**Semester Project**

**CS2006-Operating System**

**Aadil Raja 22k-4143**

**Uzain Ahmed 22k-4319**

**Syed Muhammad Bilal 22k-4492**

****

**Fast NUCES, Karachi**

**Department Of Computer Science**

**Multi-threaded Banking Account Simulator with Synchronized Operations Using Spinlocks**

**Introduction**

Concurrency in programming introduces challenges related to thread safety and data integrity, particularly when multiple threads access shared resources. To address this, synchronization mechanisms are crucial for coordinating access and avoiding data corruption.

This project **focuses** on implementing spinlock synchronization to secure concurrent operations on a shared banking account. Spinlocks offer a lightweight synchronization solution, busy-waiting until a lock is available. By applying spinlocks, we **aim** to ensure mutual exclusion and prevent race conditions during deposits, withdrawals, or modifications to the account balance.

The project simulates a banking system, demonstrating the effectiveness of spinlocks in maintaining consistency and preventing concurrency issues.

**Background**

**Research** into concurrent programming and synchronization mechanisms like spinlocks highlighted the need to manage concurrent access to shared resources. In the context of banking, ensuring that simultaneous operations do not lead to data inconsistencies or race conditions is crucial. The project was **selected** to explore synchronization, threading, and use of shared memory(Bank Account) in an online banking context offering a practical application of spinlock synchronization to secure concurrent operations.

**Project Specification**

The project implements an banking simulation with the following specifications:

* A **graphical user interface (GUI)** to interact which includes buttons,input text fields,check boxes. A suitable color combination is also used.
* Multiple **banking operations** such as deposit, withdrawal, PIN change, service fee application,Transfer Funds, account freezing, and unfreezing.
* **Multi-threading** to simulate concurrent banking operations.
* **Spinlock-based synchronization** to prevent race conditions and ensure data integrity.

**Problem Analysis**

Concurrent programming introduces **challenges** in managing multiple threads accessing shared resources. In a banking context, this can lead to data races and inconsistencies, particularly with operations like deposits,funds transfer, withdrawals, PIN changes, and account freezing. Ensuring mutual exclusion and coordinating thread access are crucial to maintaining data integrity and preventing corruption. The project addresses these issues by employing a spinlock synchronization mechanism, which helps to secure and manage concurrent operations effectively on a shared bank account. The following problems could occur without synchronization :

1) **Firstly**, actions involving bank account Balance like deposit,funds transfer,withdrawal,apply service fee can result in wrong final account balance if we use multi threaded implementation

2) **Secondly,** for eg if a thread which is going to deposit money gets preempted and another thread freezes the account in this interval, that deposit thread should no longer deposit and must exit since the account has been frozen, this is not possible without synchronization.

3)**Thirdly**, for eg if a thread which is going to deposit money before transaction it enters current pin but it gets preempted just before depositing, and in this interval if a thread changes the pin so when that withdraw get scheduled again it should not continue as it has entered the previous pin of account not current pin , to fix this there should be synchronization.

**Solution Design**

The project aims to simulate an Synchronized multi-threaded banking system that efficiently handles multiple concurrent operations on a shared bank account. The system is designed to ensure security and reliability through the use of a spinlock synchronization mechanism.

System Components:

1. **Bank Account State:**
   * Concept: The design introduces a conceptual model of a bank account that includes essential attributes such as balance, PIN, and freeze status. This model serves as the central shared resource within the system.
2. **Synchronization Mechanism:**
   * **Spinlock Structure:** A spinlock structure is added into the design to manage access to the bank account data. This mechanism ensures that only one thread can modify the account's state at any given time, providing a critical layer of data protection and consistency.
   * **Atomic Operations**: Design includes atomic check-and-act operations for critical tasks like updating the account balance or changing the account status. This ensures that every operation either completes entirely or does not happen at all, preventing partial updates that could lead to data inconsistencies.
3. **Graphical User Interface (GUI):**
   * **Interaction Design:** The GUI is planned to offer a user-friendly interface.
   * **Banking Operations:**
     + **Transaction Operations:** Deposit and withdrawal capabilities are conceptualized to allow users to manage funds directly through the interface.
     + **Security Operations:** Features to change the account PIN and toggle the account's freeze status are designed to enhance user control and security.
     + **Service Fee Application**: A functionality to automatically apply a service fee
     + **Funds Transfer:** The ability for users to transfer funds between accounts by entering a recipient's bank account number.
4. **Concurrency Handling:**
   * **Multithreading Approach:** The design specifies that each banking operation will be handled by a separate thread. This approach is intended to optimize processing efficiency and responsiveness.
5. **System Reset and Exit:**
   * **Maintenance Features:** The design includes functionalities for restarting the system and safely exiting the application, which are essential for proper resource management and system maintenance.
6. **Transaction Integrity Verification:**
   * **Transaction Logs:** Implement logging mechanisms to record each transaction with This helps in tracking changes and is crucial for audit trails, debugging, and ensuring transparency in operations.

**Implementation & Testing**

The project is implemented in C, leveraging the GTK library for the GUI and pthreads for multi-threading. A spinlock synchronization mechanism is implemented using atomic operations to manage concurrent operations securely.

Implementation Details:

1. **GTK Library:** The GTK library is used to create a graphical interface, allowing users to interact with the project. The library provides various widgets to facilitate user interaction:
   * **Text Boxes:** Used for inputting amounts for deposits,transfers, withdrawals, and PIN changes.
   * **Check Boxes:** Allow users to indicate when information has been entered or to toggle the freezing/unfreezing of the account.
   * **Scrolling Buffer:** Provides a GTK text view and scroll buffer in the main window, where outputs and updates are displayed.
2. **Spinlocks**: A Spinlock struct is used to manage access to shared resources like the bank account. Functions **Spinlock\_init,** **Spinlock\_lock,** and **Spinlock\_unlock** employ atomic operations to ensure mutual exclusion, preventing data races and inconsistencies.
3. **Thread Creation:** Operations are executed as separate threads using **pthread\_create**, allowing concurrent execution of deposits, withdrawals, PIN changes, and more. Spinlock synchronization ensures these threads do not interfere with one another.
4. **Output and Notifications**: Messages are displayed in the scroll buffer whenever a thread locks or unlocks the spinlock, performs a deposit or withdrawal, or updates the account state.
5. **CSS Styling:** The application uses **GtkCssProvider** to define the visual properties of the GTK interface, enhancing user interaction and aesthetic appeal. Styles are applied to various GUI elements such as windows,buttons etc.

6. **Security Features:**

* **PIN Verification:** Every transaction (withdrawal, deposit, funds transfer) requires PIN verification before proceeding, enhancing security by ensuring that operations are performed only by authorized users.
* **Account Freezing:** Provides functionality to freeze and unfreeze accounts, preventing unauthorized transactions when an account is frozen. This feature is crucial for stopping potential fraudulent activities.

7. **Error Handling**:Ensures that all inputs through text boxes (such as amounts and PINs) are validated before processing. This prevents errors related to invalid data formats or unauthorized access attempts.

8. **Thread Scheduling:** usleep(500000) is used in thread runner functions so that each thread is asleep for half a second which allows other threads to be scheduled; this helps in randomizing the order in which threads are executed.

10**. System Timings:** Incorporates timing mechanisms using **gettimeofday** to measure and display the time taken for operations.

11. **Exit and Restart:** Exit button will kill all windows and Restart Button will restart and initialize the program.

**More Specifics Implementation Details:**

**GUI Layout and Widget Management:**

* **GTK Layout Container:** Utilized to organize and position other GUI components such as buttons, text fields, and labels. The gtk\_layout\_new(NULL, NULL) function is used to create a flexible area where widgets can be placed based on specific coordinates, allowing for custom arrangements.
* **Widgets Configuration:**
  + **Window Creation:** gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL) initializes the main application window, setting it as a standalone window with gtk\_window\_set\_title and gtk\_window\_set\_default\_size to specify its title and dimensions.
  + **Label Widgets:** Created using gtk\_label\_new, labels provide descriptive text like "Enter Withdraw Amount:" near input fields, guiding the user on what information is required.
  + **Entry Widgets:** gtk\_entry\_new constructs text entry fields where users input data such as withdrawal amounts and PINs. The gtk\_entry\_set\_max\_length function restricts the number of characters that can be entered, enhancing form validation.
  + **Check Buttons**: Implemented with gtk\_check\_button\_new\_with\_label("Done") to offer interactive elements that users can select to confirm actions or toggle settings, such as completing an input or freezing an account.

**Signal Connections and Event Handling:**

* Event Linking: Uses g\_signal\_connect to bind GUI elements to specific callback functions, ensuring that user actions like closing the window or clicking buttons trigger appropriate responses in the application.

**User Feedback and Dynamic Updates:**

* Text Buffer Management: The GtkTextBuffer associated with a GtkTextView is manipulated to dynamically update text within the GUI. This setup allows for real-time feedback to be displayed to the user, showing the results of their actions or system messages.

**CSS Setup with GTK+:**

* **Initialization:** A GtkCssProvider is initialized to handle CSS styles, linked to the application's screen via gtk\_style\_context\_add\_provider\_for\_screen, ensuring styles are applied globally with high priority.
* **Styling Configuration:** CSS rules are defined in cssData, specifying aesthetics for windows, boxes, and buttons:
  + Windows are styled with a primary blue background and black text.
  + Boxes feature an orange background with large white text.
  + Buttons employ a dynamic gradient background that adjusts on hover, enhancing interactivity.
* **Loading and Application**: CSS is loaded using gtk\_css\_provider\_load\_from\_data and applied across the application, directly influencing the appearance of all GUI elements to maintain a consistent and engaging user interface.

**Testing:**

1. **Concurrent Operations**: Multiple threads perform different banking operations concurrently, testing the spinlock's effectiveness in preventing conflicts and maintaining data integrity.
2. **Boundary Conditions:** Edge cases such as insufficient balance for withdrawals or fees, incorrect PIN entry, and freezing/unfreezing are tested to ensure accurate handling.
3. **GUI Interaction:** The GUI is tested to ensure it responds appropriately to user inputs, correctly triggering banking operations and displaying accurate information. The final balance and elapsed time for each simulation are also displayed in the main window.
4. **Unit Testing:** Each function (e.g., deposit, withdraw) is unit tested to ensure it performs as expected in isolation.
5. **Status Messages:** Messages in the scroll buffer indicate each operation's status, including thread locks, unlocks, deposits, and withdrawals, showing that these operations happen correctly and in the proper order.

**Project Breakdown Structure**

**Week 1: Understanding Spinlocks**

* Implement spinlocks in a shared counter example, testing how they manage access to a shared resource.
* Evaluate the effectiveness of spinlocks in ensuring mutual exclusion and preventing data races.

**Week 2: Understanding GUI Implementation**

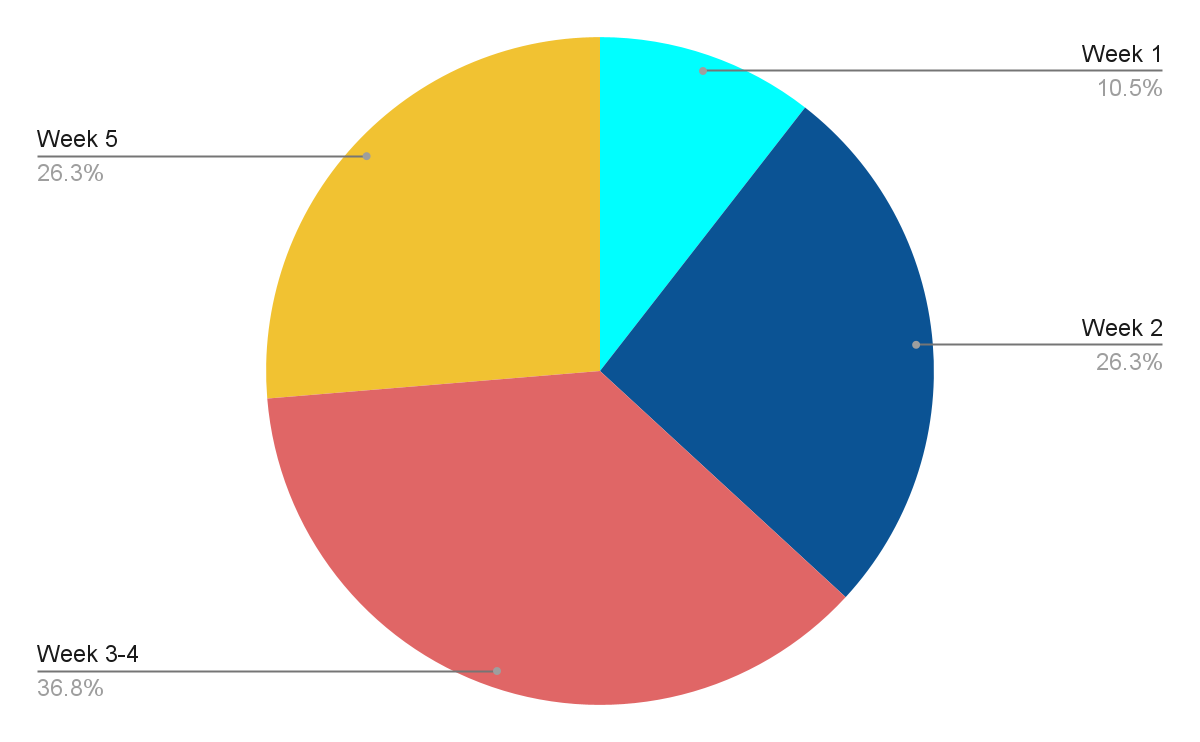
* Research GTK library, download, and set up the necessary environment.
* Watch tutorials to understand how to implement a GUI using the GTK library.

**Weeks 3-4: Banking Operations Implementation**

* Decide on the key banking operations to implement, such as deposits, withdrawals, PIN changes, and freezing/unfreezing.
* Start implementing these operations, integrating them with the GUI and spinlock synchronization.

**Week 5: Testing and Code Cleanup and Documentation and CSS**

1. Test the project thoroughly, including functional tests for each operation, concurrency tests, and edge cases like insufficient balance and incorrect PIN.
2. Writing the project report and documenting the solution.
3. Applied CSS to style window,text,and boxes etc.



**To compile the Program**

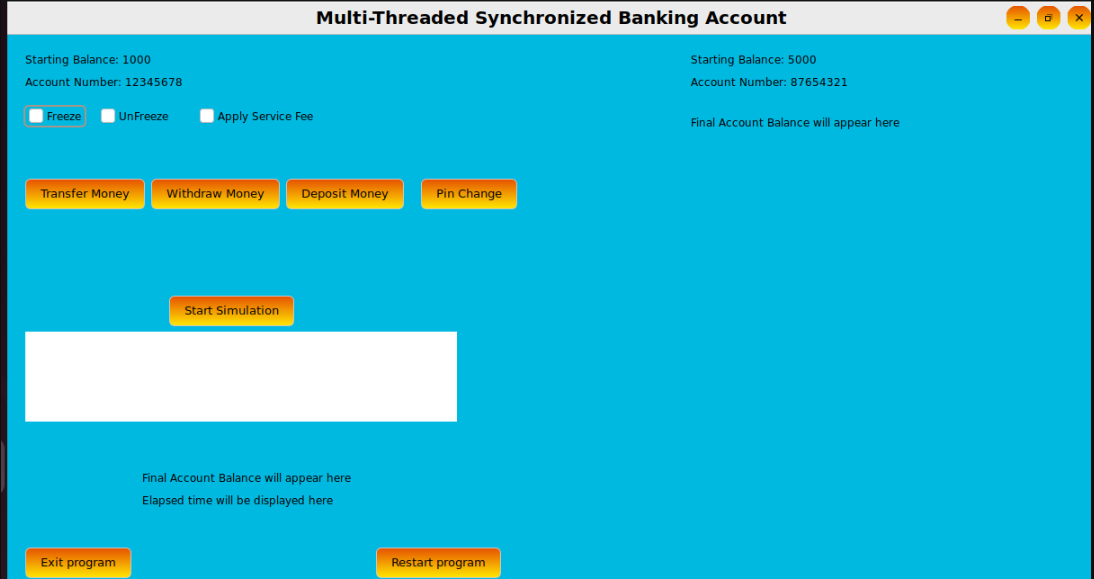
**gcc `pkg-config --cflags gtk+-3.0` -o name name.c `pkg-config --libs gtk+-3.0` -pthread**

**To run the Program**

**./name**

**OUTPUTS:**

**Main Window:**

****

**Start Simulation** runs program

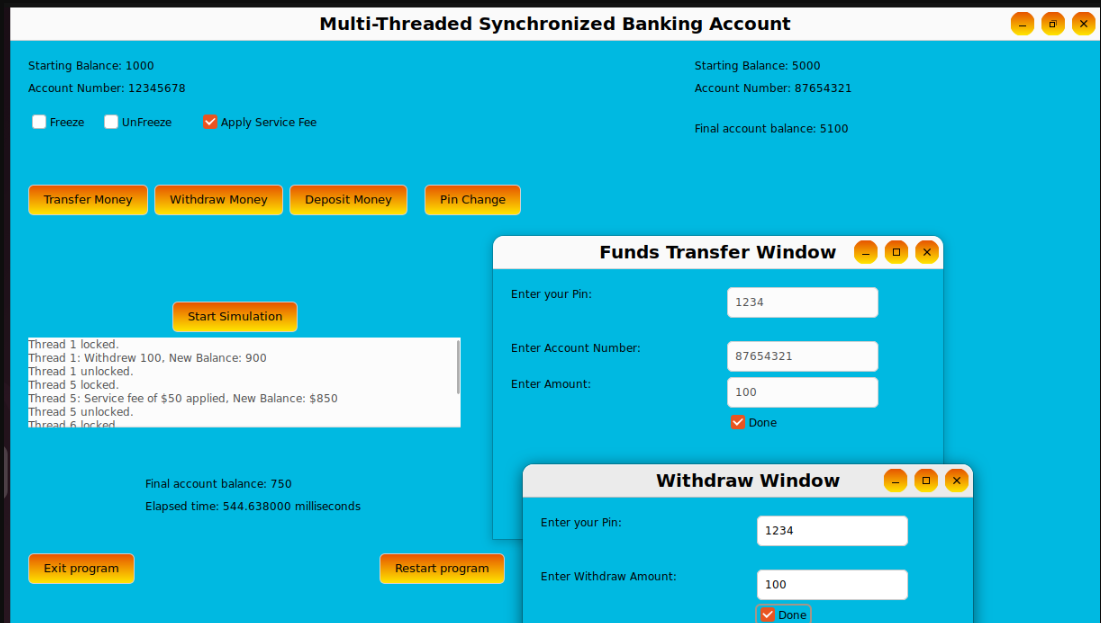
**Restart program** starts it anew

**Exit program** simply exits.

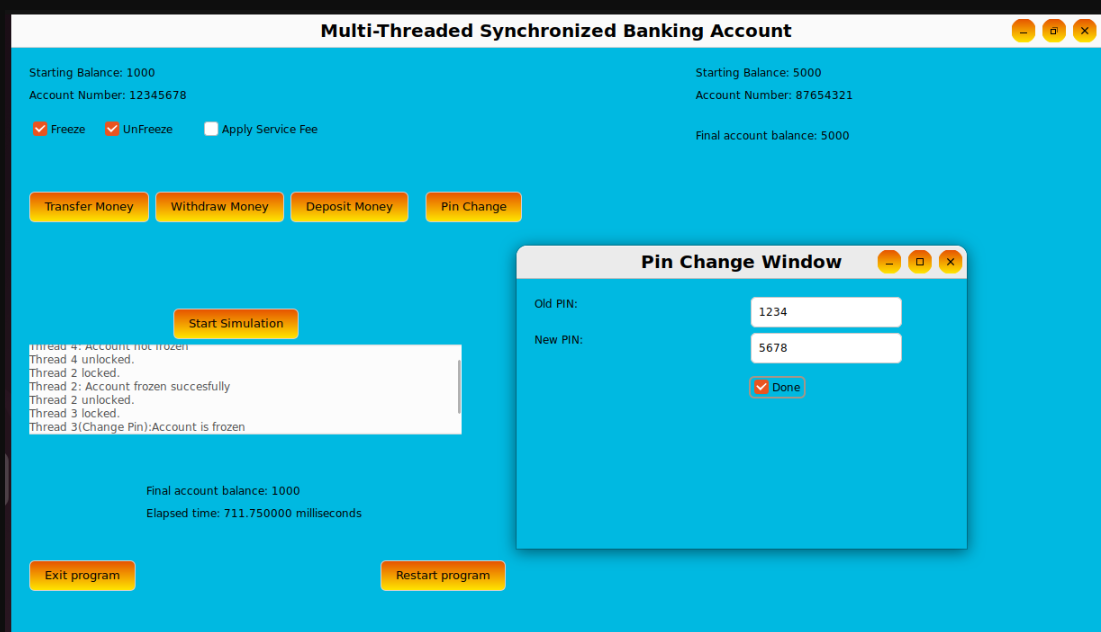
On the **top left corner** ,the information given is of our own account.

On the **top right corner**, the information given is of the other account(to which we can transfer to).

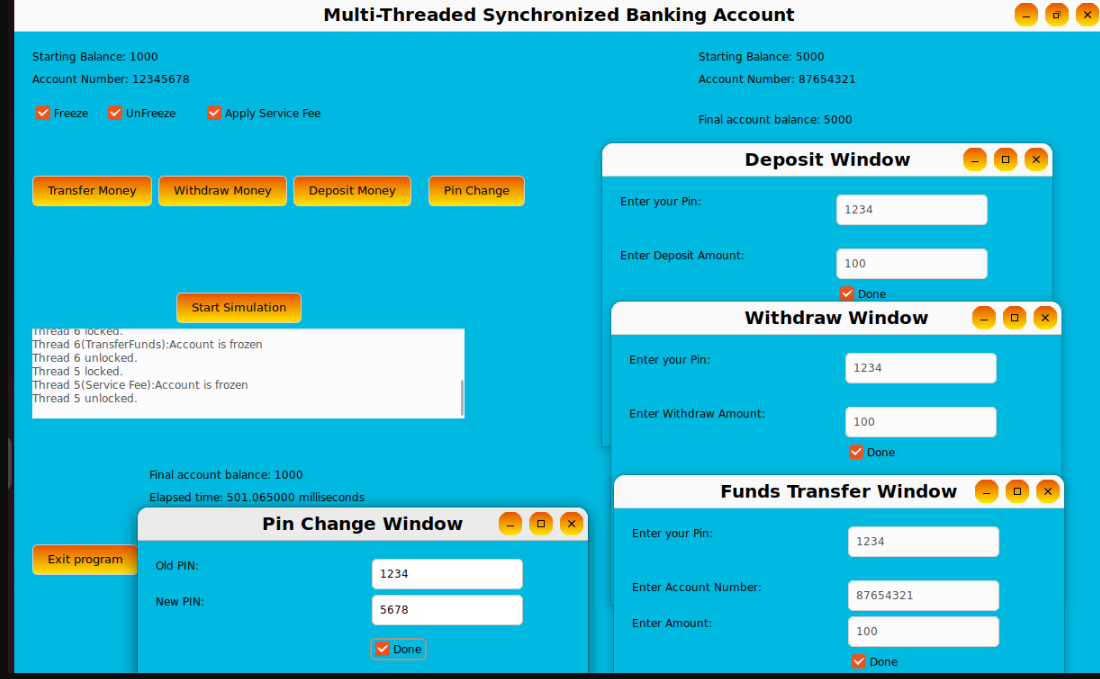
**Pop Up windows on clicking the button**



* **Applied Service Option** which deducted 50 dollars(this is fixed) from current balance
* **Applied withdraw option** which deducted 100 dollars as specified by user input
* **Applied funds transfer** option which deducted 100 dollars as specified and added 100 dollar to the other account shown on the right
* **Final and Starting Balance** is shown to check that transactions have taken place correctly
* After selecting all the options click **Start Simulation Button.**
* **This is reflected in log messages** printed in white textbox below the start simulation

****

* **Used Account Freeze** and **Unfreeze** and **Change Pin option**
* Since unfreeze thread ran first it **couldn't unfreeze** since account was not frozen
* Then account freeze thread ran and **freezes** the account
* Then Change Pin thread ran but it **couldn't change** because account was frozen no operations can take place(Default pin was 1234)
* After selecting all the options click **Start Simulation Button.**
* **This is reflected in log messages** printed in white text box below the start simulation

****

**Text Messages were this: (could not capture all the messages due to limited window size)**

Thread 0 locked.

Thread 0: Deposited 100, New Balance: 1100

Thread 0 unlocked.

Thread 1 locked.

Thread 1: Withdrew 100, New Balance: 1000

Thread 1 unlocked.

Thread 3 locked.

Thread 3: PIN changed successfully.

Thread 3 unlocked.

Thread 4 locked.

Thread 4: Account not frozen

Thread 4 unlocked.

Thread 2 locked.

Thread 2: Account frozen succesfully

Thread 2 unlocked.

Thread 6 locked.

Thread 6(TransferFunds):Account is frozen

Thread 6 unlocked.

Thread 5 locked.

Thread 5(Service Fee):Account is frozen

Thread 5 unlocked.

**Explanations:**

Thread 0 locked means thread 0 acquired the spinlock

Thread 0 unlocked means thread 0 released the spinlock

Thread 6(TransferFunds):Account is frozen means Transfer funds couldn't complete since account was frozen

**Conclusion:**

In conclusion, our project provides a comprehensive approach to implementing a multi--threaded banking account simulator using spinlocks. By addressing the challenges faced when multiple threads access shared data, this project demonstrates how effective spinlock is for tackling race conditions and ensuring data integrity.

Key components Include a model of a bank account using a GUI for user friendly interface, the implementation of atomic operations,thread management techniques, and ensuring basic banking operations such as deposit, withdrawal, PIN changes, account freezing, and transfer of amounts from one account to another are all done without any conflicts.

Furthermore, our project emphasizes security features such as PIN verification along with error handling mechanisms to validate user inputs and prevent unauthorized access.

Overall, the project provides valuable insights into concurrent programming, synchronization mechanisms, and graphical user interface design while showcasing the practical application of spinlock synchronization in a banking context. By simulating real-world banking operations in a multi-threaded environment, the project contributes to the understanding of thread safety and data integrity in Operating Systems

.