3.

TACC UserId: ngilani, sahmed5
Numan Gilani: ng22457
Shehryar Ahmed: sa43897

4.

a)

Let's assume that processes set the turn variable to themselves. Without loss of generality, assume that P0 and P1 begin their entry protocols at around the same time in the following fashion. First, P1 sets wantCS[1] to true. Before it proceeds, P0 sets wantCS[0] to true and sets the turn variable to 0. P0 then passes the while loop condition because the turn variable does not equal 1. Therefore, P0 enters the critical section. Then, P1 resumes and sets the turn variable to 1. It also passes the while loop condition because the turn variable does not equal to 0. Therefore, P1 also enters into the critical section. This violates mutual exclusion.

b)

This configuration is never a problem until both processes attempt to enter their critical sections at the same time. In that case, this configuration causes a violation of the safety property. That is, both processes are in the critical state at the same time. Without loss of generality, this situation arises when P1 sets the turn variable to 0 and before it changes wantCS[1], P0 sets the turn variable to 1 and wantCS[0] to true. In this situation, P0 passes the while loop condition because P1 has not set wantCS[1] to true yet. Therefore, P0 enters its critical state. Then, P1 sets wantCS[1] to true and then checks its while loop condition. WantCS[0] is true, however, the turn variable was set to 1 by P0, so P1 passes the while loop condition. Therefore, it enters the critical section as well. This violates the safety condition.

5.

Let's assume that P1 is in its critical section and P0 is stuck in the while loop in the entry protocol. When P1 exits the critical section and sets wantCS[1] to false in its exit protocol. P0 will at that point pass the while loop condition and enter the critical state. Even if P1 attempts to enter the critical section immediately after exiting the critical section, it will set wantCS[1] to true and set the turn variable to 0, which will always cause P0 to break from the while loop and enter the critical section before P1 can re-enter its critical section. Thus, there is no situation where a process is trying to enter its critical section and does not eventually succeed.

6.

Using two turn variables, turn0 and turn1, such that P0 can only write to turn0 and P1 can only write to turn1, mutual exclusion can be achieved in the following

protocol: A process' entry protocol consists of setting its corresponding wantCS variable to true and then setting its turn variable to the other turn variable plus one (eg. P0: turn0 = turn1 + 1;). In the while loop condition, we check if the wantCS variable of the other process is true and if our turn variable is 2 (eg. P0: (wantCS[1] && turn0 == 2)). After clearing this condition, the last statement in the entry protocol is to set the turn variable back to 0 (eg. P0: turn0 = 0;) before entering the critical section. By using this protocol, the safety condition is always satisfied because when both processes execute the second statement in the entry protocol (turn0/1 = turn1/0 + 1), one process will always execute this statement first, resulting in its turn variable being equal to 1. The other turn variable will always equal 2 in this case. As a result, both processes can never pass the condition at the same time. Deadlocks are not possible because in the case where both processes are attempting to enter the critical section at the same time, it is not possible for both turn variables to be equal to 2 (the first one that executes will see a value of 0 for the other turn variable, resulting in a value of 1). Starvation freedom is ensured by the same reasoning as Peterson's algorithm's proof. If one process is stuck at the while loop and the other is in the critical section, once it leaves, it will set its wantCS variable to false which will always allow the waiting process to enter the critical section next before the leaving process can re-enter the critical section.