HOW THE PROJECT WORKS

HOW TO COMPILE AND RUN THE CODE

DESIGN TRADE OFF ASSUMED AND MADE

OUTPUT PRODUCED

**HOW THE PROJECT WORKS**

The project aim is to create a bank server client account management system. Here, I am considering a world of animals who have an account in the bank named “OLIKKAL BANK” and wants to do transactions according to their needs.

There are 10 animals who have created an account in the bank and depending upon their need, they can do either a deposit or a withdrawal on their respective savings account.

Thus the bank management system has two parts: a server part and a client part.

The server part contains the records of the animal’s saving account details and their respective amount in the account. The client part takes the transactions part which makes all the changes in the records.

OLIKKAL BANK SERVER:

Step 1: In the server, first I would take the Records.txt file that is created with the details of the animals and the amounts respectively. And check if I am able to open the file or not.

Step 2: Now, I will calculate the number of records in the database of the bank. I take each line from the record, tokenize it by finding the space between each words and store them into an array.

Step 3: I try to establish a connection to the clients that are trying to connect to the server using TCP protocol and socket stream through socket programming. Functions of socket programming like socket(), bind(), listen(), accept() are implemented for connecting and communicating with the client. At implementation of the function, it is checked and the corresponding outputs are returned.

Step 4: Since the server has to be always on and waiting for the client to make a connection to the server, I put the accept() function in the while loop.

Step 5: Once the connection is established to a client, in order to prevent a race condition from happening which leads to starvation, I implemented a pthread mutex lock. This ensures that once a client has established a connection with the server, no other client would be able to connect to the server till the current client completes all the transactions. This puts the Records.txt file in a lock, because it will be available only to that current client which has established a connection to the server.

Step 6: Inside the accept() block, I use a function named, transactionHandler which creates a thread for each of the transaction in the client, after the client makes a connection to the server. The data sent from the client is read and checked for errors. I then put a start\_time element to track the time taken by the client for each transaction. Once the data from the client is received, it is tokenized by checking for the white space. A transaction counter is kept which keeps on running with each new transaction.

Step 7: Now, I calculate the interest amount by checking the timestamp of the received data from the client. If it has waited for more than 5, then the interest is added to its account.

Step 8: The account id that is received from the client is compared with the account id from the records. If present, I’ll will check what kind of transaction is being made, whether it is a deposit or withdraw. If not present in the database, an error is thrown

Step 9: Once the kind transaction of transaction is check, I’ll proceed to write back the final amount to the file Records.txt, thereby updating it. A confirmation of the data processed is sent to the client.

Step 10: Insufficient amount, Invalid amount and invalid account number are also handled throughout.

Step 11: The time taken to end the transaction is also obtained at this point and the difference between the initial and final time is caliculated.

OLIKKAL\_BANK\_CLIENT:

Step 1: First I start off by asking for the host address, port number, timestamp and the transaction file name

Step 2: Through the usage of socket programming, by TCP and socket stream, I try to establish a connection with the server. The usage of socket(), connect() are used here. These are checked and verified as well.

Step 3: With the usage of the host address received from the command line, I make a connection to the server.

Step 4: The data in the Transaction.txt file is taken line by line. Each line is taken and tokenized and sent to the server.

Step 5: The timestamp is taken here and a loop is run so that the initial time will match the timestamp received from the Transaction.txt file.

Step 6: Upon successful tokenization, the data is sent to server for processing.

Step 7: An acknowledgment is received from the server upon successful processing of the transaction. And the client exits.

**HOW TO RUN AND COMPILE THE CODE:**

COMPILE CODE:

$root g++ olikkal\_bank\_server.cpp -o server -lm -lpthread

$root g++ olikkal\_bank\_client.cpp -o client

This would generate two object files, server and client respectively.

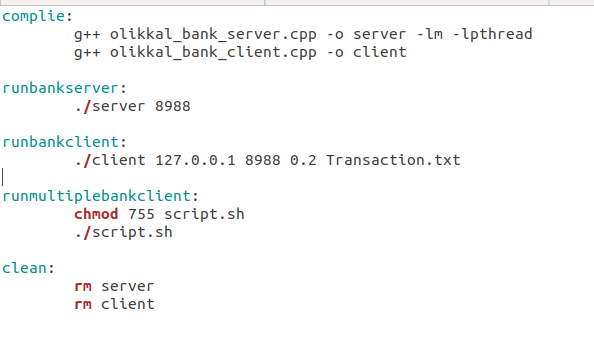
RUN CODE:

$./server 8988

$./client 127.0.0.1 8988 0.2 Transactions.txt

I have created a MakeFile for the same. Below mentioned is the make file.

It takes compile, run bank server, run bank client, run multiple bank client and clean commands.



**DESIGN TRADE OFFS ASSUMED AND MADE:**

I) One of the major trade off that I made in implementing this bank server client model is that when there are multiple clients that are getting connected to the server, if the maximum value exceed more than the length of the transaction line, then it would overflow to the next line while flushing the data back into the Records.txt file.

II) While implementing the interest, I assumed that the rate of interest to be 5% and made a built-in statement that runs through the loop enough times and then writes back to the Records.txt file

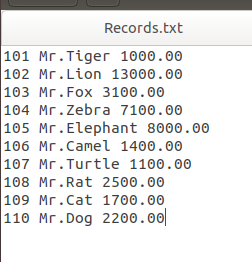
III) Assuming that the bank balance does not go below 0 and there are no negative values for the amount I proceeded further.

IV) Another assumption that I made here is that there are only two types of transactions, that is, deposit and withdrawal

**OUTPUT PRODUCED**:

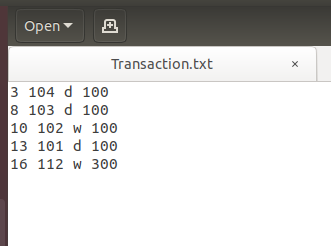
RECORDS.txt

Below shown is the RECORDS.txt file that contains the records of all the animals who made an account in OLIKKAL BANK. The file consists of an account number, animal name and the amount.

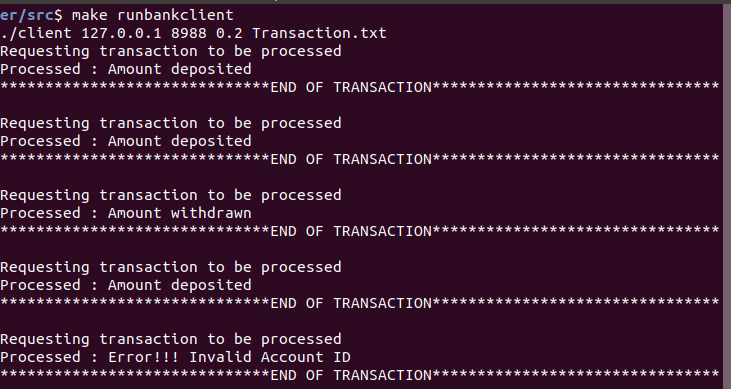
****

TRANSACTIONS.txt

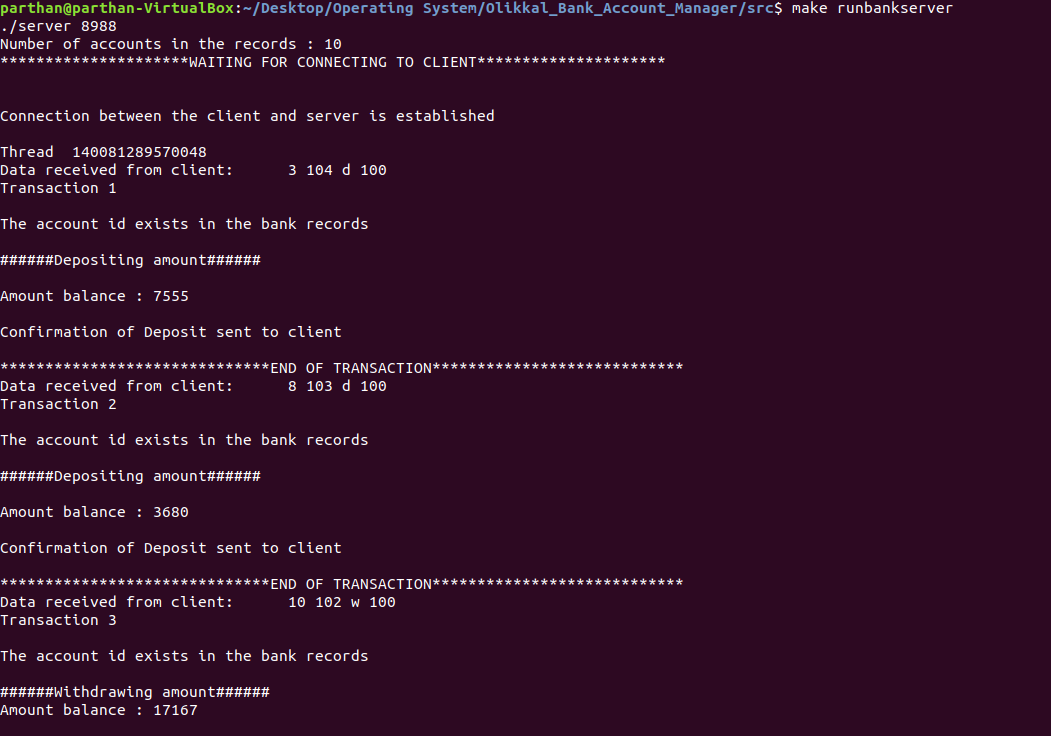
Below shown is the TRANSACTIONS.txt file that contains the transactions made by different animals. It contains the timestamp, account number, the kind of transaction (whether it is a deposit or a withdrawal) and the amount.

****

Output Screen of client

****

Output Screen of Server

****