Shilp Shah (RUID: 170009205)
Mathew Varghese (RUID: 170009651)

# Assignment 3 Documentation

PORT NUMBER USED: 9572

## Approach

Since this project deviated from our old code base, we were able to better design our project for modularity and better functionality. First, we started by writing a simple client program that creates a new thread and connects to a server. With some research online and our notes from class, we were able to setup a client that takes in 2 parameters, a machine host and a port number to be able to connect to that server and setup a data port. Then we started setting up the server side to be able to accept these client connections, keeping in mind the requirements for creating a new thread for each client. After we were able to communicate between the server and multiple clients, we started designing the layout of our structs, and how we wanted to handle each account.

## Design

We made an “account” struct that has name, balance, and inSession fields. Each struct will hold the necessary data for a specific account. In the server.c file, we have a gobal variable called Bank and this is a pointer to a pointer of account structs. This allows us to store each account in an array format as they are created. When the create command is run, we use mutex lock to make sure multiple threads are not accessing the data, then add that account to the global variable Bank. With each command, the specific account was found by name and modified based on the command the user ran. Some of the error handling was a little tricky because there were many cases to account for. For example, we had to make a variable called inSessionClient in each client thread to see if that client was already in session for another account, and if they were in session, then they had limited functionality with the commands they can run.

## SIGINT Signal

When the SIGINT signal is triggered, the server goes through the list of client file descriptors and sends each client a message and closes the connection.

```
while(temp != NULL)
{
      int fdInt = temp->fd;
      if ((send(fdInt, "Server shutting down. Ending client", 100,
      0))== -1)
     {
            printf("ERROR: Could not shut down one of the clients.\n");
            close(fdInt);
            break;
      }
      temp = temp->next;
}
```

SIGALRM Signal (includes semaphore)

```
void printThreadFcn(void * args)
{
  alarm(15);

  while(1)
  {
    sem_wait(&bankSem);
    if(flag)
    {
      printBank();
      flag = false;
      alarm(15);
    }
    sem_post(&bankSem);
  }
}
```

Printing All Account Information

After we use a semaphore to lock the all the account data, we call a function that prints each account to STDOUT on the server side. This is relatively straightforward because it is just a for loop that iterates through all the accounts and prints the name, balance and whether or not that account is in session.

Test Cases

We mainly tested our code with simple test cases, such as 5 clients open at once with about 20 accounts. It was hard to simulate many clients with hundreds of accounts at once to fully test our multithreading, mutex locks and semaphores. But all the test cases have worked so far, and the results are below.

Test Case 1:

Client 1: create test
Client 1: serve test
Client 2: serve test
→ Outputs to client 2: ERROR: Account is already in session
Client 2: create test
→ Outputs to client 2: Account name already exists. Choose another name
Client 1: end
Client 2: serve test
→ Allows client 2 to serve the account named “test”

Test Case 2:

Client 1: create test
Client 1: serve test
Client 1: deposit 3.2
Client 1: withdraw 4
→ Cannot withdraw more than balance
Client 1: end
Client 2: serve test
Client 2: query
→ Current balance of account “test” is 3.2
Client 2: quit
→ Connection Closed.

These simple test cases that each of the commands work properly and that data is being shared across all clients. After each command is run, the client is given either a confirmation that the command executed or a reason for why the command cannot execute. All while the clients are running, the server outputs the list of all accounts every

15 seconds, and also outputs when there is a new client connection or client disconnection.