**Centralized Multi-User Concurrent Bank Account Manager Documentation**

Name: Himanshu Londhe

Student id: SZ46852

**How it works**

**Banking app manager using sockets,threads and locks**

* The banking app works with one server and multiple clients. The server maintains a record of existing clients and the clients makes operations like withdrawal and deposit to the account. All the requests to the client are handled concurrently. The server then performs the operations and makes the changes in the memory.

**Steps:**

* The server, upon starting reads the records file to get the info about all the customers.
* The server and client are connected over the local network using sockets.
* We are using a tcp connection for establishing connections.
* Once the connection is established, the client sends the transactions over using the socket send() call line by line.
* On the server side, for every transaction received a new thread is created to handle the transaction.
* The server implements locks (mutex) to lock resources as the transaction threads are accessing the update or withdrawal function concurrently.
* For every transaction the server also uses a sleep() to accomplish correctness.
* Depending on the type of the transaction, i.e. ‘W’ or ‘D’ (withdrawal/Deposit) it performs the action and updates the balance in memory.
* Additionally the server deposits interest in each account by 20% after a time interval of 5 seconds.
* A record of all the transactions done are logged in a logger file.

**How to Run**

* To execute, run the make file
* Type **./server** in one terminal and **./client once** or  **./script.sh** for running client 50 times**.**

**Design Tradeoffs**

* Multiple clients can run at the same time using locks but there is no way to get the transactions done in order they come.
* We have used vectors to store the transaction info sent by the client in the server.  
  Vectors are easy to implement but can cause memory overhead