Project 1 Report
Programmer: Ka Son Chan KaSonChan@my.unt.edu
Class: CSCE 3530, Spring 2013
Due: Mar 24, 2013
cse01.cse.unt.edu
Description: This Report.pdf file contains descriptions of the project, my approach, limitations.

This project is a Server-Client math library. User can set up the server using TCP or UDP connection. After setting up the TCP or UDP, other user can choose to connect to the server using TCP or UDP. User then can input the math problem including the addition, subtraction, multiplication and division operations and the server will return the result in decimal number.

I divided files mainly into two main sections – client and server side.
Client side files:

- `Client.c`
- `ClientFunctions.h`
- `ClientFunctions.c`

Server side files:

- `Server.c`
- `ServerFunctions.h`
- `ServerFunctions.c`

In additional to the files as I mention above that I attached in the `SourceCode` folder, I attached one `makefile` file which contains the instructions for the compiler to compile all my files, and one `README.txt` file which contains the instructions for user to compile, execute and test the program. Also I attached a `TestResult` folder which contains some `.png` files of my test result on TCP and UDP.

I tested executing one server with three clients in each connection type. `TCP1.png` and `TCP2.png` are the test for TCP connections where `UDP1.png` and `UDP2.png` are for UDP connections. `TCP1.png` and `UDP1.png` contain the server connection. TCP2.png and UDP.png contain just clients.

I tested compiling and executing everything on the `cse01.cse.unt.edu` machine.

My approach

For the server side:

I implemented the main function in `Server.c` file. I first let the user to choose the connection type: TCP or UDP. Then let the user to enter port number and the program validate the port number. The program will print out error if the user input a port number that is bigger than 60000. Then according to the user input connection type, the program will call the appropriate function `TCPServer()` or `UDPServer()` function.

After going into the functions, the program will print out the user chosen server type TCP or UDP. The program will then set up the transport end point. In TCP connection, the socket is `socket(AF_INET, SOCK_STREAM, 0)`, where the UDP is `socket(AF_INET, SOCK_DGRAM, 0)`. Both server will loop forever to wait for client connection/request.

After the server receives packet from client, the server will calculate the result using the `CalculationResult()` function and pass it back to the client directly in decimal number through the socket.

For the client side:

I implemented the main function in `Client.c` file. I first let the user to input the connection type: (1) TCP or (2) UDP by entering the number. The program then will validate the user input connection type. Then the program will let the user to input the host name. Remember to input `localhost` for testing. Then the program will validate the hostname. The program will let the user to input port number and validate it. If the user input some port numbers bigger than 60000 or smaller than 0, the program will print error message.

After that the user can enter opcode for choosing addition, subtraction, multiply or division, first and second operands in binary number. After all the input, the program will concatenate all the information into a packet buffer.

1	0	3	5																							
				Operand 1								Operand 2								Opcode						

*Where 1 denotes version 1, 0 denotes request, 35 denotes the length.

I changed the number of Operand 1 and Operand 2 because original 4 can only represents up to $1111_2 = 15_{10}$; where 8 can represents $11111111_2 = 255_{10}$.

After building up the packet message, the program will call `TCPClient()` or `UDPClient()` function in `ClientFunctions.c` according to the user input connection type before. The program will then print the user input connection type. The program will set up the transport end point as the server mention above. In TCP connection, the socket is `socket(AF_INET, SOCK_STREAM, 0)`, where the UDP is `socket(AF_INET, SOCK_DGRAM, 0)`. Then the program will send the packet and receive the result if there is no error occur.

Limitations

- User must input binary number for the operands.
- If the either one of the operands is a negative number, the client side will treat it as a positive number and do the calculations.
- User need to input `Ctrl-c` to terminate the Server program because the server is in infinite loop waiting for the client math problems.