<u>Ex3</u>

<u>Itamar Chuvali</u>

<u>200049734</u>

Part 2

Question 1

a. Yes. Though both threads can overcome the line *while(busy),* only one at a time can reach the critical section as turn is either 1 or 0 and 'me' is a local variable set to the threadID, therefore locking the other thread out.

b. Deadlock can be achieved. Assume this order of operation:
   I. Thread 0 makes turn = 0
   II. Thread 1 makes turn = 1
   III. Thread 0 turns busy true

   In this situation, thread 1 will get stuck in the *while(busy)* state and thread 0 will get stuck in the *while (turn != me)* state as turn = 1 and me = 0.

c. Starvation can be achieved. As turn does not get changed in the exit code, if one of the threads exits its critical section and doesn't get recalled, the other thread can endlessly get stuck in the *while (turn != me)* state.

d. Mutual exclusion is still maintained as turn and me have to equal to enter the critical section.

   Deadlock can still be achieved as only a single thread can enter the critical section and that same thread can be stuck in the *while(busy)* state.

   Starvation can still be achieved for the same reasons as above.

Question 2

a. B array – this array *blocks* thread i from entering the critical section, conditioned on a resource being available to use. All threads where B[i] = true are waiting to enter.
   T array – the array is responsible for *taking* a resource and ensures that once a resource has been taken, the thread is ready to enter the critical section.

b. Yes. Assume thread0 and thread1 are both in the critical section. Then at some point both threads decremented value prior to either one of them incrementing value. Therefore WLOG thread[i] would not have changed B[i] to false which is a necessary criteria to entering the critical section.

c. We can see that the smaller indices are more likely to take preference over the bigger ones. This is because we turn T[j] to false starting from index 0 every time at the end of *Signal*. This means that for thread[50] to enter the critical section, it would essentially need all 49 other threads to not be waiting to enter.

d. The while loop is a busy waiting condition that unnecessarily consumes CPU usage that would otherwise be used more efficiently. Also, we can iterate j cyclically instead of starting from 0 to improve the starvation issue.

e. No. 2 threads can simultaneously be in their critical sections. Consider that both threads decrement value however they are currently still registering it as 0, therefore both B[i]'s will be turned to false and therefore enter the critical section.

Question 3

S1 = counting semaphore initiated as 2.

S2 , S3, S4, S5 are all binary semaphores initiated as 0.

Process 1	Process 2	Process 3	Process 4	Process 5

While (true) {
   Down (S1)
   Down (S1)
//p1
   Up (S2)
}
	While (true) {
	   Down (S2)
	//p2
	   Up (S3)
	   Up (S4)
	}
		While (true) {
		   Down (S3)
		//p3
		   Up (S5)
		}
			While (true) {
			   Down (S4)
			//p4
			   Up (S1)
			}
				While (true) {
				   Down (S5)
				//p5
				   Up (S1)
				}