PROJECT 3 (CMSC 621)

GIVEN OUT: 05/03/2017

ASSIGNMENT DUE: TUESDAY, MAY 16TH, 11:59 PM

Design and implement a simple, distributed banking service that permits multiple concurrent operations and exhibits simple fault tolerance. You should run 3 servers, each as a separate process on your machine. Each server should be capable of handling multiple simultaneous requests, and any account should be accessible from any server whether or not it was created on that server.

Additionally, the group of servers should tolerate the loss of any one server in the group without impacting the overall service. You can kill a process to emulate a crash. What this means is that you must replicate the data between servers.

Servers should maintain just enough state as to complete the transactions. For locking, you can use any locking mechanism. For atomicity, you should use the two-phase commit protocol. Write your server and client in C++.

One way of implementing the server is using a multi-tiered architecture. It includes a front-end process that the client always connects to. You can assume that the front-end process never crashes. The front end process then accesses data or updates data on the three servers (those are the backend processes). The locking mechanism can be implemented in the front-end process (the algorithm is upto you).

Note: **Please use setsockopt function to allow the server to bind to an address that is in the TIME_WAIT state.**

TRANSACTION PROTOCOL

SESSION

The protocol used to communicate with the servers should be a TCP service on some specified port(s).

Upon accepting a new session, and at the completion of each client command, the server should return a string with the following format:

status output_string CRLF

status

Either "OK" on connection or success, or "ERR" if an error in processing should occur.

output_string

The output of the command, as defined below. If no output is specified, the string should be null.

CRLF

A carriage return (ASCII 0x13) followed by a line feed (ASCII 0x10).

COMMANDS

All commands entered by the client should be case-insensitive and may be terminated by either a line feed or a carriage return followed by a line feed. The following commands should be supported:

CREATE *initial _deposit*

Creates a new account containing the amount specified by *initial_deposit*, which should be a real value with a precision of no more than 2 digits. On success the output should be a unique non-negative integer ID for the account. On error, the output should be an error message.

UPDATE *acct_id value*

Sets the amount in the account identified by the non-negative integer *acct_id* to the value specified by *value*, a real value with no more than 2 digits of precision. On success, outputs the new value. On error, outputs an error string.

QUERY *acct_id*

Queries the current amount in the account specified by the non-negative integer *acct_id*. On success, returns the current amount as a real value with 2 digits of precision. On error, returns and error message.

QUIT

Terminates the session. No output string is needed.

SAMPLE SESSION

```
% ./client <frontend_server_port> <frontend_server_ip>
OK
CREATE 50.00
OK 100
QUERY 100
OK 50.00
UPDATE 100 150.00
OK 150.00
UPDATE 101 20.00
ERR Account 101 does not exist.
```

```
QUERY 100
OK 150.00
QUIT
OK
Connection closed by foreign host.
```

WHAT TO SUBMIT

(1) I have kept this assignment very flexible. You are welcome to use any locking mechanism you want. You can test everything on one machine by spawning multiple processes.
(2) You will have the implementation of the client and the front-end server process and backend server processes in one directory. Also create a Makefile that will compile the code.
(3) Add a doc file to the source directory that describes how to run the code and what algorithm you are using for locking.
(4) Zip the entire folder the slack it to the TA.