Institute of Space Technology, Islamabad

Project Report

Operating Systems

COURSE INSTRUCTOR

Ms. Sabahat

SUBMITTED BY

Ariba Shuaib

Computer Science Department

Project Title:

Multithreaded Ludo Game Using POSIX Threads and Semaphores

Programming Language & Tools

- Language: C++
- Libraries Used: pthread.h, semaphore.h, unistd.h, cstdlib, ctime, iostream
- Platform: Linux / UNIX-based system

1. Introduction

This project is a console-based implementation of the popular board game **Ludo**, developed as part of the Operating Systems Lab. The main goal of this project is to demonstrate key operating system concepts such as **multithreading**, **process synchronization**, and **semaphores**, while building a functional and interactive game.

Each player in the game is represented using a separate thread, and synchronization is handled using semaphores to ensure fair turn-taking and avoid race conditions.

2. Objectives of the Project

- To understand and implement POSIX threads (pthreads)
- To apply semaphores for synchronization
- To simulate real-world concurrency using a game-based approach
- To design a shared resource (game board) accessed safely by multiple threads
- To strengthen understanding of Operating Systems concepts through practice

3. Game Overview

- The game supports 2 to 4 players
- Each player can have 1 to 4 tokens
- Dice rolls are randomized (1–6)
- Tokens move according to standard Ludo rules
- Safe points, token killing, and winning conditions are implemented
- The game continues until all players finish or are eliminated

4. Operating System Concepts Used

4.1 Multithreading

- Each player is executed as a separate thread using pthread_create()

- Threads allow players to run concurrently
- Player logic is handled inside the PlayerThread() function

4.2 Synchronization Using Semaphores

- A semaphore (sem_t sem) is used to control access to shared resources
- Only one player thread can access the board at a time
- sem_wait() is used before a player's turn starts
- sem_post() is used after the turn ends

This ensures **mutual exclusion** and prevents race conditions.

4.3 Critical Section

The following shared resources are protected using semaphores:

- Game board
- Player positions
- Dice rolls
- Token movements

5. Data Structures Used

5.1 Cell Class

Represents each cell of the Ludo board

```
class Cell {
   int x, y;
   int value;
};
```

5.2 Player Class

Stores information about each player and their tokens

```
class Player {
   int x, y;
   int team;
   int index;
   int id;
   int removed;
   int position;
```

```
    bool ingame;
    int counter;
    string name;
};
```

6. Game Logic Explanation

6.1 Dice Rolling

- Dice values are generated using rand()
- If a player gets three consecutive sixes, the turn is skipped

6.2 Token Movement

- Tokens start inside their home
- A token enters the board only when dice value is 6
- Tokens move step-by-step based on dice value
- Home path is entered after killing at least one opponent token

6.3 Safe Points

Certain board positions are marked as safe points where tokens cannot be killed.

6.4 Token Killing

- If two tokens from different players land on the same non-safe cell
- The opponent token is sent back to home
- The attacking player’s kill count increases

7. Thread Management

Player Thread

- Handles dice rolling
- Takes user input
- Moves tokens
- Calls the master thread for validation

Master Thread

- Checks for token killing
- Updates win conditions

- Removes players after 20 inactive turns
- Updates final positions

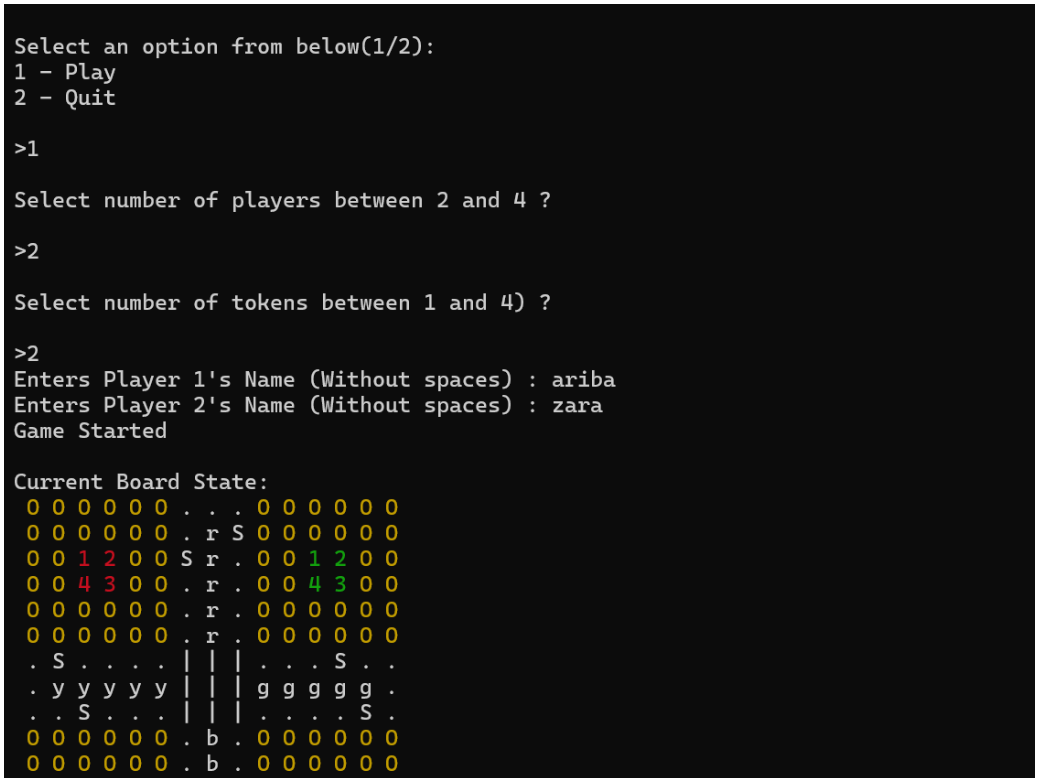
8. Game Board Rendering

- The board is displayed using a 15×15 grid
- Different colors are used for players:
    - Red
    - Green
    - Cyan
    - Yellow
- ANSI escape codes are used for coloring

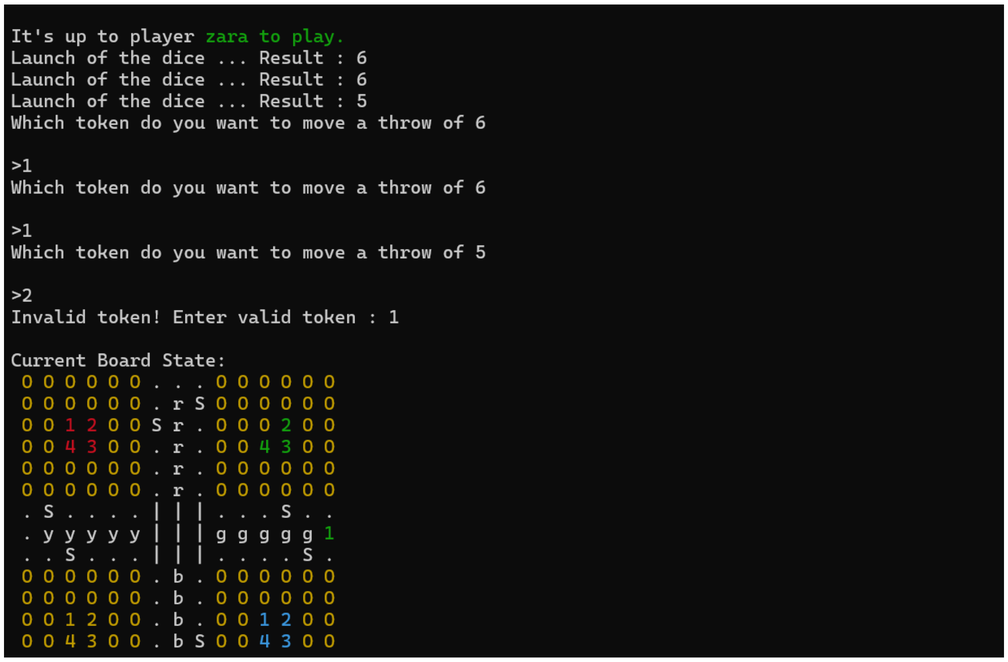
9. Result and Output

At the end of the game, the program displays:

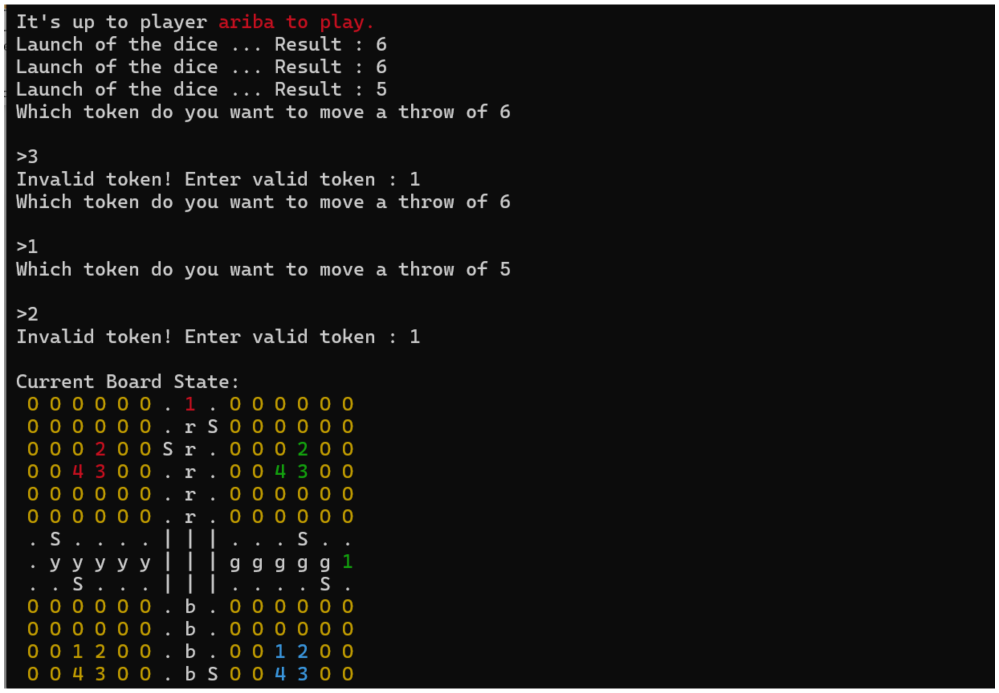
- Player names
- Final positions
- Number of tokens
- Hit rate (number of opponent tokens killed)



```
Select an option from below(1/2):
1 - Play
2 - Quit

>1

Select number of players between 2 and 4 ?

>2

Select number of tokens between 1 and 4) ?

>2
Enters Player 1's Name (Without spaces) : ariba
Enters Player 2's Name (Without spaces) : zara
Game Started

Current Board State:
 0 0 0 0 0 0 . . . 0 0 0 0 0 0
 0 0 0 0 0 0 . r S 0 0 0 0 0 0
 0 0 1 2 0 0 S r . 0 0 1 2 0 0
 0 0 4 3 0 0 . r . 0 0 4 3 0 0
 0 0 0 0 0 0 . r . 0 0 0 0 0 0
 0 0 0 0 0 0 . r . 0 0 0 0 0 0
 . S . . . . | | | . . . S . .
 . y y y y y | | | g g g g g .
 . . S . . . | | | . . . . S .
 0 0 0 0 0 0 . b . 0 0 0 0 0 0
 0 0 0 0 0 0 . b . 0 0 0 0 0 0
```



```
It's up to player zara to play.
Launch of the dice ... Result : 6
Launch of the dice ... Result : 6
Launch of the dice ... Result : 5
Which token do you want to move a throw of 6

>1
Which token do you want to move a throw of 6

>1
Which token do you want to move a throw of 5

>2
Invalid token! Enter valid token : 1

Current Board State:
 0 0 0 0 0 0 . . . 0 0 0 0 0 0
 0 0 0 0 0 0 . r S 0 0 0 0 0 0
 0 0 1 2 0 0 S r . 0 0 0 2 0 0
 0 0 4 3 0 0 . r . 0 0 4 3 0 0
 0 0 0 0 0 0 . r . 0 0 0 0 0 0
 0 0 0 0 0 0 . r . 0 0 0 0 0 0
 . S . . . . | | | . . . S . .
 . y y y y y | | | g g g g g 1
 . . S . . . | | | . . . . S .
 0 0 0 0 0 0 . b . 0 0 0 0 0 0
 0 0 0 0 0 0 . b . 0 0 0 0 0 0
 0 0 1 2 0 0 . b . 0 0 1 2 0 0
 0 0 4 3 0 0 . b S 0 0 4 3 0 0
```



```
It's up to player ariba to play.
Launch of the dice ... Result : 6
Launch of the dice ... Result : 6
Launch of the dice ... Result : 5
Which token do you want to move a throw of 6

>3
Invalid token! Enter valid token : 1
Which token do you want to move a throw of 6

>1
Which token do you want to move a throw of 5

>2
Invalid token! Enter valid token : 1

Current Board State:
 0 0 0 0 0 0 . 1 . 0 0 0 0 0 0
 0 0 0 0 0 0 . r S 0 0 0 0 0 0
 0 0 0 2 0 0 S r . 0 0 0 2 0 0
 0 0 4 3 0 0 . r . 0 0 4 3 0 0
 0 0 0 0 0 0 . r . 0 0 0 0 0 0
 0 0 0 0 0 0 . r . 0 0 0 0 0 0
 . S . . . . | | | . . . S . .
 . y y y y y | | | g g g g g 1
 . . S . . . | | | . . . . S .
 0 0 0 0 0 0 . b . 0 0 0 0 0 0
 0 0 0 0 0 0 . b . 0 0 0 0 0 0
 0 0 1 2 0 0 . b . 0 0 1 2 0 0
 0 0 4 3 0 0 . b S 0 0 4 3 0 0
```

11. Conclusion

This project successfully demonstrates the practical use of Operating System concepts such as multithreading and synchronization through an interactive Ludo game. By using threads for players and semaphores for synchronization, we ensured fair gameplay and safe access to shared resources. The project helped us understand how OS concepts are applied in real-world applications.