProcSet ≜ (Procs)

Init ≜  Global variables
          ∧ msgs = [j ∈ Procs ↦ INITIAL]
          Process Register
          ∧ i = [self ∈ Procs ↦ 0]
          ∧ payload = [self ∈ Procs ↦ INITIAL]
          ∧ pc = [self ∈ ProcSet ↦ "Main"]

Main(self) ≜  ∧ pc[self] = "Main"
              ∧ IF i[self] < N
                   THEN  ∧ ∨ ∧ pc' = [pc EXCEPT ![self] = "Assign"]
                           ∨ ∧ pc' = [pc EXCEPT ![self] = "Send"]
                           ∨ ∧ pc' = [pc EXCEPT ![self] = "Receive"]
                   ELSE  ∧ pc' = [pc EXCEPT ![self] = "Done"]
              ∧ UNCHANGED ⟨msgs, i, payload⟩
```

```
$Loop(self) \triangleq$ $\land pc[self] = $ “Loop”
                       $\land i' = [i$ EXCEPT $![self] = i[self] + 1]$
                       $\land pc' = [pc$ EXCEPT $![self] = $ “Main”$]$
                       $\land$ UNCHANGED $\langle msgs, payload \rangle$

$Assign(self) \triangleq$ $\land pc[self] = $ “Assign”
                         $\land payload' = [payload$ EXCEPT $![self] = [t \mapsto JavaTime, val \mapsto self]]$
                         $\land PrintT(ToString(self) \circ$ “ assigned ” $\circ ToString(payload'[self]))$
                         $\land pc' = [pc$ EXCEPT $![self] = $ “Loop”$]$
                         $\land$ UNCHANGED $\langle msgs, i \rangle$

$Send(self) \triangleq$ $\land pc[self] = $ “Send”
                       $\land$ IF $payload[self] \neq INITIAL$
                             THEN $\land \exists j \in Procs \setminus \{self\}$ :
                                       $\land msgs' = [msgs$ EXCEPT $![j] = payload[self]]$
                                       $\land PrintT(ToString(self) \circ$ “ sent ” $\circ ToString(msgs'[j]) \circ$ “ to ” $\circ ToString(j))$
                             ELSE  $\land$ TRUE
                                   $\land msgs' = msgs$
                       $\land pc' = [pc$ EXCEPT $![self] = $ “Loop”$]$
                       $\land$ UNCHANGED $\langle i, payload \rangle$

$Receive(self) \triangleq$ $\land pc[self] = $ “Receive”
                          $\land$ IF $msgs[self] \neq INITIAL$
                                THEN $\land PrintT(ToString(self) \circ$ “ received ” $\circ ToString(msgs[self]))$
                                     $\land$ IF $payload[self] = INITIAL$
                                           THEN $\land payload' = [payload$ EXCEPT $![self] = msgs[self]]$
                                           ELSE $\land payload' = [payload$ EXCEPT $![self] = \_Merge(payload[self], msgs$
                                     $\land msgs' = [msgs$ EXCEPT $![self] = INITIAL]$
                                     $\land PrintT(ToString(self) \circ$ “ merged ” $\circ ToString(payload'[self]))$
                                ELSE $\land$ TRUE
                                     $\land$ UNCHANGED $\langle msgs, payload \rangle$
                          $\land pc' = [pc$ EXCEPT $![self] = $ “Loop”$]$
                          $\land i' = i$

$Register(self) \triangleq$ $Main(self) \lor Loop(self) \lor Assign(self) \lor Send(self)$
                              $\lor Receive(self)$

$Next \triangleq$ $(\exists self \in Procs : Register(self))$
            $\lor$  Disjunct to prevent deadlock on termination
              $((\forall self \in ProcSet : pc[self] = $ “Done”$) \land$ UNCHANGED $vars)$

$Spec \triangleq$ $\land Init \land \Box[Next]_{vars}$
             $\land \forall self \in Procs : \mathrm{WF}_{vars}(Register(self))$

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

 END TRANSLATION
```

* Modification History
* Last modified *Fri Dec* 14 18:21:25 *PST* 2018 by *ocosta*
* Created *Fri Dec* 14 16:18:39 *PST* 2018 by *ocosta*