Yiğit Karamanlı

28105

yigitkaramanli@sabanciuniv.edu

LOCK SYSTEM

In this homework, I have used mutex locks from the pthread library. I used coarse grained global lock to lock access into the game matrix and ensure that only one of the threads can access the gameboard at any given time and thus made sure that no data race occurs during the execution of the program.

PROGRAM HIERARCHY

The program starts by executing srand(time(NULL)) command to ensure that the random number generator is thread free and will generate a different seed at each execution. After that the size of the matrix, which is taken as a command line argument, is assigned to the global integer "size" object and gameboard matrix is created by dynamically allocating memory using the CreateBoard() function. I have declared the matrix as a global variable. After that two structs, keeping the info of the X and O players, are created and two child threads are created using pthread_create function by giving the player function(gamer_boii) as the start routine argument.

The gamer_boii function starts with a while loop checking if the game has ended with one player winning (using a global variable) or the gameboard getting full (using isFull() function). Inside the loop the global variable "turn" is controlled to decide if it is the right players turn to play. (The turn variable starts as 'o' and the if condition checks if the turn variable is not the same as the player value, thus ensuring that player X starts first.) After that the mutex is locked and the game status is again controlled to check if the game has ended as the thread was waiting. If the conditions are satisfied, the function continues with selecting two random numbers using the rand() function and checking if the related matrix cell is empty or not. If the cell is empty, the player character is put into the cell and then winning conditions are checked for the player by checking the gameboard row by row, column by column, diagonal and reverse diagonal using isWinner() function. If the winning condition is satisfied then the global variable winner is changed as the player in turn's character and the loop conditions are changed, making it such that the thread coming after will not be able to play and will also return from the function. After the conditional changes the mutex lock is released thus giving access to the other thread.

Main thread waits for the children threads to join using the function pthread_join(). After execution of the threads are completed and the threads return, the appropriate game ending message is displayed and the final form of the gameboard is printed onto the terminal. Lastly, the dynamically allocated memory for the game matrix is freed to the heap and the program terminates.