HW 1

rt25884
trm2796

3. **(25 points)** Show that any of the following modifications to Peterson's algorithm makes it incorrect:

   a) A process in Peterson's algorithm sets the *turn* variable to itself instead of setting it to the other process. The remaining algorithm stays the same.

   b) A process sets the *turn* variable before setting the *wantCS* variable.

a)

If a process in Pertsons' algorithm sets the turn variable to itself instead of the other process, there isn't always mutual exclusion

The original algorithm ensures safety by P_0 and P_1 not being in the cs at the same time.'

Take this altered algorithm:

P_0 sets wantCS[0] = true

P_0 sets turn to 0

P_0 enters CS because wantCS[1] = false

P_1 sets wantCS[1] = true

P_1 sets turn to 1

P_1 enters CS because turn = 1

Both enter CS at same time which the original algorithm prevents.

b)

Take the case when P_0 and P_1 are both requesting the CS in this altered algorithm

P_0 sets turn to 1

P_1 sets turn to 0

P_1 sets wantCS[1] = true

P_1 enters critical section because wantCS[0] is false

P_0 sets wantCS[0] = true

P_0 enters critical section because turn is not 1

Both enter CS at same time which the original algorithm prevents.

4. **(25 points)** Prove that Peterson's algorithm is free from starvation.

W.L.O.G. starvation occurs when process P_0 is attempting to access the critical section but it cannot because process P_1 is repeatedly executing.

If P_0 is attempting to access the critical section it is being denied that means that it is perpetually in the while loop waiting.

It is important to note that upon exiting the CS P_1 sets wantCS[1] = false. Because of this P_0 can now exit its while loop.

Now it needs to be assured that P_1 does not re-enter the CS before P_0 thereby locking P_0 out again.

- Prior to attempting to re-enter:
  - P_1 sets wantCS[1] = true
  - P_1 sets turn = 0
- Since turn is 0 and wantCS[0] still remains true from P_0's request to enter the CS
  - The while loop is true for P_1 [because wantCS[1] and turn == 0 are true]
  - Since turn is now 0, P_0 can now enter the critical section

The other case to consider is when P_0 wants to access the CS and P_1 does not
We know that P_0 has set wantCS[0] = true
And since P_1 does not request to enter the CS we know that wantCS[1] = false

- Upon exiting the CS, P_1 sets its wantCS[1] = false
- And if it has not requested to enter it remains false

Since wantCS[1] is false, P_0 will not remain in the while loop and will then enter the critical section

Therefore there is no starvation.