CMSC 621 : Advanced Operating Systems

Centralized Multi-User Concurrent Bank Account Manager

Vangeepuram Saketh Ram
XV45698

Documentation

Project goal :

The project aims to create a centralized bank server to make concurrent transactions from various clients. The server has information about the individual user accounts and their account balances. Multiple clients can connect to the server simultaneously and perform transactions like deposits and withdrawals. The server accepts such concurrent requests successfully and performs transactions by maintaining data consistency.

Server design:

- Server initially stores the user-account information via an input file. The input file Records.txt will have the new user-account information along with the initial balances.
- Server program then will create a socket for communication and listens for any incoming requests.
- Whenever any client sends a transaction to perform, the server will process each transaction and update the balance associated with the requested account based on the transaction type. Similarly, all the transactions from that particular client are processed individually.
- For each client, a new thread is created and the thread executes all the transactions related to that client.
- Upon executing transactions from one client, it again listens on the socket for any other incoming connections for clients to accept. Therefore, each client is handled one by one.
- The server also runs a program that calculates interest on account balances for all the accounts and then adds the interest to the existing balance. Currently, the server adds 0.01% interest for every 120 transactions(assuming it takes 60 seconds to perform 120 transactions). We can further modify this by considering proper timestamps.
- All the connection messages and error messages are logged in logfile.txt file in the project folder.

Client design:

- Client program initially creates a socket for communicating with the server.
- Once connected with the server, it then sends individual transactions to the server for execution.

- The required transactions can be provided in the Transactions.txt file through which the client program reads and sends it across the server.
- Though the input file has multiple transactions, the client will send one transaction at a time for execution.

Design Trade-offs:

- I can make use of some programming techniques(standard patterns) to add logic to notify server/client if there is any data on the socket instead of waiting infinitely.

How is synchronization handled?

- Though multiple clients make concurrent requests to the server, the server will lock the critical section while executing a client's request and will only unlock after the transaction is complete.
- This will ensure data consistency and secure the critical section code by not allowing other processes to access the code while some other client is using it. (Here, clients can be ATMs or individual customers, bank employees etc).

Possible improvements in the application:

- The code can further be modularized to reduce redundancy and increase modularity and usability.
- More functionalities like checking account balance, obtaining previous transaction history, and generating statements can be added.
- Can deploy multiple instances of server in different locations based on the user location diversity and can route the request to appropriate server based on client's location.
- We can further add GUI to send and receive messages and make user interactive interfaces to perform transactions.

How to compile and run?
Steps :

1. Run "make clean" - to clean all the object files from the project folder.
2. Run "make" - make command will compile server.cpp and client.cpp files to generate server and client executables.
3. Use "./server" to run the server program and use "./client" in a different window will run the client.
4. To run multiple clients, use the "make dump" command which will run the client 100 times.