CSCI 5531: Advanced Operating Systems

Summer 2023
100 pts

Project-1 [KSH]

Due Date: June 27, 2023

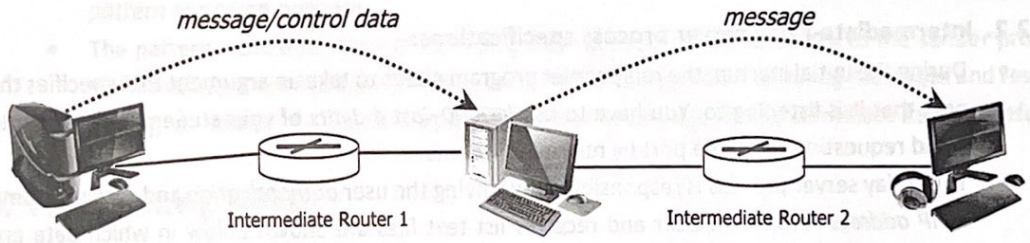
A Distributed Message Relay System

1. Objective

This first programming project aims to learn TCP iterative client-server interaction using a socket interface in Java programming language. After completing the project, you will understand the steps required to develop a networking application.

2. Project Specification

In this programming project, you are required to implement a distributed message delivery client-server application using Java language. The sender process accepts user-entered "messages" from the keyboard and sends these messages to the receiver process via the message-relay process. The receiver process displays the message and performs a few computations once the message has been received entirety. All communications among the processes are done using TCP sockets. Two application-level messages are defined: a DATA message (containing the text of a message) and a CLOSE message, which instructs the receiver of the CLOSE message to shut down. The relay process diagram and more specific project description are given below:



2.1. Sender process detail specifications:

- The sender process establishes a TCP connection with the relay server. Your sender program needs to take *two arguments* that specify the *relay server's IP address and port number* that it will try to connect.
- After a successful connection, your sender program will first prompt a welcome message that asks the user to key in a username (keyboard). This username will then be sent to the relay server. After receiving the username from your sender, your relay server will send an *acknowledgment message* back to the sender.
- Your sender, after receiving the acknowledgment message from your relay server, will prompt a message that asks the user to enter the corresponding password. This password will then be sent to the relay server. Then, your server, after receiving the password from the sender, will verify the received pair of username and password against the list of legitimate pairs. If the result is positive, the server will send a success message to the sender. If the result is negative, the server will send an error message to the sender and wait for the new pair.

- After successful authentication, your sender program will take the *name* of a receiver from the user (using the keyboard). The receiver name will be sent to the relay server to verify the received name against the list of legitimate receiver names for selecting the receiver. If the result is positive and a connection between the relay server and receiver has been set up using the IP address and port number of the receiver, the server will send a successful message back to the sender. If the result is negative, the server will send a failure message to the sender and wait for a new pair of name and port number to connect with the receiver.
- When connections between the sender, relay server, and receiver are successfully completed, the sender process prompts the user for "messages" (e.g., containing characters entered by the user via keyboard) and sends each message to the relay and waits for a reply from the relay server. Note that because of the byte-stream semantics of TCP sockets, your sender process will need to define an application-layer DATA message that lets the relay process know how many characters are in the incoming byte stream.
- When the messages are sent from the relay server to the receiver, the receiver will apply a simple algorithm to measure the *longest common sub-string* from the messages along with the length of the sub-string. This result will be sent back to the sender process. The sender process needs to print the result into the output.
- Finally, when the user wants to indicate that there are *no more messages* to send, the sender process should send a CLOSE message to the intermediate relay, and then terminate itself gracefully (i.e., releasing any sockets it has been using). Users may type 'x' or 'q' to terminate the sender program.

2.2. Intermediate-relay server process specifications:

- During the initial startup, the relay server program needs to take an argument that specifies the port that it is listening to. You have to use (*10000+last 4 digits* of your student id number) to avoid requesting the same port by multiple students.
- Your relay server process is responsible for verifying the user *authentication* and receiver name to *IP address resolution*. User and receiver list text files are shown below in which data are separated by a black space (i.e., between user name and password there is a blank space to separate them) and each line is separated by a newline:

userList.txt

Username	Password	Role
Anna	a86H6T0c	Client
Tiffany	G6M7p8az	Client
Cathie	Pd82bG57	Client
Marise	jO79bNs1	Client
Tran	uCh781fY	Client
Farah	Cfw61RqV	Client
Shaz	Kuz07YLv	Client

receiverList.txt

Receiver Name	IP Address	Port Number
Sapphire1.uhcl.edu	129.15.00.01	ServerPort+1
Sapphire2.uhcl.edu	129.15.00.02	ServerPort+1
Sapphire3.uhcl.edu	129.15.00.03	ServerPort+1
Sapphire4.uhcl.edu	129.15.00.04	ServerPort+1
Sapphire5.uhcl.edu	129.15.00.05	ServerPort+1

- After establishing a connection with the sender and receiver processes, your server process reads messages sent by the sender process. It forwards DATA messages to the receiver process over a TCP socket.
- Similarly, your server process reads messages sent by the receiver process. It forwards the resulting data messages to the sender process over a TCP socket.
- When the intermediate relay server receives a CLOSE message from the sender process, it should send a CLOSE message to the receiver process so that the receiver process can gracefully terminate itself.
- The intermediate relay server process continues to run regardless of the CLOSE message send by the sender process.

2.3. Receiver Server process specifications:

- During the initial startup, your receiver program will take one argument that specifies the port number that it is listing to. You have to use your *relay server port number + 1* as the receiver port number. As you can see, your receiver program acts more like a server.
- This process reads messages sent by the intermediate-relay process. It displays the content of each DATA message once that message is received entirely.
- Next, your receiver program requires you to implement a simple *pattern-matching algorithm* to match and count the occurrences of the received pattern (word) from a receiver-side text file (e.g., Story.txt). *Note: you may refer to wikipedia for a detailed description/algorithm of the pattern matching problem.*
- The pattern word and count of pattern matching needs to be sent back to the sender process via the relay server. It's a good idea to use structures/classes for handling messages and results.
- Finally, on receipt of a CLOSE message, the receiver process should terminate itself gracefully.

3. Programming Notes:

I would strongly suggest that everyone begin by writing the sender and relay processes first, i.e., just getting the two of them to interoperate corrections. Then modify the relay server and write the receiver process.

You will need to know your machine's IP address when one process connects to another. You can telnet to your own machine and see the dotted decimal address displayed by the telnet program. You can also use the UNIX *ipconfig* or *ping (e.g., ping gpel2.cs.ou.edu)* commands to figure out the IP address of the machine you are working on. You can also ask your local system administrator for the info.

Many of you will be running the sender, relay, and receiver on the same UNIX/Windows machine (e.g., by starting up the receiver and running it in the background, then starting up the relay in the background, and then starting up the sender. This is fine; since you are using sockets for communication, these processes can run on the same machine or different machines without modification. If you need to kill a process after you have started it, you can use the kill command. Use the ps command to find the process id of your server.

Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use though it is prohibited.

You should program each process to print an informative statement whenever it takes an action (e.g., sends or receives a message, detects termination of input, etc.), so that you can see that your processes are working correctly (or not!). One should be able to determine from this output if your processes are working correctly. You should hand in screenshots (or file content, if your process is writing to a file) of these informative messages.

Note that we will be ***using our Delta lab machines (Unix/Windows platform)*** in the lab to run and verify your codes. Since the goal of the programming project is for you to learn socket programming (not to build a hardened application), you may also assume that the relay and the receiver will only be receiving messages from a single sender or relay, respectively. You are required to work on this project alone. Any clarifications and revisions to the assignment will be posted on the course UHCL BB page.

3.1 File names:

Make sure you follow the file name guideline given below for your project:

lastNameP1Sender.java, lastNameP1Receiver.java, lastNameP1Server.java, userList.txt, and receiverList.txt

4. Points Distribution:

Bits and pieces	Points
Client Program	20
Relay Server	30
Receiver Program	30
Program Style (Coding style, comments etc.)	10
Documentation	10

5. Submission Instructions:

This project requires the submission of a *soft copy* and a *hard copy*.

5.1. Soft Copy (Due June 27, 2023, 11:59 pm)

The soft copy should consist of the sender, relay server, receiver, and any other API file(s) along with the detailed documentation of the programs. Documentation should include your problem-solving approach, discussion of data structures, algorithms, user define functions/methods, and screenshots of outputs.

5.2. Hard copy (Due June 27, 2023, beginning of the class)

The hard copy of the project should be submitted at the beginning of the class. It should include the documentation of the project followed by the source code of the sender, server, and receiver. It's also required to include the screenshot of outputs in the hard copy. You do not need to submit the user list and receiver list files. As we will be using our own user and receiver list files, you have to make sure that your program works with the file format given in this project description. The source code should be very well commented and the documentation should be detailed as well to get the full credit.

6. Late Penalty:

You have to submit your project on or before the due date to avoid any late penalty. **A late penalty of *15% per day* will be imposed after the *due date (June 27, 2023)***. After one week from the due date, you will not be allowed to submit the project under any circumstances.

Please **DO NOT MODIFY** the project code in any way after the deadline for soft copy submission. The hard copy and the soft copy must be same.

Good Luck!!

10,000+
uhcl-id (last 6 numbers)