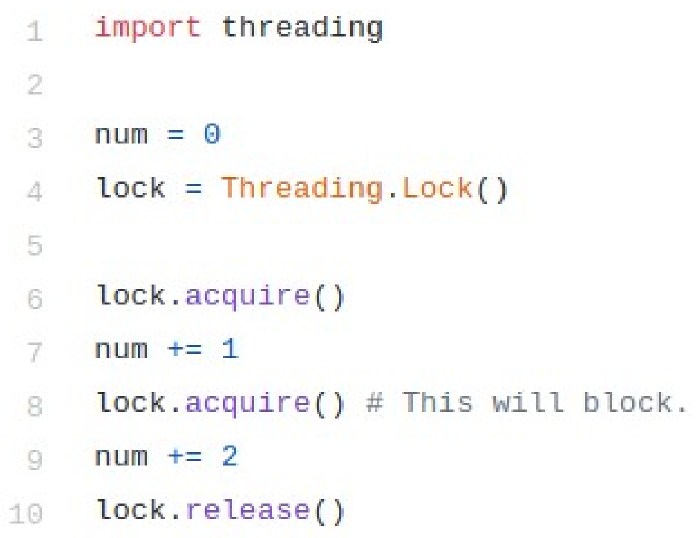
Aula 6 – Sincronismo 2

Exclusão Mútua

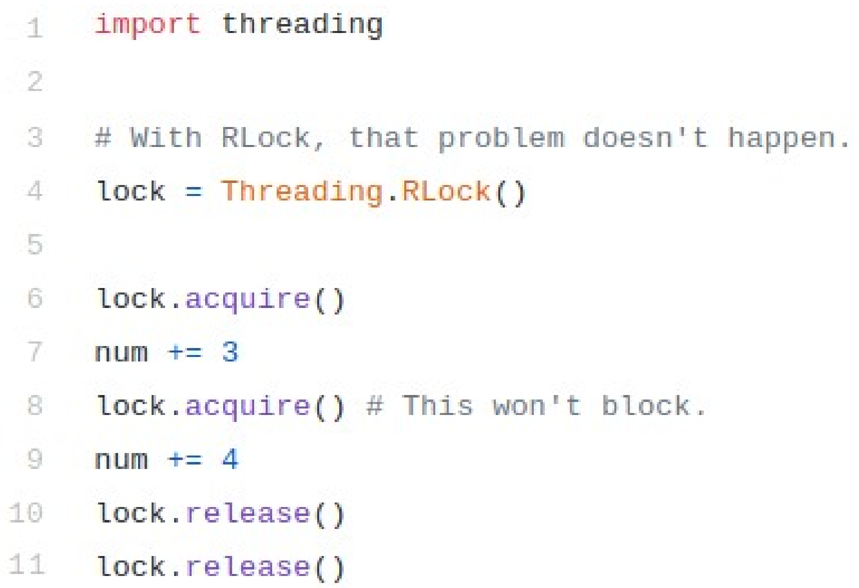
- Variável de impedimento
- Locks e Rlocks
  - Lock → qualquer thread que tentar adquiri-lo irá bloquear, mesmo se o mesmo segmento em si já estiver segurando o bloqueio.
  - Nesses casos, RLock (bloqueio de reentrada) é usado.

Exclusão Mutua

- Lock X Rlock



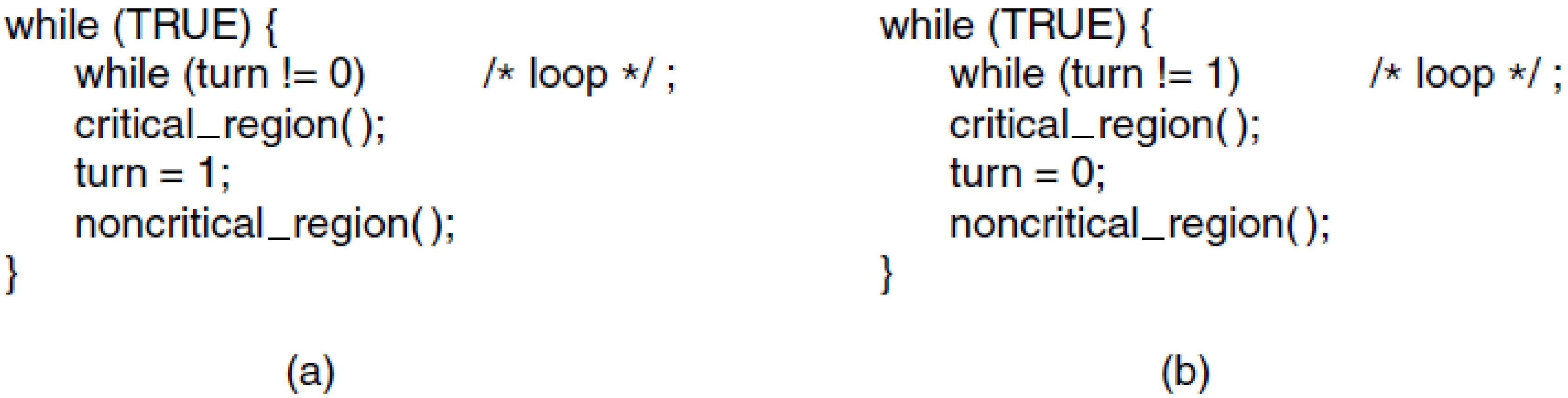
```python
import threading

num = 0
lock = Threading.Lock()

lock.acquire()
num += 1
lock.acquire() # This will block.
num += 2
lock.release()
```



```python
import threading

# With RLock, that problem doesn't happen.
lock = Threading.RLock()

lock.acquire()
num += 3
lock.acquire() # This won't block.
num += 4
lock.release()
lock.release()
```

Exclusão Mútua

- Com alternância estrita



```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

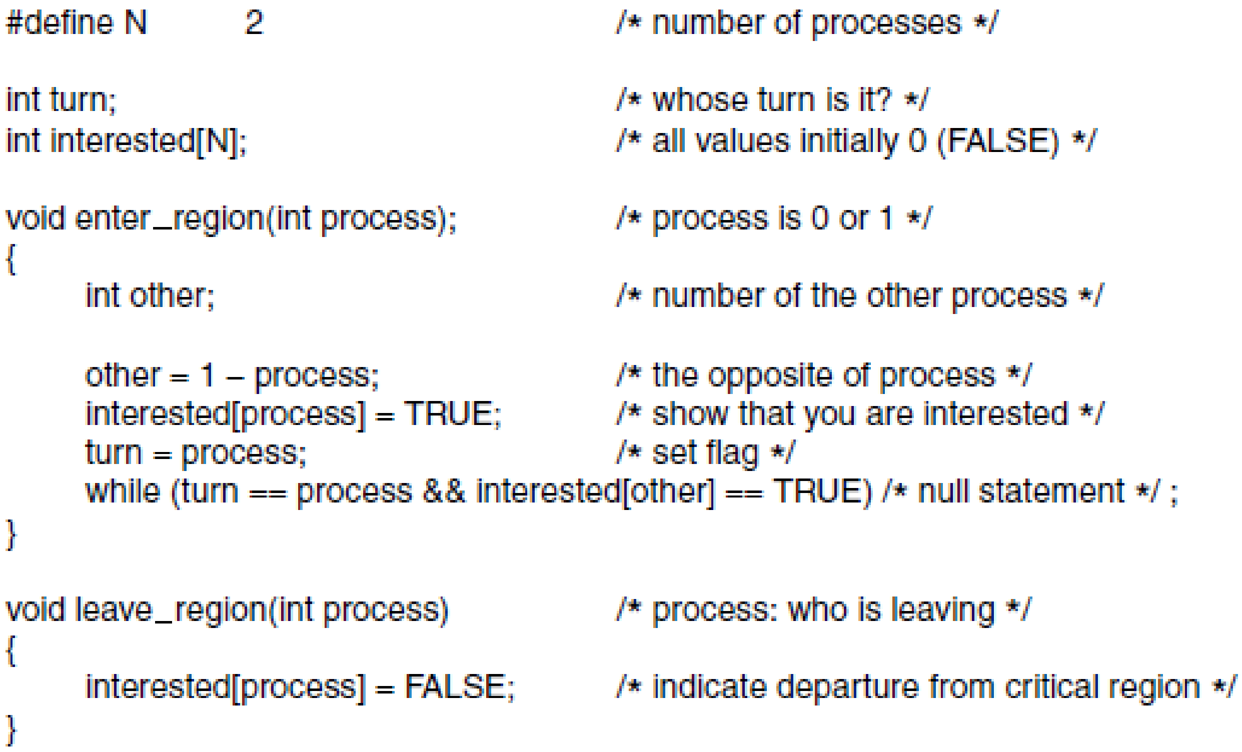
(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Exclusão Mútua

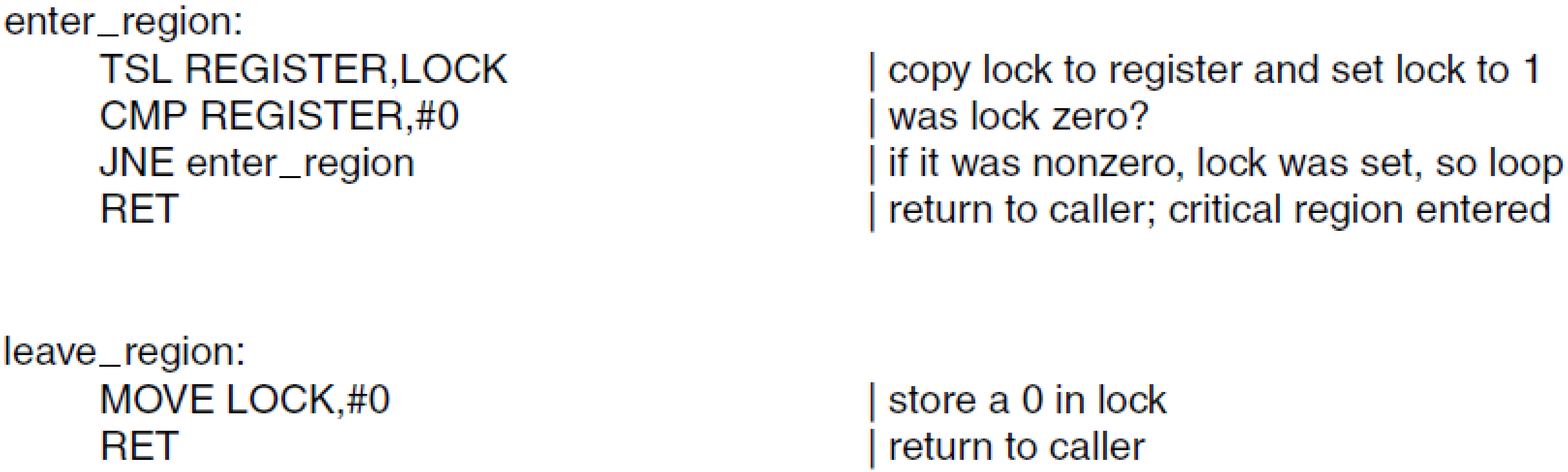
- Solução de Peterson



```
#define N     2                        /* number of processes */

int turn;                              /* whose turn is it? */
int interested[N];                     /* all values initially 0 (FALSE) */

void enter_region(int process);        /* process is 0 or 1 */
{
     int other;                        /* number of the other process */

     other = 1 – process;              /* the opposite of process */
     interested[process] = TRUE;       /* show that you are interested */
     turn = process;                   /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)         /* process: who is leaving */
{
     interested[process] = FALSE;      /* indicate departure from critical region */
}
```

Exclusão Mútua

- Instrução TSL
  - Auxílio do hardware



```
enter_region:
        TSL REGISTER,LOCK                      | copy lock to register and set lock to 1
        CMP REGISTER,#0                        | was lock zero?
        JNE enter_region                       | if it was nonzero, lock was set, so loop
        RET                                    | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0                           | store a 0 in lock
        RET                                    | return to caller
```

Exclusão Mutua

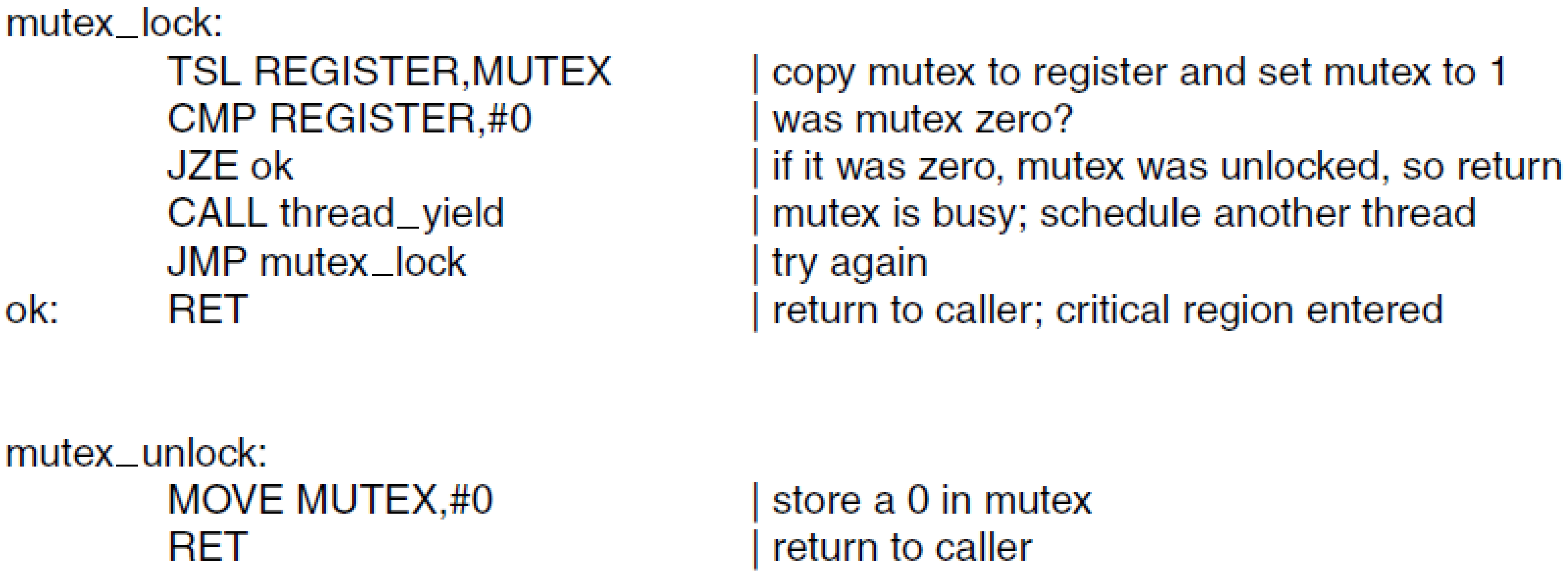
- Soluções anteriores
  - com espera ocupada
- Soluções ideais
  - Com sleep e wakeup

Semáforos

- E.W. Dijkstra (1965) → variável inteira
  - Operação **down**: se maior que zero, decrementa
  - Operação **up**: se menor que o máximo, incrementa
- Up e down são generalizações de acquire e release
- Ações atômicas

Mutex

- Mutex → Mutual Exclusion
  - Semáforo binário



```
mutex_lock:
          TSL REGISTER,MUTEX        | copy mutex to register and set mutex to 1
          CMP REGISTER,#0           | was mutex zero?
          JZE ok                    | if it was zero, mutex was unlocked, so return
          CALL thread_yield         | mutex is busy; schedule another thread
          JMP mutex_lock            | try again
ok:       RET                       | return to caller; critical region entered


mutex_unlock:
          MOVE MUTEX,#0             | store a 0 in mutex
          RET                       | return to caller
```