CS 4470 Programming Assignment 1

A Chat Application for Remote Message Exchange

1. Objective

Getting Started: Familiarize yourself with socket programming.
Implement: Develop a simple chat application for message exchange among remote peers.

2. Getting Started

Socket Programming: Beej Socket Guide: http://beej.us/guide/bgnet

3. Implement

3.1 Programming environment

You will write **C/C++** or **Java** code that compiles in Linux (e.g., Ubuntu) or Windows system

NOTE: You should (1) use **TCP Sockets** in your peer connection implementation; (2) use the select() API or multi-threads for handling multiple socket connections; (3) integrate **both** client-side and server side code into **one program** and run on each peer.

3.2 Running your program

Your process (your program when it is running in memory) will take one command line parameters. The parameter indicates the port on which your process will listen for the incoming connections. For example, if your program is called *chat*, then you can run it like this: `./chat 4322`, where 4322 is the listening port. Run your program on three computers and perform message exchange.

3.3 Functionality of your program

When launched, your process should work like a UNIX shell. It should accept incoming connections and at the same time provide a user interface that will offer the following command options: (Note that specific examples are used for clarity.)

1. help Display information about the available user interface options or command manual.

2. myip Display the IP address of this process.
Note*:* The IP should not be your "Local" address (127.0.0.1). It should be the actual IP of the computer.

3. myport Display the port on which this process is listening for incoming connections.

4. connect <destination> <port no> : This command establishes a new TCP connection to the specified <destination> at the specified < port no>. The <destination> is the IP address of the computer. Any attempt to connect to an invalid IP should be rejected and suitable error message should be displayed. Success or failure in connections between two peers should be indicated by both the peers using suitable messages. Self-connections and duplicate connections should be flagged with suitable error messages.

5. list Display a numbered list of all the connections this process is part of. This numbered list will include connections initiated by this process and connections initiated by other processes. The output should display the IP address and the listening port of all the peers the process is connected to.

E.g.,
```
id: IP address                Port No.
 1: 192.168.21.20                4545
 2: 192.168.21.21                5454
 3: 192.168.21.23                5000
 4: 192.168.21.24                5000
```

6. terminate <connection id.> This command will terminate the connection listed under the specified number when LIST is used to display all connections. E.g., terminate 2. In this example, the connection with 192.168.21.21 should end. An error message is displayed if a valid connection does not exist as number 2. If a remote machine terminates one of your connections, you should also display a message.

7. send <connection id.> <message> (For example, send 3 Oh! This project is a piece of cake). This will send the message to the host on the connection that is designated by the number 3 when command "list" is used. The message to be sent can be up-to 100 characters long, including blank spaces. On successfully executing the command, the sender should display "Message sent to <connection id>" on the screen. On receiving any message from the peer, the receiver should display the received message along with the sender information.

(Eg. If a process on 192.168.21.20 sends a message to a process on 192.168.21.21 then the output on 192.168.21.21 when receiving a message should display as shown:
```
Message received from 192.168.21.20
Sender's Port: <The port no. of the sender>
Message: "<received message>"
```

8. exit Close all connections and terminate this process. The other peers should also update their connection list by removing the peer that exits.

4. Submission and Grading

4.1 What to Submit

Your submission should contain a zipped file – Name it as < your cin_name>.tar:

- A README file that documents each member's contribution in details and explains how to install any prerequisites, build your program, and run your application.
- A tar or zipped file – Name it as < your cin_name>.tar, which includes all source files (.h and .c or .cc or .java files), including Makefile. Note: name your main program as chat.c or chat.cc or chat.java.

4.2 How to submit

Use CS Network Service (CSNS) to submit the zipped file.

Each group is required to make a project demo video and upload it to YouTube. The length of this video should be about 5 minutes (8 minutes maximum). After uploading the demo video, please submit the video link URL in Canvas. This demo video shall show:

1. The significant part of your source code with brief explanation
2. Live compilation of your source code
3. Output of all commands you implemented with brief explanation. You may do this demo in one laptop by opening two or three (if your server program accepts multiple client connections) command windows
4. Each group will need to play this video with me and be ready to answer any questions I may raise.

4.3 Grading Criteria

- *Each group will be scheduled for project demo. All the members should attend the demo, and describe their coding contribution in this project.*
- Correctness of output and exceptional handling.
- Organization and documentation of your code.

4.4 Important Key Points:

- There is just one program. DON'T submit separate programs for client and server.
- Error Handling is very important – Appropriate messages should be displayed when something goes bad.
- DON'T ASSUME. If you have any doubts in project description, please come to my office hour.
- Submission deadline is hard. Absolutely no extension!
- Please do not submit any binaries or object files or any test files.