

***Operating Systems***

***Semester Project***

***Date:*** *5th Decemeber, 2024*

***Section:*** *SE-F*

***Name & Roll-no:***

*Rayyan Ahmed (22I-2489)*

*Usman Ghani (22I-8796)*

*Safdar Jan (22I-8804)*

***To: Mam Anum Kaleem***

***Multithreaded Ludo Game in C++***

**Table of Contents:**

1. Introduction

2. Project Overview

3. Operating System Concepts Utilized

* Threads
* Semaphores
* Synchronization and Mutual Exclusion
* Thread Cancellation and Termination

4. Phase I and II Pseudocode with Illustrations

- Phase I: Thread Creation and Initialization

- Phase II: Synchronization and Game Logic

5. Implemented Code Overview

6. System Specifications

7. Application of Concepts in Another Scenario

8. Conclusion

9. References

1. **Introduction:**

This project involves developing a multithreaded Ludo game using C++. The game simulates concurrent gameplay by utilizing operating system concepts such as threads, semaphores, and synchronization mechanisms. The focus is on illustrating how these OS concepts can be applied in a practical application to manage concurrency and resource sharing.

1. **Project Overview**

* Language Used: C++
* Operating System: Linux (for POSIX thread support)
* Libraries and Tools:
* Pthread Library: For threading and synchronization.
* OpenGL Graphics: For rendering the game board and tokens.

1. **Operating System Concepts Utilized:**

**Threads**

* Purpose: Allow multiple sequences of programmed instructions to run concurrently.
* Implementation: Each player in the game is represented by a separate thread. An additional master thread monitors and manages the player threads.

**Semaphores**

* Purpose: Control access to shared resources and synchronize thread execution.
* Implementation: Semaphores manage access to the dice, game board grid, player turns, and win conditions.

**Synchronization and Mutual Exclusion**

* Purpose: Prevent race conditions and ensure data consistency when multiple threads access shared resources.
* Implementation: Critical sections are protected using semaphores to ensure that only one thread can execute a critical section at a time.

**Thread Cancellation and Termination**

* Purpose: Allow threads to be terminated before completion, freeing up system resources.
* Implementation: The master thread can cancel player threads when they have completed their tasks (i.e., when a player has won the game).

1. **Phase I and II Pseudocode with Illustrations:**

**Phase I: Thread Creation and Initialization**

Objective: Initialize the game by creating player threads and setting up shared resources.

Pseudocode

BEGIN MAIN PROGRAM

Initialize game variables

Initialize semaphores:

Initialize Dice\_Roll semaphore to 1

Initialize Grid\_Access semaphore to 1

Initialize RoundTurns semaphore to 1

Initialize winCheck semaphore to 1

Create Master Thread

FOR each player index FROM 0 TO NumberOfPlayers - 1 DO

Create Player Thread with PlayerTurn function

END FOR

WAIT for all threads to finish

END MAIN PROGRAM

**OS Concepts Illustrated:**

* + Thread Creation: Using `pthread\_create` to create master and player threads.
  + Semaphore Initialization: Using `sem\_init` to initialize semaphores for synchronization.
  + Resource Sharing: Setting up shared variables and resources accessed by multiple threads.

**Phase II: Synchronization and Game Logic**

Objective: Manage game flow by synchronizing thread actions and controlling access to shared resources.

**Pseudocode for PlayerTurn Function**

FUNCTION PlayerTurn(playerIndex)

WHILE game is not over DO

Wait on RoundTurns semaphore

IF it's this player's turn THEN

Wait on Dice\_Roll semaphore

Roll the dice and get score

Release Dice\_Roll semaphore

Wait on Grid\_Access semaphore

Move the player's token based on score

Release Grid\_Access semaphore

Check for win condition

IF player has won THEN

Signal winCheck semaphore

Notify Master Thread of completion

EXIT loop

END IF

END IF

Release RoundTurns semaphore

END WHILE

TERMINATE thread

END FUNCTION

**Pseudocode for Master\_Thread Function**

FUNCTION Master\_Thread()

WHILE game is not over DO

FOR each player index FROM 0 TO NumberOfPlayers - 1 DO

IF player has signaled completion THEN

Wait on winCheck semaphore

Cancel Player Thread

Update game state

Release winCheck semaphore

END IF

END FOR

END WHILE

TERMINATE thread

END FUNCTION

**OS Concepts Illustrated:**

* + Semaphores for Synchronization: Using `sem\_wait` and `sem\_post` to control player turns and resource access.
  + Critical Sections: Protecting critical sections to prevent race conditions.
  + Inter-thread Communication: Threads signaling each other through shared variables and semaphores.
  + Thread Termination: Master thread using `pthread\_cancel` to terminate completed player threads.

1. **Implemented Code Overview:**

1. Main Function (`game.cpp`)

* + Purpose: Initialize the game environment and semaphores, create threads, and start the graphics loop.
  + Key Functions:
  + `main()`: Entry point of the program.
  + `pthread\_create()`: Creates the master and player threads.
  + `sem\_init()`: Initializes semaphores for synchronization.

2. Player Thread Function (`globals.cpp`)

* + Purpose: Represents each player's actions, including dice rolls and token movements.
  + Key Functions:
  + `PlayerTurn(void arg)`: Function executed by each player thread.
  + `sem\_wait()` and `sem\_post()`: Synchronize access to shared resources.

3. Master Thread Function (`globals.cpp`)

* + Purpose: Monitors player threads and manages game state.
  + Key Functions:
  + `Master\_Thread(void arg)`: Function executed by the master thread.
  + `pthread\_cancel()`: Cancels player threads upon completion.
  + `sem\_wait()` and `sem\_post()`: Synchronize updates to game state variables.

4. Game Board Rendering (`board.h` and util.cpp)

* + Purpose: Render the game board and UI elements using OpenGL.
  + Key Functions:
  + `DrawSquare()`, `DrawCircle()`, `DrawRectangle()`: Functions for drawing shapes.
  + `GameDisplay()`: Manages the rendering loop.

1. **System Specifications:**
   * Operating System: Ubuntu 20.04 LTS (Linux)
   * Processor: Intel Core i5 Quad-Core CPU @ 2.5GHz
   * RAM: 8 GB DDR4
   * Graphics: Integrated Intel HD Graphics
   * Compiler: GCC/G++ version 9.3.0
   * Installed Libraries:
   * `pthread` library for threading.
   * OpenGL libraries (`freeglut3`, `libgl1-mesa-dev`, `libglu1-mesa-dev`) for graphics rendering.
2. **Application of Concepts in Another Scenario:**

Scenario: Multithreaded Web Server

The operating system concepts used in this Ludo game can be applied to develop a multithreaded web server.

- **Threads:** Each incoming client request is handled by a separate thread, allowing concurrent processing without blocking.

- **Semaphores and Synchronization:**

- Resource Access: Semaphores control access to shared resources like log files or databases.

- Load Balancing: Manage the load by synchronizing access to limited resources.

- Thread Cancellation:

- Timeout Handling: Terminate threads that exceed processing time limits, freeing resources.

**Advantages:**

* + Improved Performance: Efficient CPU utilization through concurrency.
  + Scalability: Ability to handle more clients by spawning additional threads.
  + Responsiveness: Reduced latency due to parallel request handling.

1. **Conclusion:**

This project effectively demonstrates the application of operating system concepts such as threads, semaphores, synchronization, and thread management in a practical application. By utilizing these concepts, the multithreaded Ludo game achieves concurrency, efficient resource management, and a responsive gaming experience. The detailed pseudocode and explanations provide insight into how these OS concepts are implemented in the game's logic.

**9. References:**

- Programming with POSIX Threads by David R. Butenhof

- OpenGL Programming Guide by Dave Shreiner et al.

- Operating System Concepts by Abraham Silberschatz et al.

- Official OpenGL Documentation: [https://www.opengl.org/documentation/](https://www.opengl.org/documentation/)

- Pthreads Tutorial: [https://computing.llnl.gov/tutorials/pthreads/](https://computing.llnl.gov/tutorials/pthreads/)