CS 457 Project 1

Simple Chat Program Coded in C

Introduction

In this project you will use socket programming written in C to implement a simple chat capability. It is **strongly suggested** that you reference ***Beej's guide to internet programming*** to help you with this project (beej.us/guide).

This assignment will consist of one program that acts as both the client and the server. The program will communicate simple messages back and forth starting with the client sending to the server. Your code must work on the Linux machines in the 120 Linux Lab. Test your code by running your client on one machine and your server on another. Chat will be strictly back and forth, after sending a message, the program waits to receive a message before sending again (see example interaction). Messages may be up to 140 characters and should be checked by the program. Print an error message and ask for a new message if the user inputs more than 140 characters. You will write three files for this assignment: chat.(c | cpp), README, makefile. Requirements for each are stated below

Chat Example:

Server	Client
$./chat Welcome to Chat! Waiting for a connection on 192.82.47.232 port 3360 Found a friend! You receive first. Friend: hello! You: Hi! How are you? Friend: I'm great. You: That's good. Ok, bye! ^C	$./chat -p 3360 -s 192.82.47.232 Connecting to server... Connected! Connected to a friend! You send first. You: hello! Friend: Hi! How are you? You: I'm great. Friend: That's good. Ok, bye! You: ▌

Input too long example:

```
$./chat -p 3360 -s 192.82.47.232
Connecting to server... Connected!
Connected to a friend! You send first.
You: hello! I just got back from lake
Chargoggagoggmanchauggagoggchaubunagungamaugg.
Next year, we are going on a trip to the
Mississippi river to go fishing!
Error: Input too long.
You: Hello!
Friend: Hi!
You: ▌
```

chat.(c | cpp) Requirements

The program is invoked as follows:

Server side:
$./chat

Client side:
$./chat -p 3790 -s 192.168.47.232

Where the -p flag indicates the port to connect to and the -s flag indicates the IP address of your friend.

$./chat -h

Should produce a help message and exit.

Without any arguments, the chat program acts as a server, prints out the port it is listening on and waits for a connection.
With arguments, the program acts as a client connects to a waiting server using the information provided.
With one argument, the program prints the help message and exits.
The arguments may be in any order and should be sanity-checked. For example, a port must be a number (no letters) and a server address must also be numbers but with the "." character allowed.

Server

1. Set up a TCP port and listen for connections (print out IP and PORT listening on).
2. Accept connection from client
3. Block to receive a message from the client.
4. Receive message and print to screen.
5. Prompt the user for a message to send.
6. Send the message to the client.
7. GOTO step 3.

Client

1. Set up a TCP connection to the server on the IP and port specified.
2. Prompt the user for a message to send.
3. Send the message to the server.
4. Block to receive a message from the server.
5. Receive message and print to screen.
6. GOTO step 2.

The programs will continue until terminated (^c) by the user.

README

You **must** include a readme that explains to a potential user the process of invoking your program. You may assume that the user is familiar with using terminal programs. Also include questions that you asked the TA and the answer to those questions and any assumptions you have made along the way (hint: ask before assuming!).

Makefile Requirements

You must include a makefile that produces an executable called *chat*.

Interaction Requirements

Programs will be randomly paired up and will have their client and server tested with the other's client and server. To make this possible, you must follow the packet format specified below.

Version (16 bits, Binary)	string length (16 bits, Binary)
Message (up to 140 bytes, ASCII)	

Where the Version is always 457, the string length is the length of the message in bytes, and the message follows immediately afterwards.

Grading

- No credit will be given to a program that fails to compile on the department linux machines.
- 5% Makefile included and compiles programs without errors.
- 5% README included and contains enough information to run program.
- 10% input flags accepted in any order and sanity-checked.
- 10% Program produces an error and asks for another message if the message is over 140 characters.
- 10% Program uses network-to-host and host-to-network.
- 10% Programs work with other team's randomly chosen client and server.
- 45% Program passes send/receive test cases.
- 5% Output formatting looks like the example.

Submission Instructions

Turn in your assignment submitting a tarball of your files to CANVAS. Be sure that the files are at the root of the archive and not in any folders.

POINTS

This exercise is worth 100 points. Overall, projects are worth 25% of your total grade. There will be two projects, so in essence, this project is worth 12.5% of your total grade.