Assignment 17.

State Peterson's algorithm. List its properties and prove each of them.

SHARED: inter[0] = inter[1] = false, turn_to_wait = doesn’t matter.

Code for process 0.

```
inter[0] = True
turn_to_wait = 0
while ((inter[1] == True) AND (turn_to_wait == 0)) { // busy waiting }
<CS>
inter[0] = False
```

Code for process 1.

```
inter[1] = True
turn_to_wait = 1
while ((inter[0] == True) AND (turn_to_wait == 1)) { // busy waiting }
<CS>
inter[1] = False
```

Mutual exclusion:

Assume for the sake of the contradiction that the Peterson's algorithm doesn’t provide mutual exclusion. Then, the only way to get such a situation is either:

● turn_to_wait = 0 and turn_to_wait = 1: it’s trivial that’s this isn’t possible.

● turn_to_wait = 0 and inter[0] = false: Assuming turn_to_wait = 0, we show that inter must be true. Indeed, since turn_to_wait = 0, inter[0] is true by the algorithm rules (if not then the process 0 just finish the CS and then if he is interested once again he must state inter[0] = true). Thus, this case cannot happen in anytime on the algorithm.

● turn_to_wait = 1 and inter[1] = false: similar argument than turn_to_wait = 0 and inter[0] = false.

● inter[0] = false and inter[1] = false: Trivially, the value of inter[0] is changed only in the process 1 and the same with inter[1] in the process 0. Then, every time that any process either 1 or 2 is interested to enter in the CS, he turn the value of the inter of the other process to true. Then there is no way to get inter[0] = false and inter[1] = false.

No one of those situation can happen, which confirm the claim. ■

Deadlock free:

Since turn_to_wait is either 0 or 1 but can't be both at the same time, it's trivial that Peterson's provide deadlock freedom.

∎

Starvation free:

Assume for the sake of the contradiction that the Peterson's algorithm doesn't provide starvation freedom. Then, the only way to get such a situation is either:

- inter[1] = true and turn_to_wait = 1 forever: If inter[1] = true, we can affirm that process 1 is interested. Then both of processes interested, by the deadlock free property we have than one of the two enter in the CS. Assuming that process 1 enter in the CS (else we do not have inter[1] = true and turn_to_wait = 1), when process 1 enter in the exit code, he turn inter[1] to false. Now process 0 is able to enter in the process, if he gets running time. If he doesn't, and process 1 is once again interested by enter in the CS, we will have turn_to_wait = 1, and then, only process 0 can enter in the CS. Contradicting the claim that inter[1] = true and turn_to_wait = 1 forever.
- inter[0] = true and turn_to_wait = 0 forever: with a similar argument than above, we conclude the claim.

No one of those situation can happen, which confirm the claim. ∎