Roshan Patel
198:211 SYSTEMS PROGRAMMING
PA5: MUTLITHREADED-BANK-SERVER – readme.pdf

OVERVIEW OF PROGRAM USE

My implementation of the multithreaded bank server can be compiled using the provided makefile. It will produce two executables: server and client. This project includes several utilities which are compiled into object files to produce the exectuables. These include tokenizers and hashmap implementations. The following shows the proper usage of my program. NOTE: The server port is automatically set to 60221. This is arbitrarily chosen to be the same as Avogadro's number. The port can be changed by altering the `port.h` file.

```
$ make
$ ./server

/* DIFFERENT TERMINAL TAB (same machine) */
$ ./client localhost
```

DATA STRUCTURES & HELPER FILES

My program implementation uses a Tokenizer structure to tokenize and parse through inputs. This is a slightly altered version of pa1. Additionally, I have implemented a hashmap to store and request account structures within the server program. This made dealing with data within the server program very simple, easy, and efficient.
The data structure used to store account information is a structure named account. Its declaration is shown to the left. It contains 4 fields: a mutex type, a string, a double, and a boolean. Each of these are self-explanatory. An account structure is created upon the "open" command and can be accessed by a client with the "start" command. Starting an account will activate it, changing the boolean field to reflect this. When a client is within an account, they have permissions over changing this account's balance.

```
/* Account structure */
struct account {
    pthread_mutex_t mutex;
    string name;
    double balance;
    bool active;
};
typedef struct account* account;

/* Client connection structure */
struct clientProcess {
    int FD_connect;
    pthread_t thread;
    account acc;
};
typedef struct clientProcess* clientProcess;
```

Each client program accessing the banking server is assigned a clientProcess data structure. Its declaration is shown below the account declaration. This structure stores a file descriptor of the connection (FD_connect) as well as additional information about which account it is currently accessing with "start". This will help ensure that a client cannot access more than one account at a time.

THREAD SYNCRONIZATION

My program implementation uses a Tokenizer structure to tokenize and parse through inputs. This is a slightly altered version of pa1. Additionally, I have implemented a hashmap to store and request account structures within the server program. This made dealing with data within the server program very simple, easy, and efficient.

Thread synchronization was meticulously handled between the server and client programs.

```
/* Connection with server established. */
printf("Connected to Server.\n");

pthread_create(&thread, NULL, handler, serverName);
pthread_detach(thread);

/* Receive user commands. */
pthread_create(&inHandler, NULL, sendCommand, NULL);
pthread_detach(thread);

while(write(FD_listen, "0", 2) == 2){
    sleep(1);
}
```

The server program starts by initiating a file descriptor to listen to the port number declared in port.h. Timers are set to print the contents of the bank account ever 20 seconds. This operates based on a signal handler (SIGALRM). The sever process is terminated with a SIGINT.

When a connection is heard from the port, a clientProcess structure is created for it as well as its own thread. Mutex locks are utilized to ensure that no other processes may enter its thread. The server then accepts commands from the client program and alters the data structures accordingly. Responses are sent to the client program in reaction to commands received.

As per the guidance of the assignment directions, the client produces two threads. The first thread is dedicated to command-inputting. This will read commands from the user and send them to the server. This is the second thread declared (handled by the sendCommand function). The second thread is a response-output thread. It will read messages from the server and forward them to the client program user. This is the first thread initialized (handled by the handler function). If clients cannot connect to the sever, it will retry every 3 seconds. The sendCommand() function will sleep for 2 seconds after accepting a command to throttle client command entry. Client processes will terminate with a SIGINT or the “exit” command.

The client will announce its completion of connecting to the sever with a series of print statements indicating when the threads have all been initialized. The sever program announces its acceptance of a connection from a client with the print statement “New client connection.” When clients disconnect from the server, messages are printed in the server program stating: “Ended session and unlocked account mutex.” The server will display print messages upon receiving any command from any connected client. Lastly, the client will display messages sent from the sever upon successful or failed commands.

TESTCASE ANALYSIS

Test cases attempt to tackle specific functions of the banking system systematically. First, I tested the proper connection between server and client processes. Then, I tested the proper sharing of data between the two processes. Lastly, I tested the proper implementation of banking functions. More information regarding my test cases and expected outputs can be found in the attached text files: bank-testcases.txt and testplan.txt.

TESTCASE 1: Testing connections

SERVER

```
-bash-4.1$ ./server
New client connection.
.............................................
Current Bank Accounts:
Empty bank.
.............................................
```

CLIENT_1

```
-bash-4.1$ ./client localhost
Connected to Server.
Client is prepared for commands.
Server is prepared for commands.

Server Message:  Please enter a command:
```

_____

This simple test case exhibits the messages that are printed when the client and server processes properly connect and are able to communicate.

TESTCASE 2: Connecting multiple clients

```
SERVER
          -bash-4.1$ ./server
          New client connection.
          ..............................................
          Current Bank Accounts:
          Empty bank.
          ..............................................
          New client connection.

CLIENT_01
          -bash-4.1$ ./client localhost
          Connected to Server.
          Server is prepared for commands.
          Client is prepared for commands.

          Server Message:  Please enter a command:

CLIENT_02
          -bash-4.1$ ./client localhost
          Connected to Server.
          Server is prepared for commands.
          Client is prepared for commands.

          Server Message:  Please enter a command:
```

_____

This testcase shows two separate instances of client processes connecting to the same server. The server announces its connection to each with a print statement.

TESTCASE 3: Functionality of account structures

SERVER

```
-bash-4.1$ ./server
New client connection.
Server received command:  <open roshan>
Server received command:  <open charles>
Server received command:  <open goku>
Server received command:  <open voldemort >
Server received command:  <credit>
Server received command:  <debit>
Server received command:  <balance>
Server received command:  <finish>
Server received command:  <start roshan>
Server received command:  <balance>
Server received command:  <finish>
Ended session and unlocked account mutex
Server received command:  <exit>
.........................................
Current Bank Accounts:
      Account Name: charles
      Balance: $0.00


      Account Name: goku
      Balance: $0.00


      Account Name: voldemort
      Balance: $0.00


      Account Name: roshan
      Balance: $0.00
.........................................
```

CLIENT_01

```
-bash-4.1$ ./client localhost
Connected to Server.
Client is prepared for commands.
Server is prepared for commands.

Server Message:    Please enter a command:
open roshan
Server Message: Successfully added account roshan.
open charles
Server Message: Successfully added account charles.
open goku
Server Message: Successfully added account goku.
open voldemort
Server Message: Successfully added account voldemort.
credit 10
```

```
Server Message: Not currently in a session.
debit 5
Server Message: Not currently in a session.
balance
Server Message: Not currently in a session.
finish
Server Message: Not currently in a session.
start roshan
Server Message: Successfully connected to account roshan.
balance
Balance: $0.00
finish
Server Message: Finishing session to account roshan.
exit
```

_____

This test case exercises the proper maintainence of the account data structures within the server program. Multiple accounts are opened and balance changing commands are attempted. Note that the client program received error messages stating that the client is not currently accessing an account. Then an account is accessed with start and the balance is viewed before safely finishing and terminating the client program.

TESTCASE 4: Testing basic commands on one client

```
SERVER
-bash-4.1$ ./server
             New client connection.
             Server received command:  <open bruce>
             ..........................................
             Current Bank Accounts:
                    Account Name: bruce
                    Balance: $0.00
             ..........................................
             Server received command:  <start bruce>
             Server received command:  <credit 10>
             Server received command:  <debit 5>
             Server received command:  <balance>
             Server received command:  <finish>
             Ended session and unlocked account mutex
             ..........................................
             Current Bank Accounts:
                    Account Name: bruce
                    Balance: $5.00
             ..........................................
             Server received command:  <exit>
CLIENT_01
             -bash-4.1$ ./client localhost
             Connected to Server.
             Client is prepared for commands.
             Server is prepared for commands.

             Server Message:     Please enter a command:
             open bruce
             Server Message: Successfully added account bruce.
             start bruce
             Server Message: Successfully connected to account bruce.
             credit 10
             Balance:     $0.00
             Credit:      $10.00
             New Balance: $10.00
             debit 5
             Old Balance: $10.00
             Depositing:  $5.00
             New Balance: $5.00
             balance
             Balance: $5.00
             finish
             Server Message: Finishing session to account bruce.
             exit
```

_____

This test case will exercise all the capabilities of one client process. It will open an account, start that account, credit money, debit a valid amount, view its balance, and then safely finish accessing. The client program then terminates with exit. Notifications can be seen in the server program when commands are received. The bank information is also validly updated.

TESTCASE 5: Serving multiple clients simultaneously

SERVER

```
-bash-4.1$ ./server
New client connection.
New client connection.
Server received command:  <open obiwan>
Server received command:  <open darthmaul>
.........................................
Current Bank Accounts:
       Account Name: darthmaul
       Balance: $0.00


       Account Name: obiwan
       Balance: $0.00


.........................................
Server received command:  <start darthmaul>
Server received command:  <credit 1000>
Server received command:  <start obiwan>
Server received command:  <credit 50>
Server received command:  <debit 3>
.........................................
Current Bank Accounts:
       Account Name: darthmaul
       Balance: $1000.00
       IN SERVICE

       Account Name: obiwan
       Balance: $47.00
       IN SERVICE

.........................................
Server received command:  <debit 4>
Server received command:  <finish>
Ended session and unlocked account mutex
Server received command:  <finish>
Ended session and unlocked account mutex
Server received command:  <exit>
Server received command:  <exit>
```

CLIENT_01

```
-bash-4.1$ ./client localhost
Connected to Server.
Server is prepared for commands.
Client is prepared for commands.

Server Message:     Please enter a command:
open obiwan
Server Message: Successfully added account obiwan.
start obiwan
Server Message: Successfully connected to account obiwan.
```

```
credit 50
Balance:     $0.00
Credit:      $50.00
New Balance: $50.00
debit 3
Old Balance: $50.00
Depositing:  $3.00
New Balance: $47.00
finish
Server Message: Finishing session to account obiwan.
exit
```

CLIENT_02

```
-bash-4.1$ ./client localhost
Connected to Server.
Server is prepared for commands.
Client is prepared for commands.

Server Message:    Please enter a command:
open darthmaul
Server Message: Successfully added account darthmaul.
start darthmaul
Server Message: Successfully connected to account darthmaul.
credit 1000
Balance:     $0.00
Credit:      $1000.00
New Balance: $1000.00
debit 4
Old Balance: $1000.00
Depositing:  $4.00
New Balance: $996.00
finish
Server Message: Finishing session to account darthmaul.
exit
```

_____

This test case does the same as the previous except with two clients accessing two separate accounts simultaneously. The first client process will open, start, credit, and debit an account. The second process does the same concurrently (from a different terminal tab on the same machine). The server accurately displays messages regarding the commands received from each client process.

TESTCASE 6: Filling up sever to limit

```
SERVER
              -bash-4.1$ ./server
              .
              .
              .



CLIENT_01
              -bash-4.1$ ./client localhost
              Connected to Server.
              Client is prepared for commands.
              Server is prepared for commands.

              Server Message:     Please enter a command:
              open 1
              Server Message: Successfully added account 1.
              open 2
              Server Message: Successfully added account 2.
              .
              .
              .
              Server Message: Successfully added account 19.
              open 20
              Server Message: Successfully added account 20.
              open 21
              Server Message: Cannot create more than twenty accounts.
```

This is a simple test case to ensure that greater than 20 accounts cannot be created for the server. A single client process attempts to create 21 accounts. An error message is prompted from the sever on the 21st command.

TESTCASE 7: Attempting to open duplicate accounts

```
SERVER
             -bash-4.1$ ./server
             New client connection.
             Server received command:  <open sally>
             Server received command:  <open sally>
             Server received command:  <quit>

CLIENT_01
             -bash-4.1$ ./client localhost
             Connected to Server.
             Server is prepared for commands.
             Client is prepared for commands.

             Server Message:     Please enter a command:
             open sally
             Server Message: Successfully added account sally.
             open sally
             Server Message: Cannot create account sally because the account name
already exists.
```

_____

Another simple test case to check that multiple account names cannot be registered in the banking server. An error message is prompted by the server program upon receiving the command to create a duplicate account.

TESTCASE 8: Starting client process before server

```
SERVER
            -bash-4.1$ ./server
            New client connection.

CLIENT_01
            -bash-4.1$ ./client localhost
            Attempting to connect to localhost...
            Connection refused
            Connected to Server.
            Client is prepared for commands.
            Server is prepared for commands.

            Server Message:     Please enter a command:
```

_____

This test case will start the client process before the server process. When started, the client process is unable to connect with the server on the given port. As designed, it will wait 3 seconds and try again. By this time, the server process is started and the connection is established.

TESTCASE 9: Functionality of credit and debit

```
SERVER
            -bash-4.1$ ./server
            New client connection.
            Server received command:  <open 2pac>
            Server received command:  <start 2pac>
            Server received command:  <credit 1000>
            Server received command:  <debit 1000000>
            Server received command:  <debit 10>
            Server received command:  <balance>
            ..........................................
            Current Bank Accounts:
                  Account Name: 2pac
                  Balance: $990.00
                  IN SERVICE

            ..........................................
            Server received command:  <finish>
            Ended session and unlocked account mutex
            Server received command:  <exit>

CLIENT_01
            -bash-4.1$ ./client localhost
            Connected to Server.
            Client is prepared for commands.
            Server is prepared for commands.

            Server Message:     Please enter a command:
            open 2pac
            Server Message: Successfully added account 2pac.
            start 2pac
            Server Message: Successfully connected to account 2pac.
            credit 1000
            Balance:     $0.00
            Credit:      $1000.00
            New Balance: $1000.00
            debit 1000000
            Server Message: Not enough funds to debit this amount.
            debit 10
            Old Balance: $1000.00
            Depositing:  $10.00
            New Balance: $990.00
            balance
            Balance: $990.00
            finish
            Server Message: Finishing session to account 2pac.
            exit
```

This test case exercises the proper functions of the credit and debit commands. A client can always credit to an account. A client can only debit from an account if sufficient funds are available.

TESTCASE 10: Attempting to start on an active session

SERVER

```
-bash-4.1$ ./server
New client connection.
Server received command:  <open steve>
Server received command:  <open jobs>
New client connection.
Server received command:  <start jobs>
Server received command:  <start jobs>
Error attempting to lock already locked mutex: 16
.........................................
Current Bank Accounts:
      Account Name: steve
      Balance: $0.00


      Account Name: jobs
      Balance: $0.00
      IN SERVICE

.........................................
Error attempting to lock already locked mutex: 16
Error attempting to lock already locked mutex: 16
Error attempting to lock already locked mutex: 16
Server received command:  <finish>
Ended session and unlocked account mutex
Server received command:  <credit 10>
Server received command:  <finish>
Ended session and unlocked account mutex
.........................................
Current Bank Accounts:
      Account Name: steve
      Balance: $0.00


      Account Name: jobs
      Balance: $10.00


.........................................
Server received command:  <exit>
Server received command:  <exit>
```

CLIENT_01

```
-bash-4.1$ ./client localhost
Connected to Server.
Client is prepared for commands.
Server is prepared for commands.

Server Message:     Please enter a command:
open steve
Server Message: Successfully added account steve.
```

```
open jobs
Server Message: Successfully added account jobs.
start jobs
Server Message: Account jobs is already in use.
Server Message: Waiting to start session for account jobs.
Server Message: Waiting to start session for account jobs.
Server Message: Waiting to start session for account jobs.
Server Message: Waiting to start session for account jobs.
Server Message: Successfully connected to account jobs.
credit 10
Balance:     $0.00
Credit:      $10.00
New Balance: $10.00
finish
Server Message: Finishing session to account jobs.
exit
```

CLIENT_02

```
-bash-4.1$ ./client localhost
Connected to Server.
Client is prepared for commands.
Server is prepared for commands.

Server Message:    Please enter a command:
start jobs
Server Message: Successfully connected to account jobs.
finish
Server Message: Finishing session to account jobs.
exit
```

_____

This test case exercises the mutex locking between different accounts. One client opens and starts an account. The other client (client_01) attempts to start this account, but is prompted with an error message from the server. It will begin to wait for the account to be finished by the other client process. Once client_02 finishes with the account, client_01 is able to successfully connect. Throughout this process, the server program reports messages accordingly.

TESTCASE 11: Client exits with sever exit

```
SERVER
             -bash-4.1$ ./server
             New client connection.
             ^C
             -bash-4.1$


CLIENT_01
             -bash-4.1$ ./client localhost
             Connected to Server.
             Client is prepared for commands.
             Server is prepared for commands.

             Server Message:     Please enter a command:
             -bash-4.1$
```

_____

This test case exercises the safe exiting of the client processes when the server is terminated with a SIGINT.