README: Multithreaded Bank Server

By: Augustus Chang

2 executables were created

- **server**
  - takes 0 command arguments
- **client**
  - takes 1 command argument, the machine that hosts the server (e.g local machine would be 127.0.0.1)
- Please run the **make** command to produce the two executables.
- Run the **make clean** command to remove any unwanted object files.

Overview of Project

To tackle the project, most of the banking and server/client functionalities seperately.

You can find the banking functionalities in **banksys.h** and **banksys.c**

You can find the server/client code in **server.c** and **client.c** respectively.

Thread Synchronization

Here are the main threads used in all the processes

- **2 threads in the client side**
  - For user-server and server-user processes.
  - Does not need to synchronize. In fact this is to encourage asynchronous receiving and sending for better communication.
- **n threads for n clients in the server side**
  - used mutex locks in each account to synchronize the access of each account. Only one account can be serviced as a time.
  - used mutex lock in the open command function (**accOpen**) so only one client can add an account at a time.
  - 1 extra thread is used to print all bank account balances on the server side every 20 seconds.

Client Side

Two threads: 1. Thread 1 : user **->** server 2. Thread 2 : server **->** user

pthread_create() was used to create the new **server-user** thread, while the **user-server** thread runs in the main process thread.

For every write and read, **the validity of the connection** is checked. In case the server shuts down, the client will notify the user the bank has closed and stop the entire process.

IMPORTANT: In case the client shuts down due to **SIGINT (ctrl-c)** we created a **SIGNAL HANDLER FUNCTION** to send the exit command to the server so that the account can properly be closed before the client process ends.

```
int n_fd; // network socket file descriptor

static void sigint_handler( int signo )
{
char command[5] = "exit";
printf( "Received Signal SIGNIT: Sending exit command to Server before Client
dies. . .\n");

if ( (write(n_fd, command, strlen(command) ) ) < 0){
     printf("- - - Sorry, your session has expired. - - -\n");
}
return;
}
```

Server Side

n threads created for **n** client connections

Each newly created thread is rerouted sto the the function **clientServiceFunction** where it reads input continuously to call appropriate functions and send back data through the client socket.

Additionally, another thread prints out to the bank server the account information every 20 seconds.

The main component of the client-service thread is a switch statement. Also, it is very important that we make a local variable session for each thread to determine if any session is available and if any, which account is being accessed.

```
  // session state < 0 if not in session,
  // session state >=0 if in session (specifying location of account)
  // initialize session
  int session = -1;
```

Please refer to the code to more specific implementation.

IMPORTANT : It is worth clarifying that whereas the **finish** command can only be called when the user is in session, the **exit** command can be called when the user is out OR in session. So if the **exit** command is called while the user is in session, the exit will automatically **finish** the session first.

Bank Code

banksys.c and **banksys.h** is where the majority of bank related structures and functions are used.

As you can see, each bank has a lock associated with it.

```
  typedef struct account_ {
    char name[100];
    float balance;
    int service_flag;             // -1: Not created, 1: In service,  0: Not in
service
    pthread_mutex_t lock;
} account;
```

It is worth noting that the **service flag** has **3 states**. **State (-1)** is when the account has not been opened by a client user yet. This is especially useful infromation for which accounts are important when printing out the bank information every 30 seconds.

Here are some of the commands created. **Note: exit is the only command implemented in the server because it must break out of the loop**

```
  /* rest of functions are self explanatory, directly parallel the commands
needed */
void accOpen(int client_socket_fd, int *numAccount, account ***arr_account, char
* name, int *session, pthread_mutex_t * openAuthLock);

void accStart(int client_socket_fd,  account ***arr_account, char * name, int
*session);

void accCredit(int client_socket_fd, account ***arr_account, char * amount, int
*session);

void accDebit(int client_socket_fd, account ***arr_account, char * amount, int
*session);

void accBalance(int client_socket_fd,  account ***arr_account, int *session);

void accFinish(int client_socket_fd, account ***arr_account,  int *session);
```