***CMSC 621: Advanced Operating Systems***

***Report for PROJECT 3***

In this project we are asked to design and implement a simple, distributed banking service that permits multiple concurrent operations and exhibits simple fault tolerance. In order to implement this I have created 3 programs as below:

* Client: This program accepts the transaction details from the user and sends it to the frontend server. Once the transaction gets processed, the output is passed to the client and it in turn displays the output. In order to compile this I give the port number and the server name as the localhost.
* Frontend server: This program accepts the connections from the client and sends the transaction details to the 3 backend servers. In turn, backend servers process the requests and send the output to this program. As a result, frontend sends the responses to client.
* Backend server: As mentioned in the project description, I have created 3 backend servers which gets all the transaction details from the frontend server and performs the required actions.

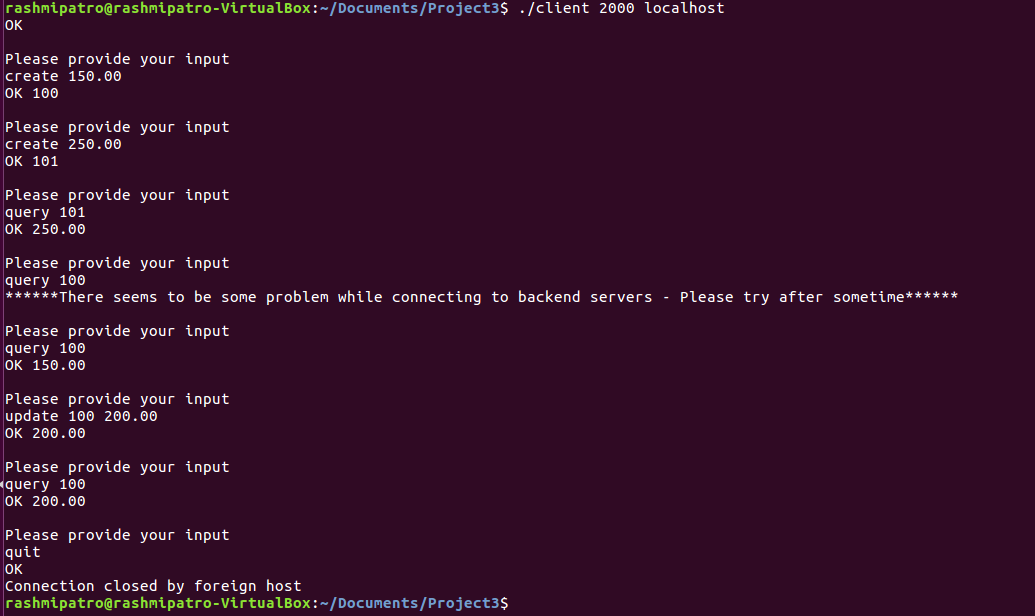
First of all, all the backend servers are started with the port number as specified in the readme file. Then, frontend server is started which again waits for the incoming connections from the client. Finally, client is started and it asks to provide the transaction details. Once user provides the first transaction detail as “CREATE 100.00”, client sends these details to frontend server.

Frontend server then accepts the connections and sends this connection details to backend servers. A separate thread is created for each client connections. All the backend servers send a “VOTE” status and simultaneously frontend server counts the number of active backend servers. Once the frontend server gets all the backend servers active status, it asks all of them to commit the transaction. As mentioned in the project description, all the transactions should be in uppercase, so frontend converts them and sends it to the backend servers. Also, it checks if the transaction is “QUIT”. If it is, then it sends an “OK” message to the client and the client socket is closed.

Backend servers handle all the transactions for “CREATE”, “UPDATE” and “QUERY”. There are separate functions for each of these transactions. There is an array of structures created which stores all the details regarding the account number and the amount.

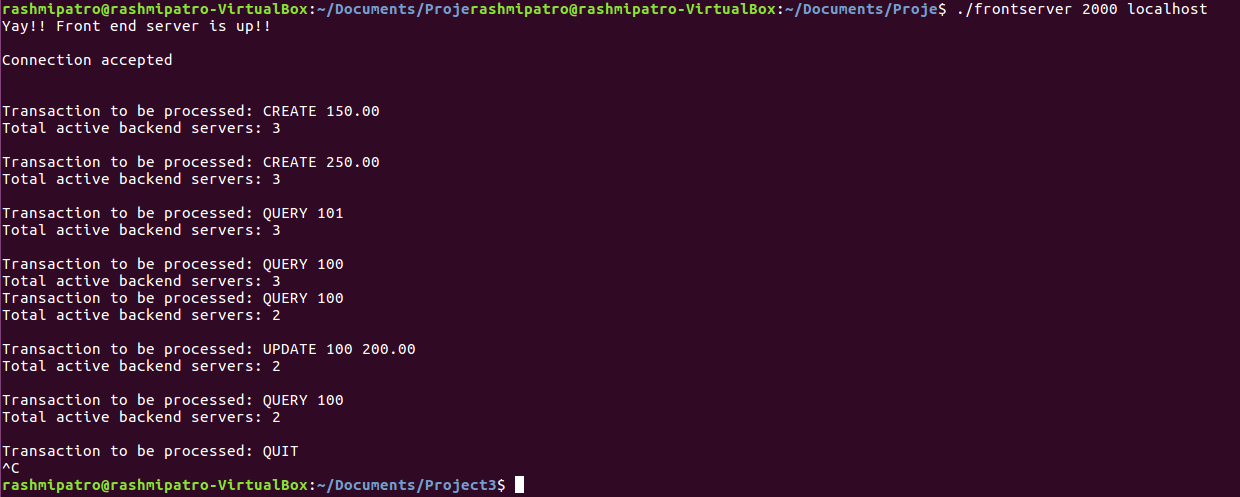
Below is the sample output:

**Client:** When client connects to the frontend server, it asks to provide the input. Each transaction needs to be completed before starting another transaction.

****

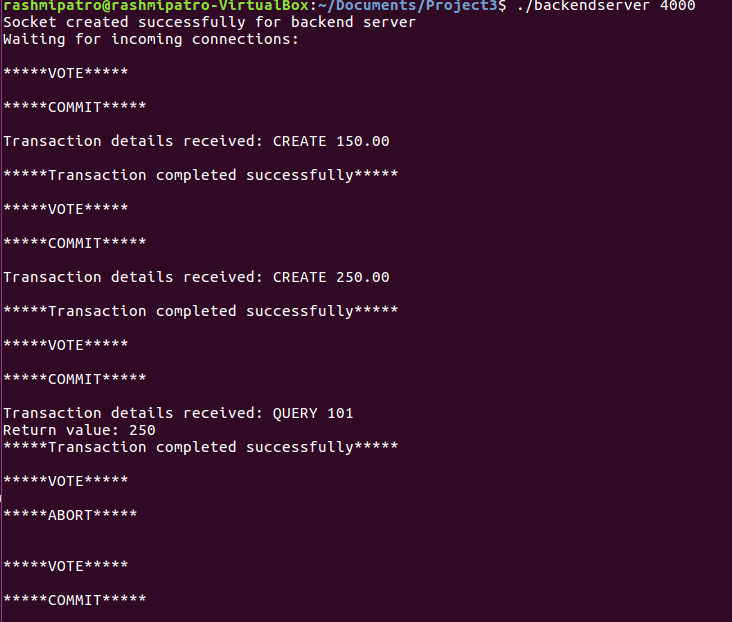
**Frontend:** It accepts the transaction details from client. It then asks the backend servers to vote and simultaneously counts the number of active backend servers. After that it asks all the active backend servers to commit and sends the transaction details as well. Once the transaction is carried out by backend servers, it receives a reply message which in turn is forwarded to the client.

While handling the transactions, one of the backend servers was shut down. This is the reason the count of the number of active backend servers are changed from “3” to “2”.

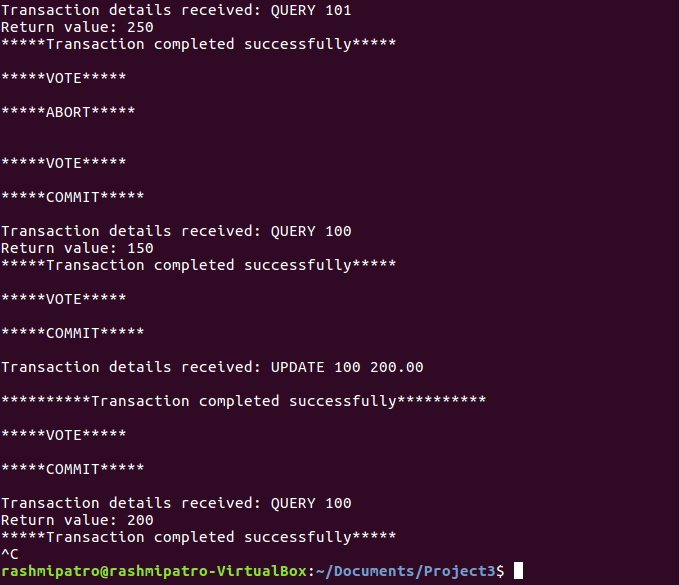
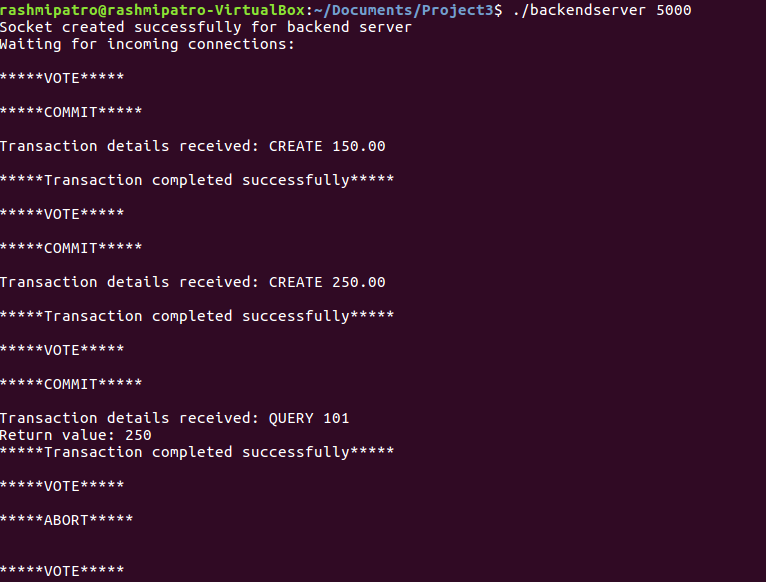


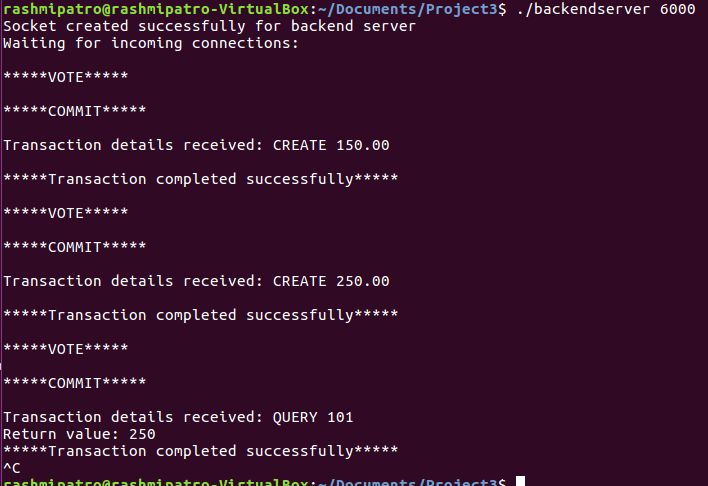
**Backend:** Initially there are 3 backend servers which gets connected to the frontend server. They process the transactions and then send the message.

**Backend server1:**





**Backend server2:**

**Backend server3:**

**Locking mechanism:** In the backend server, for each transaction “CREATE”, “UPDATE” and “QUERY”, locking mechanism is used. For each transaction, mutex is used which locks it, performs the required action and then unlocks for the other process to gain the mutex lock on the particular account.