Project Report

Ludo Board Game Simulation

Sumeed Jawad Kanwar i22 2651

Abdul Rafay i22 8762

Introduction

This project involves the design and implementation of a multithreaded Ludo board game simulation. The game is played by 2 to 4 players, each controlling up to 4 tokens. The objective is to race the tokens from the start to the finish according to the rolls of a single die. The project is divided into two phases: Phase I focuses on setting up the game grid and initial token placement, while Phase II implements the complete game logic, including player turns, dice rolls, token movement, and game rules.

Phase I: Setting Up the Game Grid

Pseudo Code for Phase I

pseudo

Copy

1. Initialize global variables:

- Grid: 2D array representing the Ludo board
- Dice: Global variable representing the die

2. Create the Ludo board grid:

- Define the dimensions of the grid (e.g., 15x15)
- Mark the starting squares, safe squares, and home columns for each player

3. Place tokens in their respective home yards:

- For each player, place their tokens in the designated home yard area

4. Display the initial state of the grid:

- Print the grid with tokens placed in their home yards

5. Implement synchronization mechanisms:

- Use mutex locks to ensure that only one thread can access the dice or grid at a time

6. End of Phase I

Illustrations of Operating System Concepts Used

1. **Global Variables and Shared Resources**:
    - The grid and dice are defined as global variables, making them shared resources among all threads.
    - **Mutex Locks**: Used to ensure mutual exclusion when accessing the shared resources (dice and grid).
2. **Thread Synchronization**:
    - **Mutex**: Ensures that only one thread can roll the dice or modify the grid at a time, preventing race conditions.

Phase II: Complete Game Implementation

Pseudo Code for Phase II

pseudo

Copy

1. Initialize game parameters:

- Number of players (2 to 4)

- Number of tokens per player (1 to 4)

2. Create player threads:

- Each player thread represents a player in the game

- Pass parameters to each thread (e.g., player ID, number of tokens)

3. Implement game loop:

- Randomly select a player to take their turn

- Player rolls the dice and moves their tokens accordingly

- Implement game rules (e.g., entering tokens into play, hitting opponent tokens, safe squares)

4. Synchronization mechanisms:

- Use mutex locks to control access to the dice and grid

- Use conditional variables to ensure correct sequence of token movements

5. Display game state after each iteration:

- Print the grid with updated token positions

- Display player statistics (e.g., hit rate, number of tokens)

6. Check for game completion:

- The first player to move all tokens to the home triangle wins

- Continue play to determine second, third, and fourth place finishers

7. Implement thread cancellation:

- If a player cannot get a 6 or hit an opponent for 20 consecutive turns, cancel their thread

- If a player wins, signal the master thread to cancel their thread

Illustrations of Operating System Concepts Used

1. **Thread Creation and Management**:
    - **pthread_create()**: Creates threads for each player.
    - **pthread_join()**: Waits for threads to complete.
    - **pthread_cancel()**: Cancels threads based on game conditions.
2. **Synchronization**:
    - **Mutex**: Ensures mutual exclusion when accessing shared resources (dice and grid).
    - **Conditional Variables**: Ensures that token movements are performed in the correct sequence.
3. **Semaphores**:
    - Used to control the number of tokens in play and ensure that tokens are moved correctly according to dice rolls.

System Specifications

- **Operating System**: Ubuntu 20.04 LTS
- **Programming Language**: C++

- **Compiler**: G++ (GNU Compiler Collection)
- **Hardware Specifications**:
    - MSI Sword 17
    - Processor: Intel Core i7 11th Generation
    - Graphics Processor: Nvidia RTX 3060
    - RAM: 16 GB
    - Storage: 1 TB SSD

  Backup Specification:

    - Processor: Intel Core i5 8th Generation
    - Graphics Processor: Intel UHD Graphics
    - RAM: 16 GB
    - Storage: 512 GB SSD

Implementation in Other Scenarios

The synchronization techniques and thread management strategies used in this project can be applied to other scenarios where multiple threads need to access shared resources. For example, in a multi-user database system, each user can be represented by a thread that accesses the database. Using mutex locks and conditional variables can ensure that database transactions are performed in a consistent and orderly manner, preventing data corruption and race conditions.

Conclusion

This project successfully implements a multithreaded Ludo board game simulation, demonstrating the use of various operating system concepts such as thread management, synchronization, and shared resource control. The project is divided into two phases, with Phase I focusing on setting up the game grid and Phase II implementing the complete game logic. The use of mutex locks, conditional variables, and semaphores ensures that the game runs smoothly and fairly, adhering to the rules of Ludo.