Socket Programming - Assignment 2

Akshansh Bhanjana - 11805

Anveshan Lal - 11808

Given below is the directory structure:

```
.
├── manual.pdf
├── README.md
└── src
    ├── crc_error_detection
    │   ├── client.c
    │   ├── makefile
    │   └── server.c
    └── file_transfer_and_rpc
        ├── client.cpp
        ├── client_files
        │   ├── wallpaper1.png
        │   ├── wallpaper2.jpg
        │   └── wallpaper3.jpeg
        ├── server.cpp
        └── server_files
            ├── wallpaper4.jpg
            ├── wallpaper5.png
            └── wallpaper6.png
```

5 directories, 13 files

Information

server.cpp
Contains the following functions:

- **CreateSocket()** → creates a socket which uses the TCP／IP protocol suite's IPv4 protocol
- **BindSocketToAddress()** → binds the given socket to a specified address in order to be able to listen to incoming connection requests to the address
- **GetConnectionFromQueue()** → listens for incoming connections at the given address; accepts in case of a connection request
- **ListenForMessages()** → continuously checks for changes in the buffer so as to monitor incoming messages, and sends back response codes

client.cpp
Contains the following functions:
- **CreateSocket()** → creates a socket which uses the TCP／IP protocol suite's IPv4 protocol
- **ConnectSocketToAddress()** → connects the socket to the given address and port combination
- **SendMessages()** → waits for user input, and sends the entered messages／files to the server

File Transfer and RPC

Client Files ／ Server Files

The **client_files** folder contains some sample files for sending to the server. As soon as one sends a file, it should reflect in the **server_files** folder.

Usage

- compile the *server.cpp* and *client.cpp* files (preferably using `g++`)
- run the **server** compiled file
  (if everything goes well, the server should be listening for connections now)
- run the **client** compiled file
  (if everything goes well, the server and client should both be connected, and the client should be asking for user input)

The **client** accepts 5 kinds of input:
1. **plain text** → write any plain text message from the client side, and the server should receive it, and send back a response code

2. **fileS** → write `fileS::` followed by the file name (with extension, sample - `file::wallpaper1.png`) you want to send (the file should be present in the **client_files** directory)
3. **fileR** → write `fileS::` followed by the file name (with extension, sample - `file::wallpaper4.jp`g) you want to receive (the file should be present in the **server_files** directory)
4. **func** → write `func::` followed by any two of the below function names, followed by 2 arguments:
   a. add
   b. subtract

   Example: `func::add 4 7`
5. **"end connection"** → write this message as plain text (without inverted commas) to end the connection between the server and client, and exit safely

NOTE: It might take some time to reinitialize the server after exiting, as it takes some time for the system to verify that the port is no longer in use.

CRC Error Detection

Usage

- go into the `crc_error_detection` directory, and run `make`
- to launch the server, run `sudo ./server <port>`
- to launch the client, run `./client <address> <port>`

Example:

```
make

sudo ./server 8081

./client 127.0.0.1 8081
```

Now, choose if you want to add errors to your messages or not. The server will respond with `ACK` or `NACK` depending upon your selection.