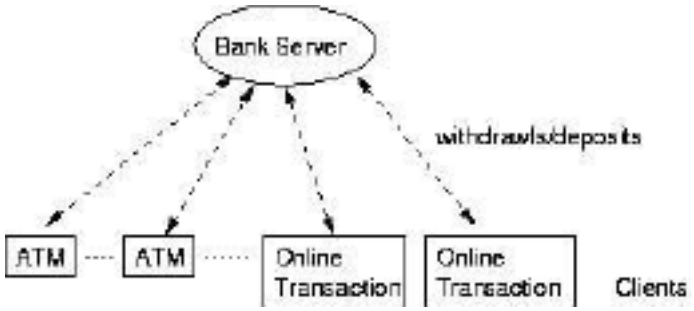
Assignment 1

In this programming assignment you are to implement a **Centralized Multi-User Concurrent Bank Account Manager**. The system has two important components.

- **Bank Server:** The server program that services online requests for account manipulations and maintains all customer records correctly.
- **Clients:** Customers are clients of the bank server and use its services to update bank accounts. The operations that can be performed on an account are: **withdrawal** of an amount from an account and **deposit** of an amount into an account. Additionally, the bank server can have it's own service that periodically deposits an interest amount to each account based on some fixed rate.

A pictorial representation of the system is as shown in the figure below.



The components of the system and their functionalities to be implemented are as follows:

Server

The server receives queries from customers (ATM clients, Clients/Billing agencies performing online transactions etc.) for operations on accounts. The server should have the following functionalities:

- Should be able to accept multiple concurrent customer requests **(i.e., must be multi-threaded)**
- It should provide **locking/protection for access to an account records during shared access** (i.e., a user might be depositing money into his account and at the same time an online billing agent might be withdrawing money from the same account). Such cases need to be correctly handled by protecting variables in the critical section.
- Maintain correctness of records at each record, (i.e., allow withdrawal from an account only if it has sufficient funds etc.)
- To create a set of records at the server initially, you can use an input file that the server reads and creates account information. You can generate your own input file. For testing, we will use our own file. However, name the file **Records.txt**.

An example input file format could be:

101 Peter 16000

102 John 1200
103 Gambo 11000
.

The format for each line being: ***account number, name, balance amount*** (space separated)

Client

A client issues requests to the server from a transaction based on account numbers. Client functionality:

- **Issue withdrawal** or deposit requests.
- For ease of testing and to make experiments bigger, the clients can issue requests at fixed time intervals. A client can read an input file for transaction information and perform those tasks accordingly. For example, following is a format you can use to generate input for each client and use it in experiments. Create your own file. However, call it **Transactions.txt**.

  10 01 w 200
  25 01 d 300
  26 05 d 150
  .

  Each line of the input file is a transaction request to be issued by the client and has the following format: ***timestamp, account number, transaction type(withdrawal/deposit), amount*** (space separated)
- Should receive status of transaction from server and print or log status for each.

The purpose of this assignment is to get you familiarized with *sockets, processes, threads and synchronization.*

You can be creative with this project (like add create new online accounts, which means add new functionalities, other than mentioned above, at both client and server etc.).

Constraints on the Implementation

- You must use C++ for implementing this project (therefore, your source files should have extensions of .cc or .cpp. You are welcome to use the STL classes). You must test your code on a linux machine/machines. Testing it on a Windows and/or Unix (MAC) machine is not acceptable.
- You must create a Makefile with at least two targets.
  - clean: ``make clean'' should clean all the object files in the directory.
  - compile: ``make compile'' should compile the code and create two linux executables, **server** and **client**.

Some things to keep in mind

- The server should be able to handle multiple transaction requests at the same time. This could be easily done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system
- The server should provide protection for shared simultaneous access to the same record using semaphores or locks etc
- No GUIs are required. Simple command line interfaces are fine
- **This is an individual project. Go group collaboration is allowed**

Evaluation and Measurement

Correctness

Demonstrate that your system works correctly according to requirements stated in the description of the system.

- Show that the server preserves correctness of transactions by deducting money only when available, is multi-threaded and accepts requests concurrently.
- Show use of locks/semaphores that protects simultaneous access to the same account. i.e., design a test experiment(s) to demonstrate this using logs or message display during events.

Scalability

- Use a set of periodic requests at each client (e.g., each sending a request every 2 secs) and vary the number of clients connected to the server. Measure the average time to complete each transaction for each client. Plot a graph to show average time to complete each transaction as number of clients are increased (say from 1 to 100).
- The same experiment can be repeated by fixing the number for clients (to say 25) and then varying the request rate. i.e., one request every 0.1, 0.2, ...1 secs etc. Measure and plot the average time to complete each transaction as the request rate is varied.
- These are guidelines only, so be creative in what can be evaluated and measured as part of your experiments to test the system.

What you will submit

Each program must work correctly and be **documented**. You should submit a **zip** of two directories.

- **Directory 1** (name: **src**): The first directory should have all your source code and the Makefile. Additionally, the directory should have a separate (**typed**) document of approximately two pages describing the overall program design, a description of ``how it works'' (how to compile and run the code), and design tradeoffs considered and made. Describe clearly how multiple threads and synchronization are handled. Also describe possible improvements and extensions to your program (and sketch how they might be made). The directory should also have a copy of the output generated by running your program. When the server receives data, have your program print a message ``**data received from client s''** . When a client is sent data by the server, have your program print messages for the data being sent by the server and for the data being received by the client.

- **Directory 2 (**Name: **Results):** The directory should have a document which describes the performance results (graphs that are discussed above).

The zip of the two directories should be submitted using **Blackboard** as described below.

Grading

- Program Listing
    - Works correctly – 50%
    - In-line documentation – 15%
- Design Document
    - Quality of design – 15%
    - Understandability of doc – 10%
- Thoroughness of evaluation – 10%

Submission

Assignment given out: 09/15/2021
Assignment due: 10/08/2021 (Midnight)

Submission method

We will be using blackboard to submit your assignment. Create a .zip file for all the files. The blackboard submission site will be made available soon.