```
┌──────────────── 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]) ∘ “; current payload ” ∘ ToString(payload) ;
    payload := _Merge(payload, msgs[self]) ;
    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 p2 ELSE p1

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) ≜ ∧ pc[self] = "Loop"
             ∧ i′ = [i EXCEPT ![self] = i[self] + 1]
             ∧ pc′ = [pc EXCEPT ![self] = "Main"]
             ∧ UNCHANGED ⟨msgs, payload⟩
```

```
Assign(self) ≜  ∧ pc[self]  = “Assign”
                ∧ payload′ = [payload EXCEPT ![self] = [t ↦ JavaTime, val ↦ self]]
                ∧ PrintT(ToString(self) ∘ “ assigned ” ∘ ToString(payload′[self]))
                ∧ pc′ = [pc EXCEPT ![self] = “Loop”]
                ∧ UNCHANGED ⟨msgs, i⟩

Send(self) ≜  ∧ pc[self] = “Send”
              ∧ IF payload[self] ≠ INITIAL
                    THEN ∧ ∃ j ∈ Procs \ {self} :
                                ∧ msgs′ = [msgs EXCEPT ![j] = payload[self]]
                                ∧ PrintT(ToString(self) ∘ “ sent ” ∘ ToString(msgs′[j]) ∘ “ to ” ∘ ToString(j))
                    ELSE ∧ TRUE
                         ∧ msgs′ = msgs
              ∧ pc′ = [pc EXCEPT ![self] = “Loop”]
              ∧ UNCHANGED ⟨i, payload⟩

Receive(self) ≜  ∧ pc[self] = “Receive”
                 ∧ IF msgs[self] ≠ INITIAL
                       THEN ∧ PrintT(ToString(self) ∘ “ received ” ∘ ToString(msgs[self]) ∘ “; current payloa
                            ∧ payload′ = [payload EXCEPT ![self] = _Merge(payload[self], msgs[self])]
                            ∧ msgs′ = [msgs EXCEPT ![self] = INITIAL]
                            ∧ PrintT(ToString(self) ∘ “ merged ” ∘ ToString(payload′[self]))
                       ELSE ∧ TRUE
                            ∧ UNCHANGED ⟨msgs, payload⟩
                 ∧ pc′ = [pc EXCEPT ![self] = “Loop”]
                 ∧ i′ = i

Register(self) ≜  Main(self) ∨ Loop(self) ∨ Assign(self) ∨ Send(self)
                      ∨ Receive(self)

Next ≜  (∃ self ∈ Procs : Register(self))
           ∨  Disjunct to prevent deadlock on termination
             ((∀ self ∈ ProcSet : pc[self] = “Done”) ∧ UNCHANGED vars)

Spec ≜  ∧ Init ∧ □[Next]_vars
        ∧ ∀ self ∈ Procs : WF_vars(Register(self))

Termination ≜  ◇(∀ self ∈ ProcSet : pc[self] = “Done”)

 END TRANSLATION

\* Modification History
\* Last modified Fri Dec 14 19:00:11 PST 2018 by ocosta
\* Created Fri Dec 14 16:18:39 PST 2018 by ocosta
```