Name: Muhammed Orhun Gale **ID**: 26754

CS307 Programming Assignment-2 Report

Program Flow:

My program's flow is handled by three threads namely, "default main thread" and two "Player X thread" and "Player O thread". While the Main thread handles the I/O operations and memory allocations, its descendant threads "Player X thread" (thread-1) and "Player Y thread" (thread-2) makes moves on a globally defined game "board". Turn by turn, until a player achieves to get a full row/column/main-diagonal/anti-diagonal with its sign or the board becomes full, player threads makes their moves. After each move, program controls if the latest move was a winning move or a move that make the table full.

Since the shared object among the "player threads" is the "board", in case of race conditions, I used a mutex lock algorithm whenever one of these player threads accesses and wants to change a slot in the board.

Locking algorithm:

My locking algorithm basically works by considering which player has the turn currently. By using pthread_mutex, my algorithm blocks the access to the board for the player thread which does not have the turn.

Pseudocode:

-Create lock (mutex)

-In both player thread

- -While one of the players did not win the game yet, or table is not full

- - - Lock the mutex

- - - If -> Game is finished, unlock the mutex and exit from the thread

- - -Else if -> It is not your turn, unlock the mutex

- - - Else -> Make your move and check whether you won or not

- - - -If you won ->

- - - - -Set the game as finished

- - - - -Declare that your turn is finished

- - - - -Unlock the mutex and exit from the thread

- - - - Else (If you did not won) ->

- - - - - Give the turn

- - - - - Unlock the mutex

More clearly, I set a global lock and for each function that will be executed on different player threads, I implement the locking mechanism with the shared lock. Intuitively, in this algorithm, I am passing the right of using the lock via considering who has the turn.

Algorithm's Evaluation in terms of how it satisfies the stated requirements:

Race Condition:

As I previously stated, race condition is in the account while updating the board entries. I solved this problem by mutex (lock) via blocking the access to the board of the player thread which does not have the turn.

Turn Structure:

Assuming that the first play right is possessed by the 'X', I started the game by giving the first turn to the "X Player thread". Starting with 'X', after a player thread makes its move, it passes the turn to other player thread. While turn structure helps the program the keep track of the locks, structures itself benefits from the locks since turn passing and unlocking operations executed sequentially.

Deadlock:

Preclude deadlocks, my algorithm conditionally locks the mutex as follows,

1) If game is not finished
2) If the thread that wants to lock the mutex has the turn

Also, after each thread made their moves, they unlock the mutex.

Termination:

Termination of threads are bounded to the game's status. If the game is finished, i.e. one player won the game or the table became full, both threads exit from the thread by "pthread_exit(NULL)" and after they finished they joins to main thread that allows main thread to continue and finish its execution.