A lease can be thought as a lock with a timeout. When a process holds a loc, it can access some resources. In case the lock owner dies, to avoid deadlock the lock has a fixed timeout time. If the lease is not renewed before the lease is expired (or explicitly released), a new process can attempt to acquire the lease. Sometimes it get implemented like this:

```
1. while TRUE{
2.    if(leaseAlmostOver()){
3.       renewLease();
4.    }
5.    doOperationOnResource()
6.}
```

This code introduces a Time Of Check/ Time Of Use Bug ($TOCTOU$) on line 5. After the lease is renewed, the lease owner could potentially be put to sleep for an amount of time greater than the reaming lease time. After waking up, the process would start the operation assuming he still holds the lease which instead has since expired while the lease was since acquired by another process.

One solution would be to use atomic commit, and check at the end if the lease is still valid.

This specification models the above algorithm to expose the concurrency bug.

The available states are:
* *WaitingForLease* - Initial. When the process doesn't own a lock.
* *RenewedLease* - maps to line 3.
* *DoingOperation* - maps to line 4.
* *Sleep* - maps to the concept of sleep and expired lease.

The invariant offered by a lease is similar to locks: only a single process can access the critical section at a time. To verify this, we assume that a process will not get a lease if we know it's already holded by another process. When the owner goes to the sleep state, we assume it goes to sleep for enough time that the lease is expired. This allows us to avoid dealing with the time variable.

References:
* Designing Data Intensive Systems, chapter 8 section "Process Pauses"
* https://*martin.kleppmann.com*/2016/02/08/how-to-do-distributed-*locking.html*

MODULE *lease*

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

CONSTANTS *Workers*

$StatesSleep \triangleq$ "Sleep"
$StatesWaitingLease \triangleq$ "WaitingLease"
$StatesDoingOperation \triangleq$ "DoingOperation"
$StatesRenewedLease \triangleq$ "RenewedLease"
$States \triangleq \{StatesSleep, StatesWaitingLease, StatesDoingOperation, StatesRenewedLease\}$

$baitinv \triangleq TLCGet(\text{"}level\text{"}) < 16$

--algorithm *leaseVerifier* **{**
variables $states = [x \in Workers \mapsto StatesWaitingLease]$ **;**
define {
$OnlyOneLeader \triangleq Cardinality(\{w \in \text{DOMAIN } (states) : states[w] = StatesDoingOperation\}) \leq 1$

$TypeOk \triangleq \forall w \in Workers : states[w] \in States$
$Inv \triangleq \land TypeOk$
$\land OnlyOneLeader$
$\land baitinv$

}

macro *stateIs*(s) **{**
$states[self] = s$
}

process ($w \in Workers$) **{**
W:
while (TRUE) **{**
either {
If lease is expired, renew
await $stateIs(StatesWaitingLease)$
await $\neg\exists\, w \in Workers : states[w] = StatesDoingOperation \lor states[w] = StatesRenewedLe$
$states[self] := StatesRenewedLease$ **;**
} or {
Leader goes to sleep before start operation
await $stateIs(StatesRenewedLease)$ **;**
this state is like saying that lease is expired.
$states[self] := StatesSleep$ **;**
} or {
Leader start the operation
await $stateIs(StatesRenewedLease) \lor stateIs(StatesSleep)$ **;**
$states[self] := StatesDoingOperation$ **;**
} or {
Leader has completed the operation, no goes through the renew lease phase.
await $stateIs(StatesDoingOperation)$ **;**
$states[self] := StatesWaitingLease$ **;**
}
}
}
}
BEGIN TRANSLATION ($chksum(pcal) =$ "92808600" $\land\ chksum(tla) =$ "8f5050cf")
VARIABLE *states*

define statement
$OnlyOneLeader \triangleq Cardinality(\{w \in \text{DOMAIN } (states) : states[w] = StatesDoingOperation\}) \leq 1$

$TypeOk \triangleq \forall w \in Workers : states[w] \in States$
$Inv \triangleq \land TypeOk$
$\land OnlyOneLeader$

$vars \triangleq \langle states \rangle$

$ProcSet \triangleq (Workers)$

$Init \triangleq$ Global variables
$\land\ states = [x \in Workers \mapsto StatesWaitingLease]$

$w(self) \triangleq \lor \land states[self] = StatesWaitingLease$
$\land \neg\exists\, w \in Workers : states[w] = StatesDoingOperation \lor states[w] = StatesRenewedLease$
$\land\ states' = [states \text{ EXCEPT } ![self] = StatesRenewedLease]$
$\lor \land states[self] = StatesRenewedLease$
$\land\ states' = [states \text{ EXCEPT } ![self] = StatesSleep]$
$\lor \land states[self] = StatesRenewedLease \lor states[self] = StatesSleep$
$\land\ states' = [states \text{ EXCEPT } ![self] = StatesDoingOperation]$
$\lor \land states[self] = StatesDoingOperation$
$\land\ states' = [states \text{ EXCEPT } ![self] = StatesWaitingLease]$

$Next \triangleq (\exists\, self \in Workers : w(self))$

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

END TRANSLATION