# CS 457 Fall 2023 Project 1: Chat Program in C

**Language**: C **Total**: 100 Points

## Introduction

In this project, you need to use socket programming written in C to implement a simple chat capability. Therefore, it is strongly suggested that you read Beej's Guide to Network Programming - Using Internet Sockets (https://beej.us/guide/bgnet/pdf/bgnet_usl_c_1.pdf) to help you with this project.

This assignment consists of one program that acts as both the client and the server. The program communicates simple messages back and forth, starting with the client sending a connection request to the server. Test your code by running your client on one machine and your server on another. Chat is strictly back and forth; after sending a message, the program waits to receive it before sending it again (see example interaction). Messages may be up to 140 characters and should be checked by the program. Print the error message and ask for a new message if the user inputs more than 140 characters. Finally, you need to complete the given source code file chat.c files for this assignment. The makefile is given for this project, and you must use this file to test your project. Your code must work on the Linux machines in the 120 Linux Lab.

**Remember: It is your responsibility to test your project on the Linux machines in the 120 Linux Lab - that is where the GTAs grade the project.**

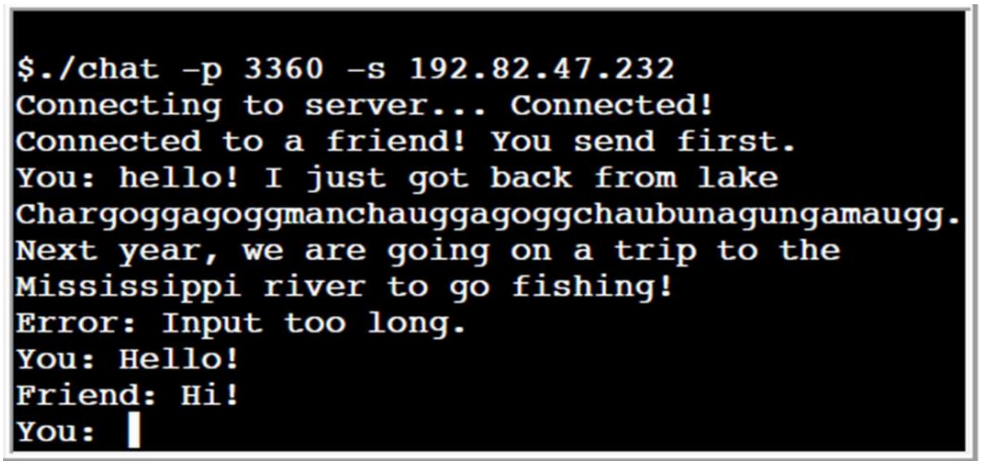
Requirements for each are stated below.

**Chat Example:**

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

**Input too long example:**

- If the user's input is to long (i.e. >140 chars) You must print “Error: Input too long.” and let the user try again.




## chat.c Requirements

The program is invoked as follows:

Server-side: $ ./chat

Client side: $ ./chat -p 3790 -s 129.82.44.141 or ./chat -s 129.82.44.141 -p 3790
The -p flag indicates the port to connect to, and the -s flag indicates the IP address of your friend.

The order (-p followed by -s / -s followed by -p) must work.

Help Command: $ ./chat -h
Should produce a help message and exit.

- The chat program acts as a server without any arguments, 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 numbered. Still, with the "." character allowed.
    - If arguments are not valid print: “<Port/IP> is not valid” Ex: if I enter a port of letters “abcd” it should print “Port is not valid” and then exit.

**Server**

1. Set up a TCP port and listen for connections (print out IP and PORT is 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.

**Note:** The programs continue until terminated (ctrl + c) by the user.

**Makefile**

You must use the given makefile that produces an executable called chat. Do not modify the makefile.

**Interaction Requirements**

Programs are randomly paired up and 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. Remember to use the correct byte order (endianness) for the packets. Packets sent on the network should have **Big-Endian** byte order and then be converted back to match the host when they are received. Refer to Beej's Guide for more information.

| 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 afterward.

**Grading**

- No credit is given to a program that fails to compile on the department's Linux machines.
- 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. (See sec 9.12 pg 101 of guide)
- 10% Program work with another team's randomly chosen client and server.
- 50% Program passes send/receive test cases.
- 10% Output formatting looks like the example.

**Points**

This exercise is worth 100 points. Overall, projects are worth 25% of your total grade. Therefore, this project is worth 10% of your total grade.

**Submission Instructions**

Turn in your assignment (only **chat.c**) submitting a tarball (GroupName) to CANVAS. Be sure that the files are at the root of the archive and not in any folders. GTAs evaluate the project using the given makefile.

**Support**

For general questions about this assignment please post the question in the "Programming Assignments" Teams channel.

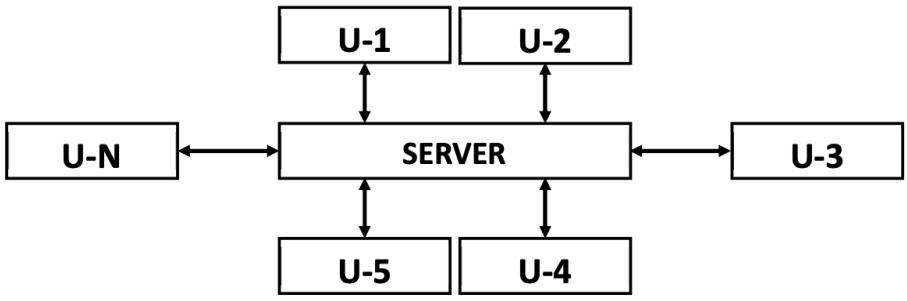
For specific questions regarding this assignment, email or message the GTAs on Teams. Email with the subject line: **CS457 Fall 2021 Project1 Query** at cs457@cs.colostate.edu

You can get supports from the GTAs during their office hours , over email, or Teams. The following table contains the required contact information.

| | | |
|---|---|---|
| **GTA Name** | James Yost | Saja Alqurashi |
| **GTA Office** | CSB 120/MS Teams | TBA |
| **GTA Office Hours** | M 9 - 12 | TBA |
| **Email** | cs457@cs.colostate.edu | |

**Extra Credits 20 Points**

In this extended version, you need to create multiple sockets to exchange messages between users and server. From the server's end, each user is given a numerical ID. The number of message exchanges between users and the server is limited. There is no way for a user to send a text to another user. Any user can receive a message from the only server. A user types and sends the message and waits to receive a response from the server. The server receives texts continuously from the users. To reply, the server gives the user id and then :: symbol and then messages. For example, if the server wants to send "Welcome to CS457 Fall-23 Class" to user 5, the input must be 5: Welcome to CS457 Fall-23 Class. User 5 receives only "Welcome to CS457 Fall-23 Class," and the rest of the parts are discarded. Users send messages directly to the server without adding any id number and symbol (::).




**Submission**

Turn in your assignment submitting a tarball (GroupName) to CANVAS Extra Credit Submission. This should include chat_extra.c, and a README. Where the README contains instructions on how to run your code, and any assumptions you needed to make to complete this part of the assignment. Be sure that the files are at the root of the archive and not in any folders. GTAs evaluate the project using the given makefile.