**Elif Cemre Durgut – 26493**

**CS 307 – OPERATING SYSTEMS PA2**

**Pseudocode of my locking algorithm:**

pthread mutex;

int turn = 1; //initially turn is 1 (for X) becomes 2(for Y) in an alternating manner

thread\_func(arg): //arg is 1 for X, and 2 for Y

while the game is not over:

if turn and thread arg matches:

lock mutex

player chooses a random cell and marks it

turn variable is changed //If it is 1, becomes 2. If it is 2, becomes 1

unlock mutex

return null

main:

create thread 1

create thread 2

join thread 1

join thread 2

**Evaluation of my lock algorithm:**

1. **Correctness:**

It is correct because it provides a mutual exclusion, the lock allows only one of the threads to enter the critical part. Because the entrance depends on the turn variable and that variable is changed just before the unlock operation. Turn is changed only there not anywhere else. Two threads cannot get the lock at the same time unlike simple flag lock algorithm. Because arg == turn condition is different than the flag in a way that thread cannot enter the if block if arg and turn are not same. And this approach also prevents deadlocks.

1. **Fairness**

It is (more than) fair because threads must wait their turn. The turn variable is global and initally 1 so that the player X starts first to play. Turn is changed only in critical section which is protected with mutex (so it is secure). After the variable is changed, the lock is given to the other thread (because the arg = turn condition holds.). The threads enters the critical section one by one.

1. **Performance**

The thread spins until it acquires the lock therefore, it is not very good in term of performance. However, according to the homework document spinning is acceptable.