

**Introduction:**

To implement a distributed banking service that permits multiple concurrent operations and exhibits simple fault tolerance. For this project, three backend servers and one frontend server has been implemented where two phase commit protocol is used for checking the consistency among the three backend servers. The fault tolerance is also implemented by creating the replicas of each and every transaction data in all the three servers. The entire project was implemented using C++ as the programming language in the Linux operating system.

**Execution:*****Make compile***

***./frontend\_server***

***./main\_client 8023 "127.0.0.1"***

***Python3 backend\_pycode.py 3***

where ,

8023 – port number assigned to frontend and the backend servers

127.0.0.1 – is the ip address

3- to signify the number of backend process to be executed.

***IMPORTANT NOTE: \* Assign different port number if it doesn't for second time execution of the program\****

**Implementation:****Client side:**

The client side is where the inputs are given to the frontend server. The inputs are given as queries such as "CREATE", "UPDATE", "QUERY" and "QUIT". These queries are stored in the structure as data. The structure object is passed through the socket to the frontend server. After the required operation for the given command, the client receives the output from the frontend server and prints it out to the user. The transactions maintains atomicity and consistency among servers.

**Servers:**

The server are of two types for this project , the first one is the frontend server and the second is the three backend servers as three processes.

**Front end Servers:**

The front end server is the one where the client's data is received and sent to the three backend servers. The front end server checks the command i.e. whether the command is "CREATE", "UPDATE", "QUERY" and "QUIT". After checking the command , it forwards the received data for the command to the backend servers. The frontend server receives data from the backend servers after all the manipulations and storage. The received data is then again forwarded to the main client who gave the query. The data is

forwarded from the frontend server as a struct object and receives the data in the struct object itself. Structure array object is used as storage for all the input data received in both the frontend and the backend servers. Concurrent transaction from the client is handled using threads.

### **Backend Servers:**

Here we keep the backend servers as three clients connected to the main frontend server. The backend server receives the data as struct object from the frontend server. Then stores the received data from the frontend in a struct object array and sends back the manipulated data to the frontend server.

### **Correctness:**

For the command CREATE from the client , the frontend server allocates a space for the received value in the structure object array .For this CREATE command, a new account number is created and then the whole data is passed to the backend servers for the purpose of storage

For the command UPDATE, the frontend sever checks the struct object array for the account number sent by the client with the account numbers stored in the struct array , if those two matches , the data is then forwarded to the backend servers. The backend servers updates the data in its database ( which is nothing but the structure array). The updated data is then forwarded to the frontend and also to the client .

For the command QUERY, the frontend server checks for the account number sent by the client , if its in the frontend server struct array , it forwards the data to the backend servers and gets the amount in the account to the client through frontend server. If the data is not found , it shows an error like “ERR , ACCT NOT FOUND”.

For the command QUIT, it exits from all the other three backend servers and shutdown the connection.

### **Two phase commit protocol implementation:**

The two phase commit protocol implementation is done by sending the transaction details to the 3 backend servers. The backend servers approve the transaction by either sending “ **OK ,commit**” for the correct transaction and sends “**ERR acct not found**” for the improper ones. I have set above messages as a way of voting by the backend servers to the frontend to approve and send the transaction to the client.

### **Fault tolerance:**

Fault tolerance is one of the significant element of this project where the frontend server and other non-crashed/ non faulty backend servers should continue working despite any crashes of one or two backend servers. This methodology was implemented using getsockopt command that helps in tolerating the fault of one server when it is crashed and uses the other two servers. To check the status of the backend server every time a status array has been used in the code which sends the data to the backend server only when the status of that backend server is 1 if zero then we can know that its faulty .

### **Implementation issues faced:**

The major implementation issue that I faced was to maintain consistency between the client , frontend and the backend servers. The other implementation challenge was to use setsockopt with timeouts which

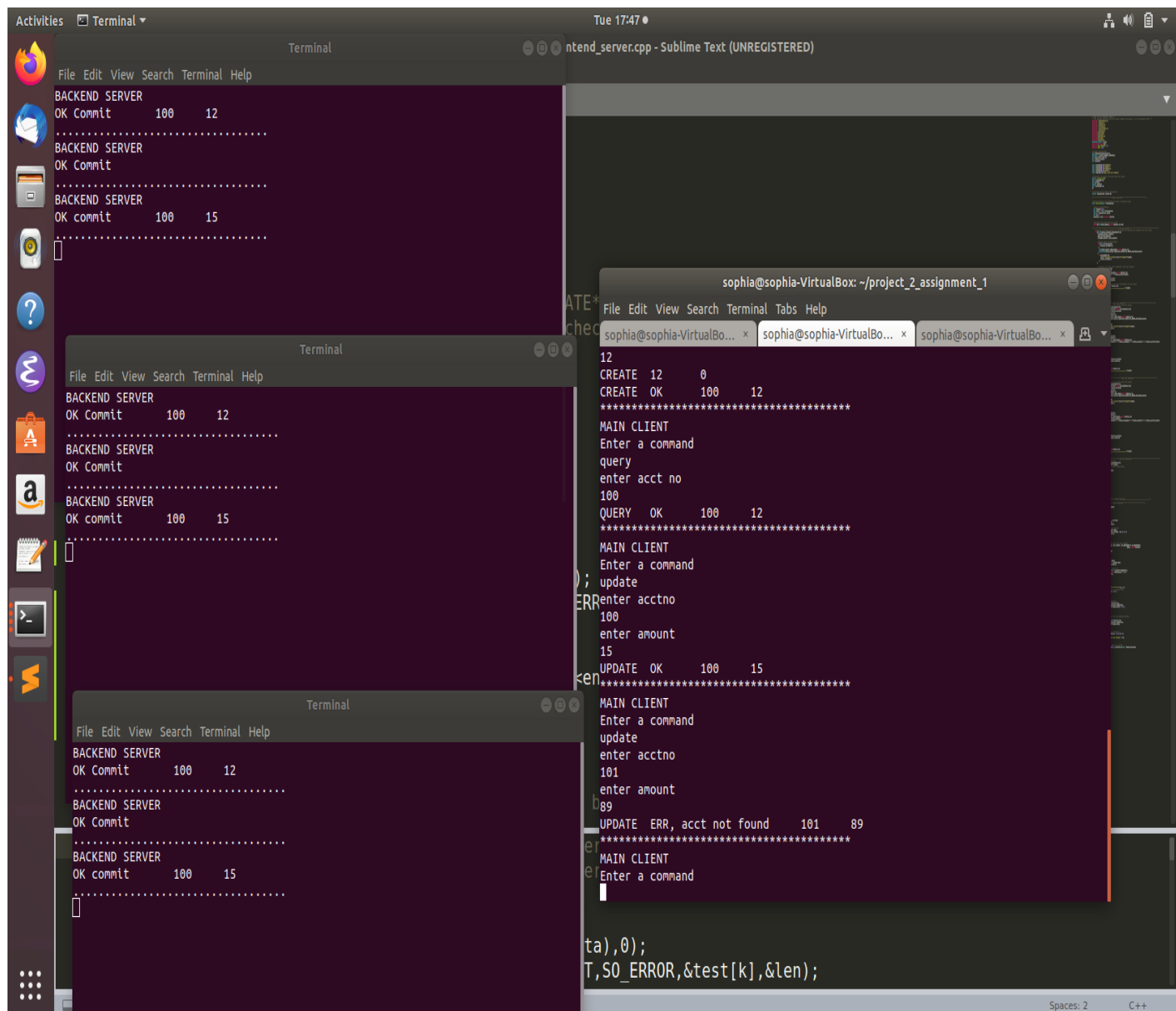
quite didn't work out for me. Hence I have used the getsockopt instead of setsockopt to maintain the backend server when one or two is crashed.

### Output:

#### Transaction phase among the client, frontend and the backend servers.

The transaction between the client , frontend and backend servers is shown.

Figure 1:



```
File Edit View Search Terminal Help
BACKEND SERVER
OK Commit 100 12
.....
BACKEND SERVER
OK Commit
.....
BACKEND SERVER
OK commit 100 15
.....

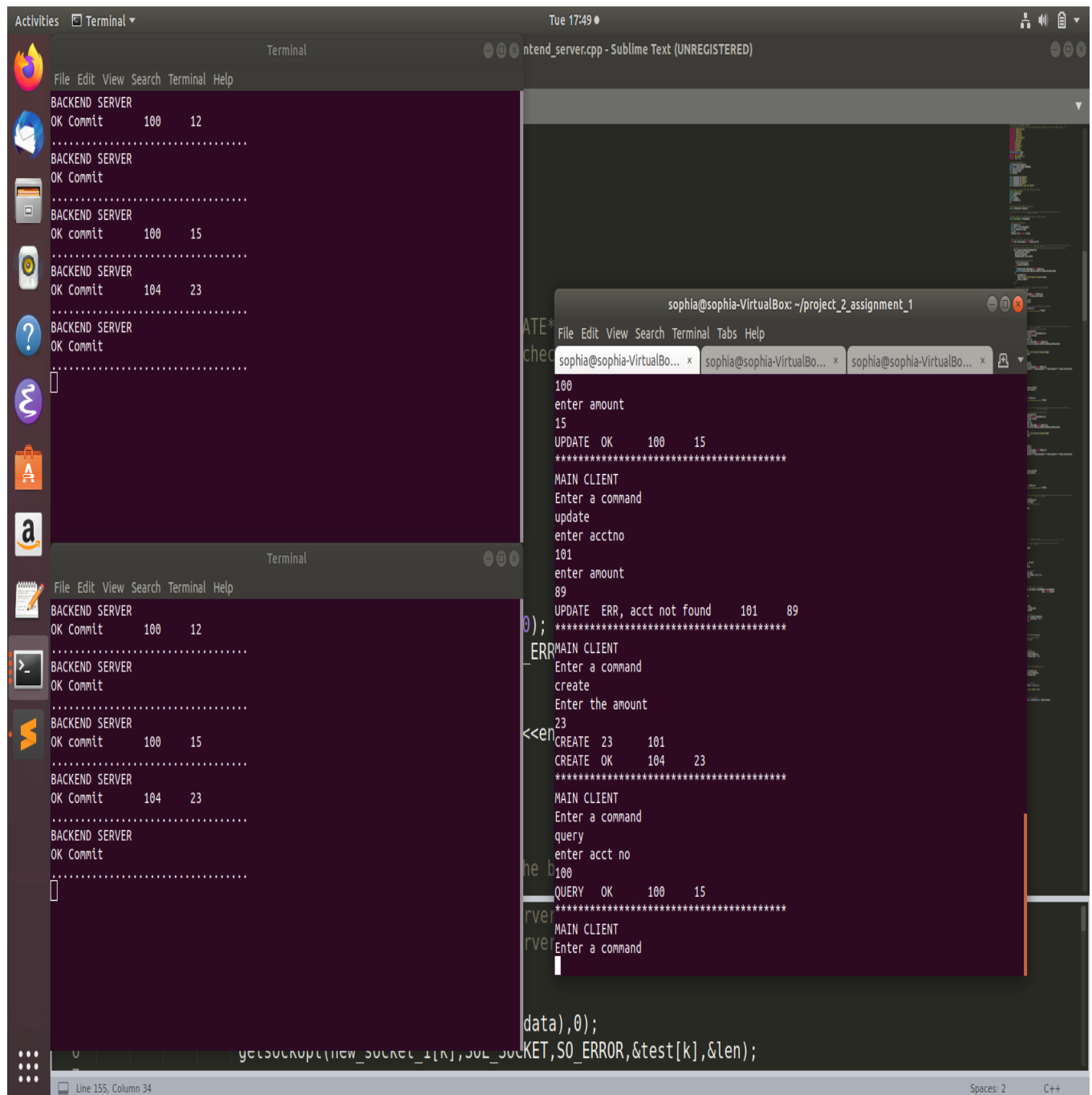
File Edit View Search Terminal Help
BACKEND SERVER
OK Commit 100 12
.....
BACKEND SERVER
OK Commit
.....
BACKEND SERVER
OK commit 100 15
.....

File Edit View Search Terminal Help
BACKEND SERVER
OK Commit 100 12
.....
BACKEND SERVER
OK Commit
.....
BACKEND SERVER
OK commit 100 15
.....

sophia@sophia-VirtualBox: ~/project_2_assignment_1
File Edit View Search Terminal Tabs Help
sophia@sophia-VirtualBo... x sophia@sophia-VirtualBo... x sophia@sophia-VirtualBo... x
12
CREATE 12 0
CREATE OK 100 12
*****
MAIN CLIENT
Enter a command
query
enter acct no
100
QUERY OK 100 12
*****
MAIN CLIENT
Enter a command
update
enter acctno
100
enter amount
15
UPDATE OK 100 15
*****
MAIN CLIENT
Enter a command
update
enter acctno
101
enter amount
89
UPDATE ERR, acct not found 101 89
*****
MAIN CLIENT
Enter a command
ta),0);
T,SO_ERROR,&test[k],&len);
```

The fault tolerant phase of the servers is shown below

Figure 2:



Indication of backend server crashed displayed in the frontend server is shown below. Figure 3

```

Activities  Terminal
File Edit View Search Terminal Help
BACKEND SERVER
OK Commit 100 12
.....
BACKEND SERVER
OK Commit
.....
BACKEND SERVER
OK Commit 100 15
.....
BACKEND SERVER
OK Commit 104 23
.....
BACKEND SERVER
OK Commit
.....

Terminal
File Edit View Search Terminal Help
BACKEND SERVER
OK Commit 100 12
.....
BACKEND SERVER
OK Commit
.....
BACKEND SERVER
OK Commit 100 15
.....
BACKEND SERVER
OK Commit 104 23
.....
BACKEND SERVER
OK Commit
.....

sophia@sophia-VirtualBox: ~/project_2_assignment_1
[1]+ Stopped ./frontend_server
sophia@sophia-VirtualBox:~/project_2_assignment_1$ make compile
g++ frontend_server.cpp -o frontend_server -lpthread
g++ main_client.cpp -o main_client
g++ backend_server.cpp -o backend_server
sophia@sophia-VirtualBox:~/project_2_assignment_1$ ./frontend_server
*****Server*****
OK 100
OK 100
OK 100
.....
QUERY OK 12 100
QUERY OK 12 100
QUERY OK 12 100
.....
UPDATE OK 15 100
UPDATE OK 15 100
UPDATE OK 15 100
.....
backend server3crashed
OK 104
OK 104
OK 104
.....
backend server3crashed
QUERY OK 15 100
QUERY OK 15 100
QUERY OK 15 100
.....
data),0);
getsockopt(new_socket_1[1], SOL_SOCKET, SO_ERROR, &test[k], &len);

```

### Result:

Thus , the distributed banking service with multitier servers was built using two phase commit protocol and fault tolerance mechanisms for the multiple concurrent transaction requests between the client and servers with proper consistency and atomicity checked.