How to use and implement Multi threaded Bank System

Files included:

bankingClient.c

bankingClient.h

bankingServer.c

bankingServer.h

bank.c

bank.h

Makefile

How to run our code:

Our Makefile complies our three c Source files to two .o files. These are bankingClient executable and bankingServer executable. To run our code you need to run:

```
 ./bankingServer <port number>
```

```
./bankingClient <hostname> <port number>
```

You will need to run these in separate terminals as the Client will connect to the Server and will communicate. The port number in both needs to be the same and the server must also be running on the host provided as the input for the Client. If all of these criteria are met the client will connect to the Server and you can start sending instructions from the Client to the Server.

The instructions you can send are limited to these:

```
create <accountname (char) >
serve <accountname (char) >
deposit <amount (double) >
withdraw <amount (double) >
query
end
quit
```

Makefile:

```
ALL: gcc -Wall -g -o bankingServer bankingServer.c bank.c -lpthread; gcc -Wall -g -o
bankingClient bankingClient.c -lpthread
```

clean:

```
	rm -rf bankingClient; rm -rfbankingServer
```

Description of Files:

bank.c:

- Global Variables:
    - bankAccountNode* allBankAccounts;
        - The global linked list that stores all our accounts. In effect it is our Bank

- bankAccountNode* searchAccount(char inAccountName[256]);
    - Inputs: account name
    - Outputs: the account to be searched from the global Linked List Bank. Returns Null if not found

- void createAccount(char inAccountName[256]);
    - Inputs: account name
    - Outputs: adds an account to allBankAccounts. Checks for duplicate account

- int withdraw(char inAccountName[256], double amount);
    - Inputs: account name, amount to be withdrawn
    - Outputs: gets the proper account and then subtracts the input value from the balance. If it is more than the balance it will throw an error.

- void deposit(char inAccountName[256], double amount);
    - Inputs: account name, amount to be added
    - Outputs: gets the proper account and then adds the input value from the balance.

- void printAccounts();
    - Inputs: n/a
    - Outputs: prints all the accounts to the terminal.

bank.h:

Here are the Global variables and data structures that were used in our bank source file:

- enum State{INSERVICE, INACTIVE, SUCCESS, FAIL};
- Used as flags to let our program know the outcome or status in various functions.

- typedef struct bankAccountNode{

  char accountName[256];

  double balance;

  int status;

  struct bankAccountNode* next;

}bankAccountNode;

- This was the linked list node that is used in our global Linked list of bank accounts. This is the data structure that the entire bank and its data were held.

bankingClient.c:

- int main(int argc, char** argv)
  - This is the place where our Socket is Made. We used the command socket(AF_INET, SOCK_STREAM, 0) to open the network socket that will connect to the Listening Socket running on the Server side. It also initializes the file descriptor for this socket and returns it so that we can use it in our program. It will get the correct inputs from the command line, check if the port is valid, and will get the hostname and convert it to an IP Address used in our Socket methods. It also initialized our two threads: input and output. These threads are both going to be communicating with the Server. At the end it will coin the threads and free.
- int hasMessage(char *string);
  - Checks if a string is empty. This is for our output thread function. It is used to check if a receiving message is empty or not.

- void* outputThread(void* fd);
  - Inputs: the connecting socket file descriptor.
  - Outputs: constantly listens for messages from the Server using the function “recv”. If there is a message it will print it out.

- void* inputThread(void* fd);
  - Inputs: the connecting socket file descriptor
  - Outputs: listens for the inputs from the Client in the Terminal and then checks if the inputs are valid. Then it will send the inputs as one string to the Server utilizing the send function with the Socket.

- void clientQuitSignal(int sig);
  - Inputs: signal that this function is handling
  - Outputs: prepares for the SIGINT signal and will gracefully quit if received

bankingClient.h:

Contains the global variables and the prototypes of our programs

- enum state{FALSE, TRUE};
    - Used as a flag to check the output of some programs
- char receiving[1024];
    - the string that contains the data returning from the Server
- char sending[1024];
    - the string that contains the data sent to the Server
- int serverFD = 0;
    - the File Descriptor of the Socket to be used. This is connecting socket that will attempt to connect to the server Socket.

bankingServer.c:

sem_t bankMutex;

- The semaphore we used to protect commands that requires to be executed exclusively.

FDnode* head.

- The Linked list head of our FD list. This is populated with all of the client connection FD's

- int main(int argc, char** argv);
    - Takes in the proper arguments. And checks to make sure the port is in a valid range. Also initializes and created the threads used in this program. Also sets up the sigint handler that will listen for the ^C shortcut.
- void* listenForClients(void* arg);
    - Inputs: the port that was inputted
    - Outputs: listens for new and incoming connections. This will be various clients trying to connect. Upon new connections, this will spawn a new client session thread.
- void* clientConnection(void* arg);
    - Inputs: The Client session File Descriptor
    - Outputs: It listens for the incoming messages from the client sends and then executes the action received

- bankAccountNode* executeAction(bankAccountNode* curAccount, char message[]);
    - Inputs: The currently accessed node and the message that was sent from the Client
    - Outputs: This is the program that actually implements all of the logic for the input instructions. It will read the message that was inputted and will execute the action as specified in the instructions.

- void* printBank(void* arg);
    - Inputs: none

        - Outputs: Will print the entire Bank Account Linked List

- int isValidAccountName(const char *s);
    - Inputs: char* s
    - Outputs: makes sure there are no spaces or if there is no empty string

- int isValidAmount(const char *s);
    - Inputs: char * s
    - Outputs: Checks that it is a positive number

bankingServer.h:

Here is the global linked list of file descriptors that was used in Server:

```
typedef struct FDnode{

        int fd;

        struct FDnode *next;

} FDnode;
```

void deleteFD(int fd);

- will take the Linked list and delete a node.

void addFD(int fd);

- will take the Linked list and adds a node.

Also contains the prototypes of the Server.c.

Problems we encountered:

- We needed to understand the Semaphores and when to use them vs the mutexes.
- We had trouble when we needed to connect to a host name that was different than our ilabs. We solved this problem by making sure we were always in the same host.

Test Cases:

We checked all of our error messages and various other test cases. We included the test case with nearly all the instructions implemented and all of the Errors and their messages.

TERMINAL 1

```
[jeg239@ilab1 Asst3]$ ./bankingClient ilab1.cs.rutgers.edu 42000

Currently connecting with the server. If not connected within 10 s, Close terminal and try again.
Currently connecting with the server. If not connected within 10 s, Close terminal and try again.
Server: Successfully connected to server.
What action would you like to take?
create joe
Error: That account already exists.
What action would you like to take?
serve joe
Error: Chosen account is already in service.
What action would you like to take?
create ashish
Server: Created account: ashish
What action would you like to take?
serve ashish
Server: Serving account: ashish
What action would you like to take?
deposit ashish
Error: Please enter a valid amount.
What action would you like to take?
deposit 100
Server: Deposit successful.
What action would you like to take?
withdraw 1000
Error: Invalid withdraw amount.
What action would you like to take?
withdraw 10
Server: Withdrawl successful.
What action would you like to take?
Client will close. Thank You.
```

TERMINAL 2

```
[jeg239@ilab1 Asst3]$ ./bankingServer 42000
Server: Current balances:
No accounts in bank:
Server: Successfully connected to client.
Server: Successfully connected to client.
Server: Action received from client: create joe.
Server: Current balances:
joe     $0.00
Server: Action received from client: serve joe.
Server: Action received from client: create joe.
Server: Current balances:
joe     $0.00   IN SERVICE
Server: Action received from client: serve joe.
Server: Action received from client: create ashish.
Server: Action received from client: serve ashish.
Server: Current balances:
ashish  $0.00   IN SERVICE
joe     $0.00   IN SERVICE
Server: Action received from client: deposit ashish.
Server: Current balances:
ashish  $0.00   IN SERVICE
joe     $0.00   IN SERVICE
Server: Action received from client: deposit 100.
Server: Action received from client: withdraw 1000.
Server: Action received from client: withdraw 10.
Server: Current balances:
```

```
ashish  $90.00 IN SERVICE
joe     $0.00   IN SERVICE
Server: Action received from client: deposit 50.
Server: Action received from client: query.
Server: Current balances:
ashish  $90.00 IN SERVICE
joe     $50.00 IN SERVICE
Server: Action received from client: end.
Server: Current balances:
ashish  $90.00 IN SERVICE
joe     $50.00
Server: Action received from client: create testAccount.
Server: Current balances:
testAccount     $0.00
ashish  $90.00 IN SERVICE
joe     $50.00
Server: Current balances:
testAccount     $0.00
ashish  $90.00 IN SERVICE
joe     $50.00
Server: Successfully connected to client.
Server: Action received from client: quit.
Server: Connection was terminated with client.
Server: Closing client service session.
Server: Current balances:
testAccount     $0.00
ashish  $90.00 IN SERVICE
joe     $50.00
```

```
^C
Server: Server is shutting down and closing all of it's clients.
```

TERMINAL 3

```
[jeg239@ilab1 Asst3]$ ./bankingClient ilab1.cs.rutgers.edu 42000
Server: Sucessfully connected to server.
What action would you like to take?
create joe
Server: Created account: joe
What action would you like to take?
serve joe
Server: Serving account: joe
What action would you like to take?
deposit 50
Server: Deposit successful.
What action would you like to take?
query
Account: joe has a balance of $50.00
What action would you like to take?
end
Server: Account session ended.
What action would you like to take?
create testAccount
Server: Created account: testAccount
What action would you like to take?
Client will close. Thank You.
```

TERMINAL 4

```
[jeg239@ilab1 Asst3]$ ./bankingClient ilab1.cs.rutgers.edu 42000
Server: Sucessfully connected to server.
What action would you like to take?
quit
Client will close. Thank You.
```