Sarah Dickerson
Talat Oguz

3)

a)
```
requestCS (int i)
    int j = 1-i
    wantCS[i] = true
    turn = i
    while(wantCS[j] and turn == j)
```

A process trying to access the CS sets the turn to itself and therefore can access the CS regardless of the wantCS value of the other process. this would violate the safety property as a process can enter the CS w/o the need of the other releasing it first

b)
```
requestCS (int i)
    int j = 1-i
    turn = j
    wantCS[i] = true
    while(wantCS[j] and turn == j)
```

Suppose WLOG process 0 calls requestCS, setting j=1 and turn = 1 then process 1 sets j=∅ and wantCS[1] true. process 1 now enters the CS by leaving the while loop b/c wantCS[0] is still false

Now process 0 sets wantCS[0] to true and enters the CS since turn = ∅ ≠ j = 1 so the while loop condition is not satisfied.

However, this violates the mutual exclusion property as now both processes are in the CS at the same time.

4) Free from starvation: if a process is trying to enter CS, it eventually will

Suppose there are two processes, P_0 and P_1, and WLOG P_0 is trying to enter the CS. then there are three possible cases for P_1.

<u>case 1:</u> P_1 is inside the CS

wantCS[1] is or will be false, meaning it will break P_0's busy wait and let P_0 enter the CS

<u>case 2:</u> P_1 is not in the CS and does not wantCS

wantCS[1] is or will be false, meaning it will break P_0's busy wait and let P_0 enter the CS

<u>case 3:</u> P_1 and P_0 both outside CS, both want CS

Depending on which process sets the turn variable last, since the turn variable is a condition of the busy wait, the other will enter the CS first.

As we can see from above, there are no cases where a process trying to enter the CS is starved from entering, thus Peterson's Algorithm is free from starvation