Multithreaded Banking System

Introduction

The proposed project aims to develop a multithreaded banking system capable of handling multiple concurrent transactions efficiently. By leveraging the power of multithreading, the system will enhance performance, scalability, and responsiveness.

Implementation Steps

1. **Database Design:** Create tables for accounts, transactions, and users.
2. **Account and Transaction Classes:** Define data structures for accounts and transactions.
3. **Multithreaded Server:** Develop a server to handle multiple client connections and process requests.
4. **Transaction Processing:** Implement deposit, withdrawal, transfer, and interest calculation logic.
5. **User Management:** Implement user registration, login, and authentication.
6. **Security Implementation:** Integrate encryption, authentication, and authorization.
7. **Testing and Debugging:** Conduct thorough testing to ensure system correctness and performance.

Source Code

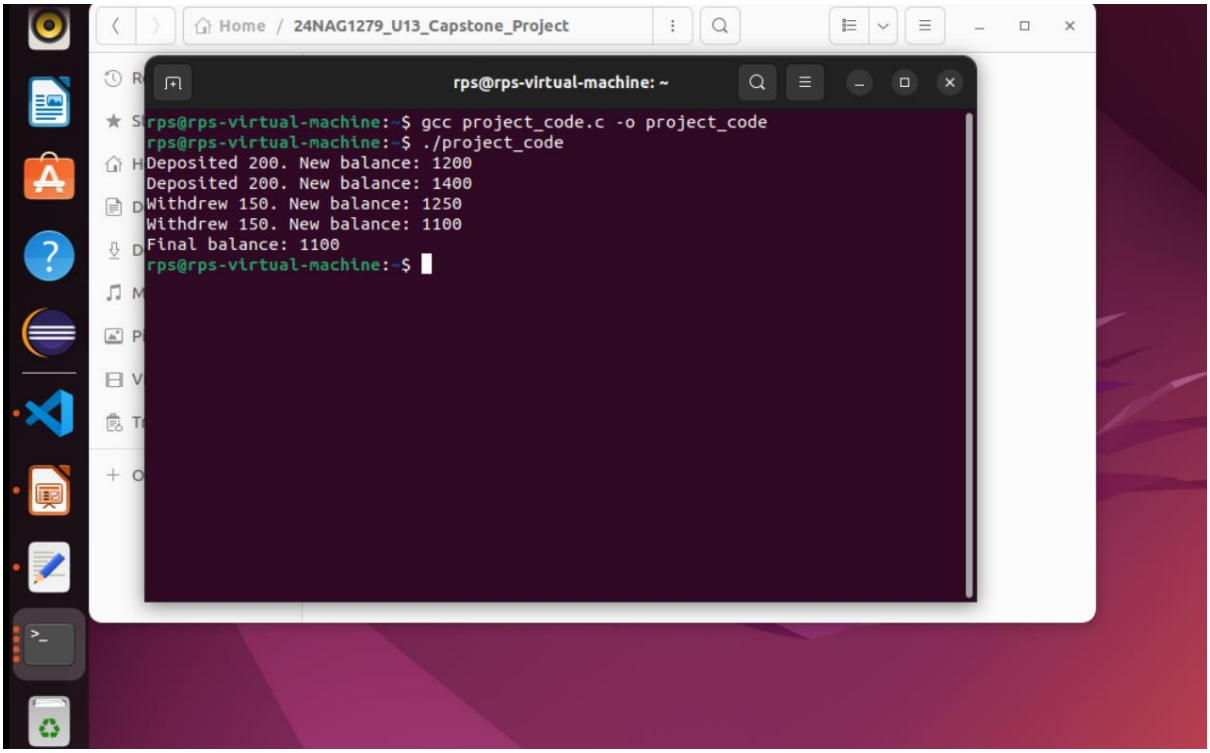
```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>
```

```
// Define a structure to represent a bank account
typedef struct {
    int balance;
    pthread_mutex_t lock; // Mutex to ensure thread-safe access to balance
} BankAccount;

// Function to deposit money into the bank account
void* deposit(void* arg) {
    BankAccount* account = (BankAccount*)arg;
    int amount = 200; // Amount to deposit

    pthread_mutex_lock(&account->lock); // Lock the mutex before modifying balance
    account->balance += amount;
    printf("Deposited %d. New balance: %d\n", amount, account->balance);
    pthread_mutex_unlock(&account->lock); // Unlock the mutex after modifying balance

    pthread_exit(NULL);
}

// Function to withdraw money from the bank account
void* withdraw(void* arg) {
    BankAccount* account = (BankAccount*)arg;
```

```
    int amount = 150; // Amount to withdraw

    pthread_mutex_lock(&account->lock); // Lock the mutex before modifying balance
    if (account->balance >= amount) {
        account->balance -= amount;
        printf("Withdrew %d. New balance: %d\n", amount, account->balance);
    } else {
        printf("Insufficient funds for withdrawal of %d. Current balance: %d\n", amount, account->balance);
    }
    pthread_mutex_unlock(&account->lock); // Unlock the mutex after modifying balance

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[4]; // Array to hold thread identifiers
    BankAccount account;  // Bank account instance

    // Initialize the bank account
    account.balance = 1000; // Starting balance
    pthread_mutex_init(&account.lock, NULL); // Initialize the mutex
```

```
    // Create threads to perform deposit and withdrawal operations
    pthread_create(&threads[0], NULL, deposit, (void*)&account);
    pthread_create(&threads[1], NULL, deposit, (void*)&account);
    pthread_create(&threads[2], NULL, withdraw, (void*)&account);
    pthread_create(&threads[3], NULL, withdraw, (void*)&account);

    // Wait for all threads to complete
    for (int i = 0; i < 4; i++) {
        pthread_join(threads[i], NULL);
    }

    // Destroy the mutex
    pthread_mutex_destroy(&account.lock);

    printf("Final balance: %d\n", account.balance);

    return 0;
}
```

Output:



Future Enhancement

A multithreaded banking system offers a strong foundation, but there's always room for improvement and adaptation to evolving technological landscapes. Here are some potential future enhancements:

 Cloud Computing: Migrate to cloud platforms for scalability, cost-efficiency, and disaster recovery.
 Microservices Architecture: Break down the system into smaller, independently deployable services for better maintainability and scalability.
 Internet of Things (IoT): Integrate IoT devices for enhanced security and user experience.

Conclusion

Multithreaded banking systems offer a substantial advantage in handling the high concurrency demands of the financial industry. By enabling simultaneous processing of multiple transactions, these systems significantly enhance performance, scalability, and responsiveness.