Authors: Manohar Chitoda & Suraj Upadhyay
Assignment: Asst3
Date: December 12, 2018

Description:

This application uses a server/client program to simulate many clients accessing a bank server from different machines. It shows clients connecting and communicating with the server to manage accounts or creating them. Use the makefile to compile this package and produce the executables bankingClient and bankingServer.

Compilation:

In order to compile the source files, use "make" or "make all" in the command line.

Execution:

To run:

```
./bankingServer 26197

./bankingClient IP address 26197
```

Making use of the make file, will provide you with the executables bankingClient and bankingServer. 26197 is the port number our group chose, but any port number above 8k can be used. The IP address is the address to which the client machine will connect to, IP address of the server.

Design:

Server: It starts off by setting up a socket and starting the timer. It then continuous to establishing a server network by binding to a socket and listens for incoming connections. To do that it generates a thread which enters n infinite loop that accepts any incoming connections to the server. When the bank server is connected to the client, a new thread is generated to start the client. This is done to generate a thread every time a new connection is established. The client thread reads for any client input from the clients. Once the message is arriving, it is double checked and then sent to do any task that is asked to perform. The inputbraker() function separates the command and the string and forwards them separately to commandHandler(). This directs the command to appropriate functions and performs appropriate operation on the appropriate account. When SIGINT is invoked, the function termination is called and executed.

Client: It starts off by setting up a socket and acquiring a proper IP address to connect to. Then it enters a loop which constantly tries to connect to the IP address until it's connected. Once it is connected, it spawns two threads; one to read the data sent from the server and one to write the command entered by the user to the server. Any data sent from the server is printed to STDOUT. The thread that takes command from the user, reads the commands and evaluates them. If they are valid, they are sent to the server, if not, hen the user is prompted to enter again.

Menu options presented to the user for the client:
create <account name>
serve <account name>
deposit <amount>
withdraw <amount>
query
end
quit

The client program will send the commands entered by the user to the server and receive responses. At point of quit, the program stops and ends the connection by closing the socket. It also exits the threads and ends the program gracefully.

Assumptions:

We assumed that the hardest part of this assignment was the server and client part and getting a connection setup. We also assumed of the command functions not be as difficult but they were equally as difficult as setting up the socket connection. We faced difficulty tying to convert "cp.cs.rutgers.edu" to an IP address. We had assumption on how to separating the command sent by the user and executing them appropriately.

Difficulty:

We had great difficulty setting up the sockets to connect to the server. We faced great difficulty in finding critical sections to protect using semaphore and implement them appropriately. We faced great difficulty getting the server program to end properly when SIGINT was invoked and not crash. We also had hard time directing the program properly, where we faced infinite loops and segmentation faults.

Function Prototypes:

Server.c :

```
int main (int argc, char *argv[]);
void printData();
void create(char*, BANK *, int);
void serve(char*, BANK**, int);
void withdraw(double, BANK*, int);
void deposit(double, BANK*, int);
void query(BANK*, int);
void end(BANK**, int);
void quit(BANK**, int);
void* termination();
BANK* accountFinder(char*);
```

Client .c :

```
int main (int argc, char *argv[]);
int createConnection(char*,int);
void* readNPrint(void*);
void* sendToProcess(void*);
int inputChecker(char* );
int commandHandler(char*, char*);
void* termination();
```