200 points. Individual Work Only. Due February 7, 2023 before 12:30 PM.

1. Objective

Design and implement a multi-threaded client-server program in Java using sockets for communication between the client and server programs.

2. Description

The server program performs the following operations:

1. Opens a socket connection on the specified port (port # is provided as a command-line argument to the server program)
2. Listen for request from clients
3. When a client program sends a connect request, assigns a new port for the client to communicate, creates a new thread that will handle all the client requests, updates the list of users currently connected to the server, and continues to listen for request from other clients (step 2)
4. The new thread created opens a socket connection on the port # assigned by the server, listens to subsequent requests from the client, logs all communication (to a file) between the client and server, and responds accordingly (this thread will terminate when the client program sends a disconnect request).

The client program takes the hostname and port # on which the server process is listening along with the userid of the client as command-line arguments and performs the following operations:

1. Open a socket connection to connect with the server process (the first client request will be connect)
2. If the connection is successful, the server returns on new port # for the client to connect for all further communications. Close the first connection and open connection to the new port provided by the server.
3. Provide a command prompt for user for enter commands and wait for the user to enter commands/messages
4. Process the commands entered by the user
5. Send the appropriate command/request to the server process
6. Wait for the server to perform the appropriate action and respond to the client request
7. Process server response if necessary
8. Go to step 3, if the input is “QUIT” then send the request disconnect to the server and exit.

The various commands that the client and server can process along with the action performed at the server is given below:

1. CONNECT – Establish connection with the server, server assigns a new port for the client, creates a new thread, and returns the new port # for the client to connect (if there is any error with assigning a new port, or creating a new thread, the server should return an error code (say, a negative number) – use a different number for each error condition)
2. LIST – List users currently connected to the server
3. QUIT – client program exits after closing any open connections and the corresponding server thread is also terminated (this is the thread created by the server for this client)
4. SEND – The client send the messages entered by the user
5. BROADCAST – The server sends the message it received from a client to all other clients

The server keeps a log of when the server started along with all the messages, userid, and timestamp for each message one per line in the format *timestamp: userid: message*.

The server process must be started before starting the client process. For this assignment you can start the server and client process on the same machine. Choose a port number greater than 8000 for the server process and a port range greater than 8000 for client processes. A sample output is provided for a simple client-server session below (user input is highlighted):

$ java ChatServer
Server started at Tue Jan 17 10:40:23 CDT 2023, waiting for connections…
User1 Connected at Tue Jan 17 10:40:53 CDT 2023
User 2 Connected at Tue Jan 17 10:42:31 CDT 2023
……..
User 1 Disconnected at Tue Jan 17 10:54:07 CDT 2023
……..

$ java ChatClient localhost 90000 User1
(Server) Welcome User1, there are 1 user(s): User1
(User1) ***Hello any one out there?***
(Server) User 2 joining now
(User2) Hello User1, How are you?
(User1) ***Hello User2, I am glad to hear from you***.
(Server) User 3 joining now
(User3) Hello All, How are you?
(User1) LIST
(Server) Active users: User1, User2, User3
(User1) ***I am glad to talk to both of you***
……..
(User1) ***EXIT***
$

$ java ChatClient localhost 90000 User2
(Server) Welcome User2, there are 2 user(s): User1, User2
(User2) ***Hello User1, How are you?***
(User1) Hello User2, I am glad to hear from you.
(Server) User 3 joining now
(User3) Hello All, How are you?
(User1) I am glad to talk to both of you
……..
(Server) User1 Disconnected, there are 2 user(s): User2, User3
……..

$ java ChatClient localhost 90000 User3
(Server) Welcome User3, there are 3 user(s): User1, User2, User3
(User3) ***Hello All, How are you?***
(User1) I am glad to talk to both of you
……..
(Server) User1 Disconnected, there are 2 user(s): User2, User3
……..

3. Guidelines/Hints

First follow the Java Tutorials for sockets and threads (see Resources in Section 10 below) and learn the Java APIs for sockets and threads. Initially, implement the client and server program without any socket code. Use Java packages to separate the client and server programs (*e.g.,* cs591.hw1.server and cs591.hw1.client). Make sure you can parse all the input commands correctly on the client side, and then test the basic functionality on the server side. Then write the sockets code and test it first without threads. Finally, add threads and synchronization to complete

the program (you must design your program from the start with multi-threading in mind, if you try to add threads at the end you will end up modifying your code). Make sure that your program is modular and well designed; you will use this program for subsequent homework assignments.

4. Graduate Students Only (Bonus for Undergraduates)

Logging described above is optional for undergraduate students but required for graduate students.

5. Working Environment

You can develop and test the client and server programs on any machine that you have access to. You must demonstrate the working program to the instructor to get full credits for this assignment.

6. Short Answer Questions

Answer the following questions and submit your answers as a separate Word or PDF file.

1. If student A has the server program running on machine P, could student B use his client program running on machine Q to connect to A's server program if student B knows the port on which student A's server is running?
2. If your answer to question (1) is yes, then explain how to prevent this from happening.
3. If your answer to question (1) is no, then explain why this is not possible.
4. Why do you need to use a port # > 1024?

7. Feedback Questions (answer to these questions has no impact on your grade)

Was this homework too difficult, or too easy?
Was the assignment fun or challenging?
Was there something that was unclear?
Was the homework too long for the given amount of time?
What did you learn from this homework?

8. Submission Instructions

List ALL the references you used in this homework as well as test cases used to test your programs. This includes any classes that you used that you did not write and any help you received from any other sources. Use appropriate class name and include comments to indicate various operations performed by the program. Your program must have the following header information within comments:

```
/*
        Name:
        Section: CS 491 or CS 591
        Homework #:
*/
```

Make a directory CS591/homework1 and then two separate directories one for the server (server) and one for the client program (client). Include a README.md file in the homework1 directory that provides the instructions for executing the client and server programs.

Create a github repository (say, Spring2023_CS691_HW1) and upload your source files to the github repository. Make sure that you have created a **private repository/project** (click on visibility to be private). Add pvbangalore@ua.edu as a collaborator to this project. If you are not familiar with github follow the instructions given in Section 11 at the end of this document to create a github account and create a new project.

Create a tar/zip file (only tar and zip are accepted, all other file formats will not be considered) of the homework1 directory using the following filename format: <crimsonId>-cs491-hw1.tar or <crimsonId>-cs591-hw1.zip. Upload the tar/zip file that includes the source code and instructions for executing the programs and a separate Word/PDF document for the typed solution to short answer questions and the feedback questions. In the comment section of the homework submission, include the link to your github repository for this homework.

9. Late Submissions

Submissions must be made on the due date before the beginning of the class. Any submissions received after the due date will receive a score of 0 for this homework.

10. Resources

1. For Packages: Java Tutorials: Creating and Using Packages: http://java.sun.com/docs/books/tutorial/java/package/packages.html
2. For Sockets: Java Tutorials: All About Sockets: http://java.sun.com/docs/books/tutorial/networking/sockets/index.html
3. For Threads: Java Tutorials: Concurrency: http://java.sun.com/docs/books/tutorial/essential/concurrency
4. General Java Help: Java Tutorials: https://docs.oracle.com/javase/tutorial/index.html

11. Instructions for creating github repository

1. Create a github account at https://www.github.com using your **crimson email address**.
2. Create a new repository (say, *Spring2023_CS591_HW1*) on the github server. Make sure that you have created a private project (click on visibility to be private).
3. You can use the browser to either add files or upload files from your local machine. If you are using a terminal, you can execute the following commands in a terminal (assuming you have the solution as *hw1.c* in the directory *CS591/hw1* and repository name is *Spring2023_CS591_HW1*; use your github username instead of <githubid>):

   cd CS691/hw1
   git init
   git remote add origin https://github.com/<githubid>/*Spring2023_CS591_HW1*.git
   git add **.**
   git commit –m “Initial check in”
   git push -u origin master

4. After this if you make changes to the file hw1.c you can update the repository using:

   git commit -m "Appropriate commit message"
   git push origin master

5. Click on the setting tab, click on Manage Access on the left side, and then click on the green box in the middle of the screen “Invite a collaborator.” Enter pvbangalore@ua.edu as the email address and click on “Add …” button.

The URL above (https://www.github.com/<githubid>/*Spring2023_CS591_HW1*.git) will be different for you depending on your github id and repository names, you can get the URL by selecting HTTPS and clicking on the copy icon next to the URL when you click on the green “Code” button.