Cauldierre Mckay, Curtis Nelson, and Matthew Turczmanovicz

NSSA.290 Networking For Developers

Professor Warren

May 4th, 2017

NSSA.290 Networking for Programmers – Homework 3: Part 2 Documentation

Table of Contents

* The project has been broken down into two components: TCP/IP and UDP. The TCP/IP component consists of a TCP/IP based Server and Client GUI. Likewise, the UDP component consists of a UDP based Server and Client GUI.
* TCP/IP Component – Pages 2 & 3
* UDP Component – Pages 4 & 5

TCP/IP Component

Programs required to run the programs?

* The Windows Command Prompt will be required to run the programs.
* If you would like to view the source, open the .java files in Notepad++ or any Java IDE of your choice, like jGRASP.

How to compile the Programs?

* The .java files are already compiled, and ready to run. The compiled files are the .class files in the TCP/IP Chat folder. These come included with the .java files in the folder. If you would like to recompile the .java files, open the .java files in a Java IDE and compile. For jGRASP, open the .java files and click the green plus sign at the top of the toolbar area to compile the file.

How to run the application?

* The application can be run through the Windows Command Prompt. On the computer that you want the TCP/IP server to be, open the Command Prompt. Change directories to the folder that contains both the .java and .class files for the Chat TCP/IP Server. Once there, enter in the Command Prompt the following command:
  + java ChatServerTCPIP
* If the .java file name for the Server has changed, then replace “ChatServerTCPIP” with the new name of the file. Furthermore, if the .java file names have been changed, recompilation will be required.
* This starts up the TCP/IP Server on this computer.
* Select another computer(s) for the Client(s).
* Once again, navigate to the proper folder in the Windows Command Prompt. Once there enter the following command:
  + java ChatGUITCPIP \*Server IP Address\*
* Once you run the Server, a GUI will appear with the Server’s IP Address. This is the IP Address that must be entered with the command above. Example:
  + java ChatGUITCPIP 10.100.100.08
* If the Server is running properly, and the Client(s) was started properly, then the Chat Application should function as expected. Enter a message in the Message text area. Click the Send Button to send the message. That’s it! Once you are done, close the Client by clicking on the top, right close button. Once you are done with the Server entirely, click the top, right button on the Server’s GUI that displays the IP Address.

How does it work?

* The Server itself will connect a ServerSocket to a Port number and loop for incoming Clients that want to connect to this IP Address and Port number. Once the Client(s) make a successful connection, the Server will accept the Client and create a Thread to run the Client’s interactions. The new Client is added to a collection of PrintWriters. This thread creates IO reader and writer objects. These are used to read in messages from a Client and send the message to all Clients that are connected. This is performed through the IO streams. Once a message is received, then the Server will go through its collection of Clients and send the received message to all of the connected Clients. The message from the Client is received through the BufferedReader Input Stream. The message is then sent to all Clients through the PrintWriter Output Stream.

The Client operates through Java Swing to create the GUI. The Client connects by entering the IP Address of the Server. If the Client is accepted by the Server, the Client will create IO reader and writer objects as well. In addition, the Client has a separate thread that focuses on receiving messages. This is to ensure the messages can be sent and received simultaneously. The Client also implements ActionListener. This gives the GUI’s Send Button the ability to send messages to the Server. The text is retrieved from the message area and it is sent to the server using the PrintWriter Output Stream. The message is received from the Server using the BufferedReader Input Stream. This String message is then appended to the conversation text area. View the code in Notepad++ or a Java IDE of your choice to discover more.

UDP Component

Programs required to run the programs?

* The Windows Command Prompt will be required to run the programs.
* If you would like to view the source, open the .java files in Notepad++ or any Java IDE of your choice, like jGRASP.

How to compile the Programs?

* The .java files are already compiled, and ready to run. The compiled files are the .class files in the UDP Chat folder. These come included with the .java files in the folder. If you would like to recompile the .java files, open the .java files in a Java IDE and compile. For jGRASP, open the .java files and click the green plus sign at the top of the toolbar area to compile the file.

How to run the application?

* The application can be run through the Windows Command Prompt. On the computer that you want the UDP server to be, open the Command Prompt. Change directories to the folder that contains both the .java and .class files for the Chat UDP Server. Once there, enter in the Command Prompt the following command:
  + java ChatServerUDP
* If the .java file name for the Server has changed, then replace “ChatServerUDP” with the new name of the file. Furthermore, if the .java file names have been changed, recompilation will be required.
* This starts up the UDP Server on this computer.
* Select another computer(s) for the Client(s).
* Once again, navigate to the proper folder in the Windows Command Prompt. Once there enter the following command:
  + java ChatGUIUDP \*Server IP Address\*
* Once you run the Server, a GUI will appear with the Server’s IP Address. This is the IP Address that must be entered with the command above. Example:
  + java ChatGUIUDP 10.100.100.08
* If the Server is running properly, and the Client(s) was started properly, then the Chat Application should function as expected. Enter a message in the Message text area. Click the Send Button to send the message. That’s it! Once you are done, close the Client by clicking on the top, right close button. Once you are done with the Server entirely, click the top, right button on the Server’s GUI that displays the IP Address.

How does it work?

* The Server connects to a Datagram socket using a Port number. Once it’s connected and running, it will loop for incoming Clients. Once a thread connects to the ServerSocket, a new thread is created for this new Client and the thread is started. The thread assigns a DatagramSocket to the Socket that was passed through when the Client connected. The thread contains a collection of IP Addresses of the Clients who have connected to the Server. In a loop, the thread waits for the DatagramSocket to receive a packet from the connected Client. Once it’s received, the packet is turned into a String message. This is the text that was sent to the Server by the Client. The packet has its IP Address inspected. If the IP address doesn’t exist in the collection, it’s then added to the collection of Clients. After this check, the Server goes through its collection of Clients and converts the message that it received into a packet. This packet is sent to each Client in the collection.

The Client extends the JFrame in order to create a Swing GUI. The user must supply the IP Address of the Server in order to connect to the Server. This is entered in the command line as an argument. This IP Address is displayed when the Server starts. The Client makes a DatagramSocket that connects to the Server’s Port number. The Client has a separate thread used for receiving messages. This ensures that the Client can send and receive messages simultaneously. The Client implements ActionListener so the Button can send messages when it is clicked. The message that is entered in the text box is retrieved and it is turned into a packet. The packet is sent to the server using the Server’s Port number and the Server’s IP Address. The thread for receiving messages acts similarly to the Server. The socket loops to receive packets and once it receives a packet, it will turn the packet into a String message. This message is the text that was sent to it by the Server. This message is then appended to the conversation text area. View the code in Notepad++ or a Java IDE of your choice to discover more.