|  |  |
| --- | --- |
|  |  |
| **Andleeb Zahra** | **21i-2741** |

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**To implement the described scenario, we need to manage access to shared resources (the grid and the dice) using semaphores to ensure that only one resource can be accessed by a thread at a time. Below is the pseudo code for the initialization and phase I of the Ludo game:**

# Global variables

Semaphore diceSemaphore = new Semaphore(1)

Semaphore gridSemaphore = new Semaphore(1)

Grid ludoBoard

Dice dice

# Function to initialize the Ludo grid Function initializeLudoGrid():

ludoBoard = createLudoBoard() placeTokensInHomeYards(ludoBoard)

# Function to display the complete Ludo grid Function displayLudoGrid(): displayGrid(ludoBoard)

# Function representing a player action Function playerAction(playerID): # Try to access the dice diceSemaphore.wait() rollDice(playerID) diceSemaphore.signal() # Release the dice resource

# Try to access the Ludo board gridSemaphore.wait() moveTokenOnBoard(playerID, dice) gridSemaphore.signal() # Release the grid resource

# Main function

Function main():

# Initialize the Ludo grid initializeLudoGrid()

# Create player threads

Thread player1 = createThread(playerAction, 1)

Thread player2 = createThread(playerAction, 2)

Thread player3 = createThread(playerAction, 3)

Thread player4 = createThread(playerAction, 4)

# Display the initial Ludo grid displayLudoGrid()

# Start player threads player1.start() player2.start() player3.start() player4.start()

# Join player threads (optional, depends on game logic) player1.join() player2.join() player3.join() player4.join()

# Utility functions

Function createLudoBoard():

# Logic to create and initialize Ludo board grid return new LudoBoard()

Function placeTokensInHomeYards(board):

# Logic to place tokens in their home yards on the board

Function rollDice(playerID):

# Logic for a player to roll the dice print("Player " + playerID + " rolls the dice.")

Function moveTokenOnBoard(playerID, dice):

# Logic for a player to move token on the board based on dice result print("Player " + playerID + " moves token on the board.")

Function displayGrid(board):

# Logic to display the Ludo board grid

print(board)

# Start the main function main()

**Explanation:**

1. Semaphores Initialization: Two semaphores (diceSemaphore and gridSemaphore) are initialized to control access to the dice and the grid respectively.
2. Grid Initialization: The initializeLudoGrid function creates and sets up the Ludo board and places the tokens in their home yards.
3. Display Grid: The displayLudoGrid function is used to display the initial state of the Ludo grid.
4. Player Actions: The playerAction function simulates a player's turn, where the player tries to access the dice and grid sequentially, ensuring mutual exclusion using semaphores.
5. Main Function: The main function initializes the game, creates player threads, displays the initial grid, and starts the player threads.

# —--------------------

**To implement the complete Ludo game with the given conditions, we'll extend the initial structure and add more functionalities for token movement, player turns, safe squares, and handling mutual exclusion. Here’s the pseudo code for Phase 2:**

# Global variables

Semaphore diceSemaphore = new Semaphore(1)

Semaphore gridSemaphore = new Semaphore(1)

Semaphore[] tokenSemaphores

Grid ludoBoard Dice dice int numTokens

ConditionVariable lastTurn = new ConditionVariable() int previousPlayerID = -1

# Function to initialize the Ludo grid and semaphores for tokens Function initializeLudoGridAndTokens(numTokens): ludoBoard = createLudoBoard() placeTokensInHomeYards(ludoBoard, numTokens) tokenSemaphores = new Semaphore[numTokens]

for i from 0 to numTokens - 1: tokenSemaphores[i] = new Semaphore(1)

# Function to display the complete Ludo grid Function displayLudoGrid(): displayGrid(ludoBoard)

# Function to get random player turn Function getRandomTurn(players): return random(1, players)

# Function representing a player action Function playerAction(playerID): while not gameFinished(): # Try to access the dice diceSemaphore.wait() diceValue = rollDice(playerID) diceSemaphore.signal() # Release the dice resource

# Ensure the player gets Ludo board access in the correct order lastTurn.wait() if previousPlayerID == -1 or previousPlayerID == playerID:

previousPlayerID = playerID lastTurn.signal()

else: lastTurn.wait()

# Try to access the Ludo board gridSemaphore.wait() moveTokenOnBoard(playerID, diceValue) gridSemaphore.signal() # Release the grid resource

# Update the previous player ID previousPlayerID = playerID lastTurn.signal()

# Main function Function main():

# Take user input for number of tokens numTokens = getUserInput("Enter the number of tokens for each player (1-4):")

# Initialize the Ludo grid and token semaphores initializeLudoGridAndTokens(numTokens)

# Create player threads

Thread player1 = createThread(playerAction, 1)

Thread player2 = createThread(playerAction, 2)

Thread player3 = createThread(playerAction, 3)

Thread player4 = createThread(playerAction, 4)

# Display the initial Ludo grid displayLudoGrid()

# Start player threads player1.start() player2.start() player3.start() player4.start()

# Join player threads (optional, depends on game logic) player1.join() player2.join() player3.join() player4.join()

# Utility functions

Function createLudoBoard():

# Logic to create and initialize Ludo board grid return new LudoBoard()

Function placeTokensInHomeYards(board, numTokens): # Logic to place tokens in their home yards on the board

Function rollDice(playerID):

# Logic for a player to roll the dice diceValue = random(1, 6) print("Player " + playerID + " rolls the dice and gets " + diceValue + ".") return diceValue

Function moveTokenOnBoard(playerID, diceValue):

# Logic for a player to move token on the board based on dice result # Handle mutual exclusion and token movement rules print("Player " + playerID + " moves token " + diceValue + " steps.") Function displayGrid(board):

# Logic to display the Ludo board grid print(board)

Function gameFinished():

# Logic to determine if the game is finished return False

Function getUserInput(prompt): # Logic to get user input return input(prompt)

# Start the main function main()

**Explanation:**

1. Semaphores and Condition Variables: The dice and grid are protected by semaphores to ensure mutual exclusion. tokenSemaphores are initialized for token-specific synchronization if needed. The lastTurn condition variable ensures that the player who had the last turn gets priority access to the Ludo board.
2. Initialization and User Input: The initializeLudoGridAndTokens function initializes the Ludo board and token semaphores based on user input for the number of tokens.
3. Player Actions: The playerAction function simulates each player's turn. Players try to roll the dice and then move tokens on the board, ensuring mutual exclusion using semaphores. The condition variable lastTurn ensures proper sequence in accessing the Ludo board.
4. Main Function: The main function initializes the game, takes user input, sets up the board, creates and starts player threads, and displays the initial grid.
5. Utility Functions: Helper functions like createLudoBoard, placeTokensInHomeYards, rollDice, moveTokenOnBoard, displayGrid, and gameFinished handle specific tasks within the game.

**In the pseudo code provided for implementing the Ludo game, several key operating system concepts are utilized to manage concurrency, synchronization, and resource sharing. Below are illustrations and explanations of these concepts:**

1. **Semaphores:**

Concept: A semaphore is a synchronization primitive used to control access to a common resource by multiple processes in a concurrent system.

Usage in Pseudo Code:

* + diceSemaphore and gridSemaphore are used to ensure that only one player can access the dice or the Ludo board at a time.
  + tokenSemaphores are initialized for individual tokens, if needed, to manage their movement safely.

**Illustration:**

Semaphore diceSemaphore = new Semaphore(1)

Semaphore gridSemaphore = new Semaphore(1)

Semaphore[] tokenSemaphores

Explanation:

* + new Semaphore(1) creates a semaphore that allows only one thread to access the resource at a time.
  + Before a player rolls the dice or moves a token, they must wait (acquire) the corresponding semaphore. After completing their action, they signal (release) the semaphore.

1. **Condition Variables:**

Concept: A condition variable is used to block a thread until a particular condition is true. It is typically used with a mutex to avoid race conditions.

Usage in Pseudo Code:

* + lastTurn is a condition variable that ensures the correct sequence of players accessing the Ludo board, based on the previous turn.

**Illustration:**

ConditionVariable lastTurn = new ConditionVariable() int previousPlayerID = -1

Explanation:

* + lastTurn.wait() blocks the current player if it's not their turn to move.
  + lastTurn.signal() wakes up the next player in sequence after the current player has completed their turn.

1. **Threads:**

Concept: Threads allow multiple sequences of instructions to run concurrently within the same program.

Usage in Pseudo Code:

* + Each player in the Ludo game is represented by a separate thread, allowing players to perform actions concurrently.

**Illustration:**

Thread player1 = createThread(playerAction, 1)

Thread player2 = createThread(playerAction, 2)

Thread player3 = createThread(playerAction, 3)

Thread player4 = createThread(playerAction, 4)

1. **Mutual Exclusion:**

Concept: Mutual exclusion ensures that only one thread can access a critical section of code at a time, preventing race conditions.

Usage in Pseudo Code:

* + Semaphores are used to enforce mutual exclusion for accessing the dice and the Ludo board.

**Illustration:**

# Try to access the dice diceSemaphore.wait() diceValue = rollDice(playerID) diceSemaphore.signal()

# Ensure the player gets Ludo board access in the correct order lastTurn.wait() if previousPlayerID == -1 or previousPlayerID == playerID:

previousPlayerID = playerID lastTurn.signal()

else: lastTurn.wait()

# Try to access the Ludo board gridSemaphore.wait() moveTokenOnBoard(playerID, diceValue) gridSemaphore.signal()

Explanation:

* diceSemaphore.wait() ensures only one player can roll the dice at a time.
* gridSemaphore.wait() ensures only one player can move a token on the board at a time.
* lastTurn condition variable ensures players access the board in the correct sequence based on their previous turn.

**5. Resource Sharing:**

Concept: Resource sharing involves managing access to shared resources (like the dice and the Ludo board) among multiple threads.

Usage in Pseudo Code:

* The dice and the Ludo board are global shared resources that must be accessed in a controlled manner using semaphores.

**Illustration:**

Global variables:

Semaphore diceSemaphore = new Semaphore(1)

Semaphore gridSemaphore = new Semaphore(1)

Grid ludoBoard

Dice dice

Explanation:

* dice and ludoBoard are shared resources accessed by multiple player threads.
* Semaphores manage access to these resources to prevent conflicts and ensure fair gameplay.

**Summary**

These operating system concepts ensure that the Ludo game runs smoothly with proper synchronization and resource management. Each concept is implemented through specific structures and mechanisms in the pseudo code, providing a robust foundation for concurrent gameplay.

By leveraging semaphores, condition variables, threads, and mutual exclusion, we effectively manage the complexity of concurrent actions and resource sharing in the Ludo game, preventing race conditions and ensuring a fair and synchronized game environment.

**System Specifications:**

**Hardware**

* Processor (CPU): Intel Core i5 (4 cores, 8 threads, with a base clock speed of

2.5 GHz and turbo boost up to 3.5 GHz)

* RAM: 8 GB DDR4
* Storage:
* SSD: 256 GB (for fast read/write operations and quick boot times)
* HDD: 1 TB (for additional storage, if needed)
* Graphics: Integrated Intel HD Graphics 620 (or equivalent) ● Display: 14-inch Full HD (1920 x 1080) IPS display ● Network:
* Wi-Fi: 802.11ac
* Bluetooth: 4.2 ● Ports:
* USB 3.0 ports: 2
* USB-C port: 1
* HDMI port: 1
* Audio jack: 1

**Short paragraph how could you implement these concepts in some other scenario:**

Implementing a Ludo game using operating system concepts such as semaphores and threads in a multithreaded environment showcases the practical application of these concepts in managing concurrent processes and synchronization. This approach can be extended to other scenarios where multiple processes or threads need to operate concurrently while sharing resources. For instance, in a banking system, threads could represent different transactions occurring simultaneously, with semaphores ensuring that critical sections, like account balances, are accessed safely to prevent race conditions. Similarly, in a networked chat application, threads could handle multiple user messages concurrently, using semaphores to manage access to shared data structures like message queues. These concepts can also be applied in manufacturing systems, where different assembly line robots (threads) need to coordinate and synchronize their tasks to ensure smooth production without conflicts. By utilizing multithreading and synchronization mechanisms, one can efficiently manage complex systems requiring high levels of concurrency and resource sharing.