CS4065 Computer Networks and Networked Computing
Project 2 - A Simple Bulletin Board Using Socket Programming

Instructor: Giovani Abuaitah

1 Overview

A bulletin board system allows users to connect to it using a terminal program and perform various functions. One of the core functions of a bulletin board is to allow users to read messages posted publicly by other users. It can also be implemented in a way to allow users to join a certain group and post messages that can only be seen by users in that group. The purpose of this project is to implement a fully fledged client-server application (i.e., the bulletin board) using pure **unicast** sockets. In this project, you can write both the client and server implementations using the same programming language, or you can have the client program written in a langauge different from the one used for the server program. Additional credit ($\sim 5\%$) will be given if you choose to do the latter. You can use Java, Python, and/or C/C++. **IMPORTANT**: you **MUST** use **sockets** regardless of the language you choose (no other built-in or third party libraries/modules are allowed for networking purposes). You will work in a group of **3 students** to complete this project. I advice you to pick your partners as soon as you can if you have not done so already and **start early** on the project, there will not be any extensions given under normal circumstances. You may be required to **demo** your implementation after submission date. If a demo is required, more details about demo requirements will be posted later this semester.

The project is divided into two parts. You should start with Part 1 first. Then you can extend your code to implement Part 2. Despite the fact that each part can be implemented separately (i.e., as two separate projects), you must not separate the implementation of both parts. In other words, you must submit **one common code** that can allow both parts to work seamlessly as intended during the same process execution (For example, users should be able to switch between part 1 and part 2 without having to disconnect from the server).

1.1 Part 1: A Public Message Board

In the first part of this project, you will consider that all clients belong to one and only one public group. A client joins by connecting to a dedicated server (a standalone process) and is prompted to enter a non-existent user name in that group. Note: in this project, you are *not* required to implement any user authentication mechanisms. The server listens on a specific non-system port endlessly. The server keeps track of all users that join or leave the group. When a user joins or leaves the group, all other connected clients get notified by the server. When a user (or client) joins the group, he/she can only see the last 2 messages that were posted on the board by other clients who joined earlier. A list of users belonging to the group is displayed once a new user joins (in this part, the list represents all users/clients that have joined earlier). When a user posts a new message, other users in the same group should see the posted message. Messages are displayed in the following format: "Message ID, Sender, Post Date, Subject." A user can retrieve the *content of a message* by contacting the server and providing the message ID as a parameter. Your client program should also provide the option to leave the group. Once a user leaves the group, the server notifies all other users in the same group of this event.

1.2 Part 2: Multiple Private Message Boards

Extend Part 1 to allow users to join multiple private groups. Once a user is connected to the server, the server provides a list of 5 groups. The user can then select the desired group by id or by name. A user can join multiple groups at the same time. Remember that a user in one group cannot see users in other groups as well as the messages they have posted to their private board in other groups.

2 Some Hints

2.1 Protocol Design

Like in any client-server application, both the client and server have to communicate using an agreed-upon protocol. Protocol messages should have a format understood by the client and server alike (as an example, think about HTTP request and response messages we studied in this course and in project 1; HTTP request messages contain header information parsed and understood by the server, and response messages have body section carrying the data back to clients). You may use plain text, or consider using XML or JSON representations for all or parts of the protocol messages.

2.2 Server Implemenation

- Since you are required to use **unicast** sockets, the server should keep a list of all connected clients (i.e., all client sockets).
- Consider using mulithreading, one thread for each TCP connection. Similar to what you did in project 1 "Multithreaded Web Server". In addition, the TCP connection should always be open until the client decides to leave the group and disconnects from the server.

3 Grading (Total 50 Points)

- Functionality (70%): The program should correctly perform the tasks described above. You will risk receiving a **zero** or very low credit for the entire project if your code does not compile, or the program does not perform as expected. 40% will be given to the first part and 30% to the second part.
- Usability (15%): Your program should be user friendly. You can use special input commands to handle user's requests. For example, you can use the following set of console commands with options:
  - a %connect command followed by the address and port number of a running bulletin board server to connect to.
  - a %join command to join the single message board
  - a %post command followed by the message subject and the message content or main body to post a message to the board.
  - a %users command to retrieve a list of users in the same group.
  - a %leave command to leave the group.
  - a %message command followed by message ID to retrieve the content of the message.
  - an %exit command to disconnect from the server and exit the client program.

  For Part 2:
  - a %groups command to retrieve a list of all groups that can be joined.
  - a %groupjoin command followed by the group id/name to join a specific group.
  - a %grouppost command followed by the group id/name, the message subject, and the message content or main body to post a message to a message board owned by a specific group.
  - a %groupusers command followed by the group id/name to retrieve a list of users in the given group.
  - a %groupleave command followed by the group id/name to leave a specific group.
  - a %groupmessage command followed by the group id/name and message ID to retrieve the content of the message posted earlier on a message board owned by a specific group.

  There will be **additional points** given to a graphical user interface (**GUI**) implementation (depending on how good your GUI is - up to 5%).
- Documentation (15%): Documenting your code and using more expressive variables will be taken seriously. In addition, provide a **Makefile**, if generating the executable requires a number of steps, and a **README** file. The README should contain instructions on how to compile and run your server/client programs (if certain software or packages need to be installed, provide instructions as well), usability instructions **ONLY** if different from suggested above, and a description of any **major issues** you came across and how you handled them (or why you could not handle them).

4 Submission Instructions

Add your name and your partner's name at the very top of the README file, place both your client and server source code, README, and a Makefile if any, in one directory (name it Project2-Lastname-PartnerLastname), zip it, and submit to Canvas by the due date (Please do NOT submit binaries).