UĞUR KAĞAN ÇAKIR   27058   PA_2 REPORT

```
Define a struct to pass the char mark of the player to the threads
Define a struct to be used as the lock mechanism
Define the functions that will be used for locking & unlocking mechanisms and lock initializer
Initialize board, threads, lock object
Initialize int size to later update to the input taken from the screen
Initialize int empty_cells to later keep track of number of empty cells on the board
Initialize int isGameFinished which will keep track of the board’s status
Initialize char winner to hold the information of the winner thread
Void fill_the_board get the char of the player
    While game is not finished
        Lock the thread
        Get random row and column values
        While board [random row] [random column] is already taken
            Get random row and column values
        Insert the mark to the board
        Empty cells decreased by 1
        Check the winning status of the player
        Check horizontal
        Check vertical
        Check first diagonal
        Check second diagonal
        If the player won
            Change the winner
            Game has ended
        Else if all the cells are full
            Game has ended
```

```
Int main
    Take size input from standard input
    Create thread_info objects to be passed to the threads
    Set size value
    Set empty_cells value
    Allocate memory for board
    Fill the board with “-“
    Initialize lock
    Initialize threads
    If threads are created successfully
        Main thread waits until child threads terminates
        If there is a winner
            Print out the winner
        Else
            Print “It is a tie”
        Print out entire board
        Free the dynamically allocated memory
    Else (If there was a problem creating at least one of the threads)
        Print error message
```

Discussion of the Locking Mechanism:

I used a spinning lock mechanism that works on this specific problem. Firstly, the lock object is initialized, and when it is initialized it sets the turn variable as 0 and it stores the array of the character marks of the players. The first index of the array (marks[0]) is reserved to “x” and second index of the array (marks[1]) is reserved to “y”. The turn variable is there to hold the index of the mark that will play next. Turn variable is initialized to 0 to force player “x” starts first.

I transferred the turn information using the actual marks of the threads. In my mechanism, child threads get mark as a parameter then they pass it as a parameter to the lock mechanism. Inside the lock function the function spins until it gets its turn. Since there are only 2 child threads, which thread gets out of the lock will make its operations and then at the end with unlock the turn will be changed. The unlock function changes the turn variable from 1 to 0 or 0 to 1. With the change in the next iteration same thread can not pass the loop inside lock function, it starts spinning until it gets its turn back.

My implementation satisfies the needs described in the homework since, it operates the critical sections of the code in an atomic way (between lock and unlock mechanisms). There is no unexpected situation or any deadlocks in my implementation, the turn simply changes after each time that unlock is called. Moreover, my implementation is fair none of the threads starve. And finally, the child threads execute their turns in an alternating manner.