CISC450/CPEG419 Programming Assignment 2

DUE: Friday May 19th 11:59PM

General Instructions

1. This is an individual programming assignment.
2. You are allowed to consult any Internet resource, books, the course TA, or the instructor. You are not allowed to exchange code snippets or anything across groups. While you may read on general socket programming on the Internet and look at code examples, this should be only for the purpose of understanding. Do all the coding yourself, do not copy or cut-paste code from **ANY** website.
3. You can use only socket programming for the project. Do not use any sophisticated libraries downloaded from the web.
4. You should use C/C++ for coding.
5. You should allocate sufficient time for testing your code.
6. It is important that your code has adequate comment and proper indentation. Comment during/before coding, not at the end. For every variable/function you should have a comment. Use appropriate code comments for explaining the logic where necessary.
7. Provide documentation for your code. This should explain the code structure in terms of directories, files, how to compile etc. Detailed instructions for submitting the documentation along with code are at the end of the document.
8. Choose proper and intuitive names for the variable, function, file, directory, etc.
9. All of the above aspects of your code will be evaluated, not just whether or not the code runs.
10. The project below involve multiple hosts. You may want to write the code such that your code can be run on a single host where the multiple hosts are mimicked by using the loopback address (127.0.0.1) with different ports. This may help during the initial phases, where you need to do a lot of debugging. The IP address/port specification for the different hosts is ideally provided through configuration files or as arguments to your code.

Project: Client-Server Chat System

In this project, you have to write a chat program use UDP sockets under the client-server architecture for real-time communication between 3 users.

- A user starts the chat program sending its username to the server. The server then returns the list of the usernames of current users to the user, which should be displayed at the user side.
- The user can then choose to send messages to either one designated user through unicast (i.e., private chat) or all the other users through broadcast.
- The chat program needs to allow a user to type messages, which should be conveyed to the other end(s) and displayed at the other end(s).
- As a next step, implement a feature to allow users to exchange files. That is, in a two-user session chat between machines A and B, the user at machine A should be able to send a file to machine B. The machine B's user will choose whether or not to receive the file. Check that your program works for both text files as well as binary files.

Note that under the client-server architecture, clients cannot directly communicate with each other and must communicate through a server. You will need to design a protocol for the chat program. The user interface need not be sophisticated, but should be usable. It can be text-based or graphical.

Submission Guidelines

Organizing your submission

- All relevant files should be under one directory. This directory should be named after you, e.g., ″RuiZhang″.
- Within ″RuiZhang″, you should have a file called ″README.txt″ or ″README.pdf″. The following contents are required in the README file:
  - **Protocol design**: Describe the protocol messages you defined along with their syntax and semantics, and the actions that need be taken upon sending or receiving the message.
  - **List of relevant files**: Give the list of relevant files including all source files and configuration files which you have written. Do not include any irrelevant, old, or temporary files which clutter the directory. Clean-up before submission.
  - **Compilation instructions**: How should one go about generating the executable from your source files? Give the actual set of commands which someone can cut-paste from the README in order to compile. Provide such instructions for each executable file you have to generate.
  - **Configuration file(s)**: If your code uses any configuration file(s), describe the format of such file(s) clearly. Also include in the directory some example configuration files.
  - **Running instructions**: You may be generating more than one executable. Describe logically what each executable does. In addition, for each such executable file, how should one run it? What are the command line arguments (describe each argument)?
- You may create any number of sub-directories (multiple levels too if you want) within your main directory. The README file within the main directory should describe everything: all files/directories within any sub-directory too.
- **Commenting**:
  - Each source file should describe in the beginning, in a comment, what that source file contains logically.
  - Make sure to name variables, functions, file names with intuitive names wherever possible. In addition, provide a comment for each variable/function describing it logically. You need not do this for very trivial variables/functions; use your common sense judgment. The overall objective is that a third person should be able to easily understand what that variable/function does logically.
  - Please also comment sections of the code which will help in understanding the logical flow of the code. For example, comments for a loop can describe any non-obvious invariant involved.

How and when to submit

- In the final submission, you have to tar-gzip or zip the main directory (e.g., ″RuiZhang″) and submit a single file. Make sure to tar-gzip or zip from the parent directory of this directory, not from within this directory itself. The tar-gzip or zip file should also have the same name as the directory (RuiZhang.tar.gz, RuiZhang.tgz or RuiZhang.zip in this case).
- Submit the code via Canvas before the deadline.