# CHESS GAME USING MULTI-THREADING

**MINI PROJECT REPORT**

***Submitted by***


**ABIRAMI M (9517202109003)**

**BHUVANIKA S (9517202109011)**

**RAJAKUMARI S(9517202109042)**


***in***

**19AD481 – OPERATING SYSTEM PRINCIPLES**


**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**MEPCO SCHLENK ENGINEERING COLLEGE**

**SIVAKASI**


**MAY 2023**

# MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

# AUTONOMOUS

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

## BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **ABIRAMI M (9517202109003), BHUVANIKA S(9517202109011), RAJAKUMARI S (9517202109042)** for the mini project titled **"CHESS GAME USING MULTITHREADING"** in 19AD481 – Operating System Principles during the fourth semester January 2023 – May 2023 under my supervision.

**SIGNATURE**

**Mrs. P. Swathika,**
**Asst. Professor (Senior Grade),**
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

**SIGNATURE**

**Dr. J. Angela Jennifa Sujana,**
**Asso. Professor (Senior Grade)& Head,**
AI&DS Department
Mepco Schlenk Engg. College, Sivakasi

# ABSTRACT

A chess game implemention using multithreading involves creating separate threads for different aspects of the game, such as user input, computer moves, and game logic. The multithreading approach allows simultaneous execution of these threads, providing a more interactive and responsive gaming experience.

By utilizing multithreading, the chess game can provide real-time interaction between the user and the computer, allowing for simultaneous moves and smoother gameplay. The threads can run concurrently, independently executing their respective tasks, and communicating with each other as necessary to maintain a consistent game state.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

The game of chess has long been a fascinating and challenging activity that tests strategic thinking and decision-making skills. In modern implementations of chess games, multithreading has become increasingly popular to enhance the gaming experience. By utilizing multithreading, the game can provide improved responsiveness, real-time interaction, and concurrent execution of different game components.

The main components of the chess game using multithreading are:

User Thread: This thread handles user input, allowing the player to make moves by selecting the source and destination squares on the chessboard. It continuously prompts the user for input, validates the moves, and updates the game state accordingly. It also checks for game-ending conditions, such as checkmate or stalemate.

Computer Thread: This thread represents the computer's intelligence and handles generating the computer's moves. It may employ algorithms and strategies to make decisions on the best move to play. The computer thread communicates with the game logic to make valid moves and updates the game state accordingly.

Game Logic: The game logic is responsible for maintaining the state of the chessboard and enforcing the rules of the game. It validates moves, detects checkmate or stalemate conditions, and updates the board accordingly. It communicates with both the user thread and computer thread to ensure consistent gameplay.

Synchronization: Proper synchronization mechanisms, such as locks or semaphores, are employed to prevent race conditions and ensure thread safety. Access to shared data structures, such as the chessboard, needs to be properly synchronized to avoid conflicts between threads.

### 1.1.1 Pthreads:

Pthreads, short for POSIX threads, is a standard threading library in the POSIX (Portable Operating System Interface) specification. It provides a set of functions and APIs (Application Programming Interfaces) that allow for the creation, manipulation, and synchronization of threads within a multi-threaded program.

Pthreads is widely used on Unix-like operating systems, including Linux, macOS, and various flavors of Unix. It allows developers to write parallel and concurrent programs by utilizing multiple threads of execution that can run concurrently and independently.

Key features and functionalities of Pthreads include:

Thread Creation: Pthreads provides functions to create new threads within a program. The pthread_create function is used to create a new thread, specifying the thread attributes, the entry point function, and any arguments to be passed to the thread.

Thread Synchronization: Pthreads offers mechanisms for thread synchronization, allowing threads to coordinate their actions and avoid conflicts. This includes features such as mutexes (mutual exclusion), condition variables, barriers, and read-write locks. Mutexes protect critical sections of code, ensuring only one thread can access them at a time. Condition variables enable threads to wait for specific conditions before proceeding. Barriers synchronize multiple threads at designated points in the code, while read-write locks allow for shared or exclusive access to data.

Thread Termination and Joining: Pthreads provides functions for terminating and joining threads. The pthread_exit function is used to terminate a thread, exiting its execution. The pthread_join function allows one thread to wait for the termination of another thread and retrieve its exit status.

Thread Attributes: Pthreads allows developers to specify various attributes for threads, including stack size, scheduling policy, priority, and scope. Thread attributes can be set using functions such as pthread_attr_init and pthread_attr_setXXX.

Thread Local Storage: Pthreads supports thread-specific data, allowing each thread to have its own unique data that is not shared with other threads. This can be useful for maintaining thread-specific context or variables.

Thread Cancellation: Pthreads provides functions for canceling threads. A thread can be asynchronously or deferred canceled, depending on the cancellation type specified.

Inter-thread Communication: Pthreads facilitates communication and data sharing between threads. This can be achieved using shared variables or more advanced synchronization mechanisms such as condition variables or message queues.

Pthreads provides a flexible and efficient way to implement concurrent and parallel programs by leveraging the power of multiple threads. It allows developers to take advantage of multi-core processors and parallel architectures, enabling better utilization of system resources and improved program performance.

## 1.2 Objective

Enhanced User Experience: The primary objective of incorporating multithreading in a chess game is to enhance the user experience. By using separate threads for user input, computer moves, and game logic, the game can provide a more interactive and responsive gameplay environment. The concurrent execution of threads allows players to make moves, see the updated board state, and receive computer moves without experiencing noticeable delays or interruptions.

Real-Time Interaction: Multithreading enables real-time interaction between the user and the game. The user can input their moves, and the game can respond promptly, updating the board state and providing feedback. This real-time interaction adds to the immersive nature of the game and makes it more engaging and enjoyable for the players.

Concurrent Execution: By utilizing multithreading, different aspects of the chess game can be executed concurrently. The user thread can handle user input, allowing the player to make moves, while the computer thread can generate and execute its moves simultaneously. The game logic thread ensures the consistency and validity of the game by enforcing rules and updating the board. Concurrent execution maximizes the utilization of system resources and improves the overall efficiency of the game.

### 1.3 Scope of the project

It uses Multiplayer Support system. Multithreading can enable the implementation of a multiplayer feature, allowing players to engage in chess matches with each other over a network. Multiple threads can handle the communication between players, ensuring smooth and simultaneous gameplay.

It has Move Validation and Game Logic. Concurrent threads can handle move validation and enforce the rules of the chess game. They can check the legality of moves, detect checkmate or stalemate conditions, and ensure fair gameplay by preventing invalid or illegal moves.

# CHAPTER 2
# IMPLEMENTATION

## 2.1 Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// Define the chess board as a 2D array of characters
char board[8][8] = {
    {'r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'},
    {'p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'},
    {' ', '.', ' ', '.', ' ', '.', ' ', '.'},
    {'.', ' ', '.', ' ', '.', ' ', '.', ' '},
    {' ', '.', ' ', '.', ' ', '.', ' ', '.'},
    {'.', ' ', '.', ' ', '.', ' ', '.', ' '},
    {'P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'},
    {'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'}
};

int game_ended = 0;

// Define the user thread function
void* user_thread_func(void* arg) {
    int row, col;
    while (!game_ended) {  // Check if the game has ended
        // Get a move from the user
        printf("Enter your move:\n");
        int from_row, from_col, to_row, to_col;
```

```
        scanf("%d %d %d %d", &from_row, &from_col, &to_row, &to_col);

        // Check if the move is valid
        if (from_row >= 0 && from_row < 8 && from_col >= 0 &&
from_col < 8 &&
            to_row >= 0 && to_row < 8 && to_col >= 0 && to_col < 8 &&
            board[from_row][from_col] != ' ') {
            board[to_row][to_col] = board[from_row][from_col];
            board[from_row][from_col] = ' ';
        } else {
            printf("Invalid move\n");
        }

        // Print the updated chess board
        for (row = 0; row < 8; row++) {
            for (col = 0; col < 8; col++) {
                printf("%c ", board[row][col]);
            }
            printf("\n");
        }

        // Check if the game has ended
        if (from_row == -1 && from_col == -1 && to_row == -1 &&
to_col == -1) {
            game_ended = 1;
            break;
        }
    }
}

// Define the computer thread function
```

```
void* computer_thread_func(void* arg) {
    while (!game_ended) {  // Check if the game has ended
        // Generate a move for the computer (not implemented in this
example)

        // Check if the move is valid

        // Make the move on the chess board

        // Print the updated chess board

        // Check if the game has ended (e.g. a player has won or it's
a draw)
        // In this example, the game never ends, so we will just
break out of the loop
        break;
    }
}
int main() {
    // Create the user thread
    pthread_t user_thread;
    int user_thread_result = pthread_create(&user_thread, NULL,
user_thread_func, NULL);
    if (user_thread_result) {
        printf("Error creating userthread: %d\n",
user_thread_result);
exit(-1);
}

// Create the computer thread
pthread_t computer_thread;
```

```
int computer_thread_result = pthread_create(&computer_thread, NULL,
computer_thread_func, NULL);
if (computer_thread_result) {
    printf("Error creating computer thread: %d\n",
computer_thread_result);
    exit(-1);
}

// Wait for the threads to finish
pthread_join(user_thread, NULL);
pthread_join(computer_thread, NULL);

// Wait for the user to signal the end of the game
printf("Game over!\n");

return 0;
}
```

**2.2 Output**

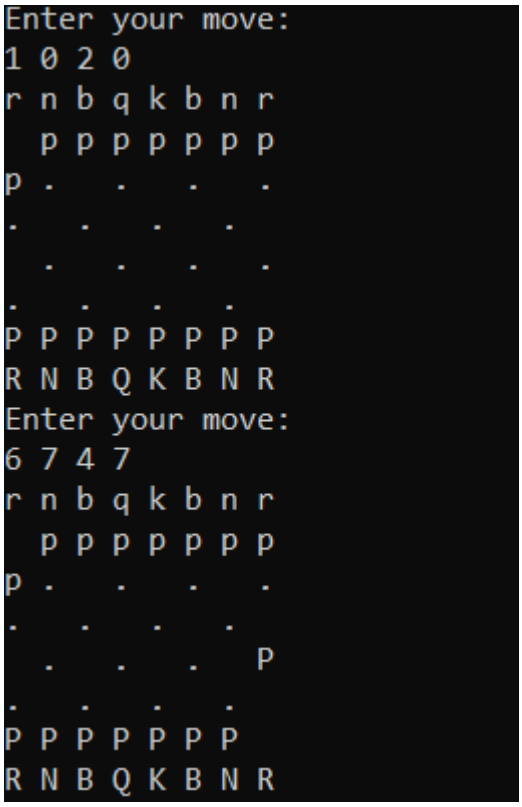


Figure 2.1.1 - chess module

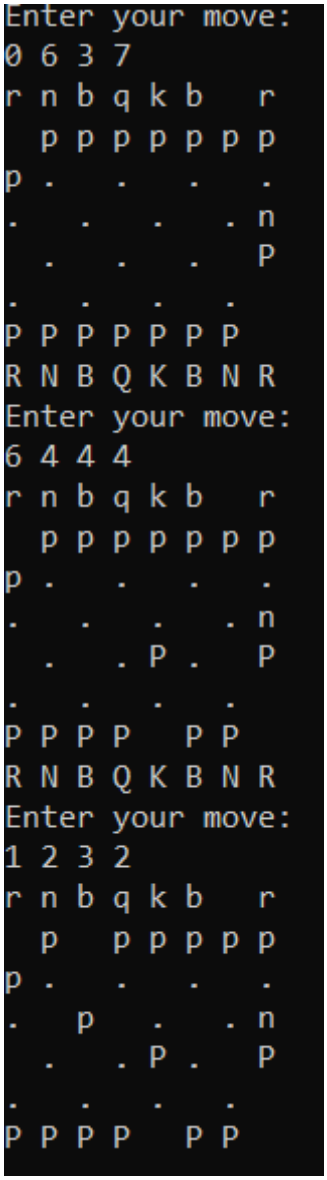

Figure 2.1.2 – Chess module

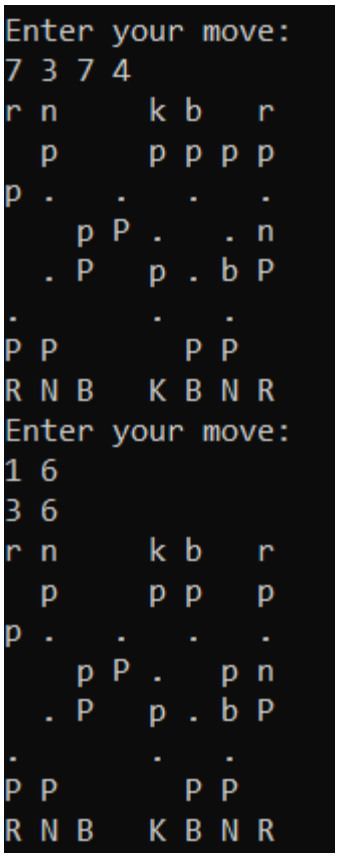

Figure 2.1.3 – Chess module

# CHAPTER 3

# CONCLUSION

Implementing a chess game using multithreading brings several advantages and enhancements to the gameplay experience. By utilizing multiple threads, the game can achieve concurrent execution, allowing for simultaneous player and computer moves and providing a more responsive and interactive environment for the players.

The use of multithreading enables real-time interaction between the user and the computer, making the game more engaging and dynamic. The user can input their moves and see the updated game state without waiting for the computer's response. Likewise, the computer can generate its moves and evaluate the board state concurrently, ensuring a smooth and uninterrupted game play.

# REFERENCES


1. https://codereview.stackexchange.com/questions/234619/chess-game-inc
2. https://www.youtube.com/watch?v=L7F4vsPl3sE
3. https://www.geeksforgeeks.org/multithreading-in-c/
4. https://www.tutorialspoint.com/multithreading-in-c