# Socket Programming:  Multi Client Chat Program

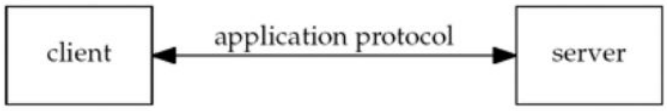
Sefiyetu Abdullah
Computer Science
Virginia State University

This paper presents a multi client chat program that allows multiple different clients to chat with a server at the same time. Socket programming allows us to connect two nodes and share information between them, it is the basis of this project. We applied the basics of socket programming to create a common chat program that is the foundation of many popular messaging software systems on the market today. The system effectively sent and received responses from various clients.

## INTRODUCTION

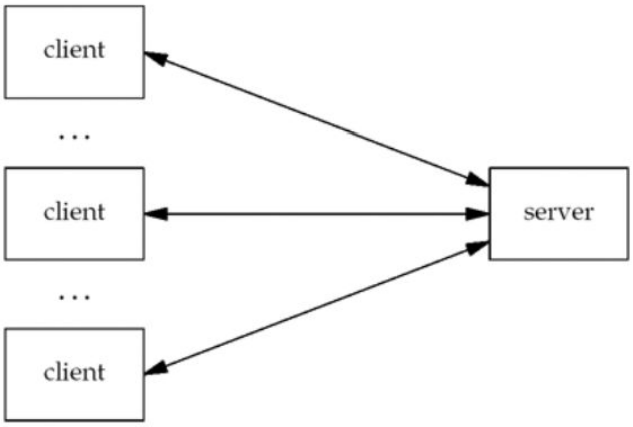
Socket programming allows two nodes to connect and share information. There are two different sides, the client and server side. During the creation of a socket, the server and client share their IP addresses and TCP port information. The connection of nodes allows the client and the server to send information back and forth. This is reliable and standard, as shown in Figure 1.1.

**Figure 1.1. Network application: client and server.**




For this program to run effectively, the server must be able to handle multiple clients simultaneously, as shown in Figure 1.2.

**Figure 1.2. Server handling multiple clients at the same time.**




While clients and servers can be connected on different LANs, this project has them connected on the same one for ease of access.

## Materials & Method

This project uses JAVA to code the chat program, due to its status as a popular and well known high level language. It is easy to understand and compile on any machine. This code was run on the windows command prompt, with the client program and the server program being run in different application windows simultaneously on the same machine. Both programs must be run simultaneously in order to connect the sockets. If the server program does not detect the client program, it will continue to say "Waiting for Connection..." and will not run.

Example 1 will show how the server interacts with a single client, example 2 will show multiple clients.

**Example 1**

After running the code, the client program prompts the client to type a message or type "Bye" to end the program, as seen below.

```
C:\myproject>java client
Type Message ("Bye." to quit)
Hello!
Server : Hello Client, how are you?
I am doing wonderful, thank you for asking. How is the weather over there?
Server : Absolutely fantastic
I have to go, but I will check in with you later
Server : Thats fine! I will be here. Bye!
Bye.

C:\myproject>
```

The server side of the program waits until the client side is detected, then establishes a connection and prompts the server to wait until a message is received in order to reply. It should be noted that the client is identified by their IP Address and TCP port, allowing multiple clients to be differentiated by their identifiers.

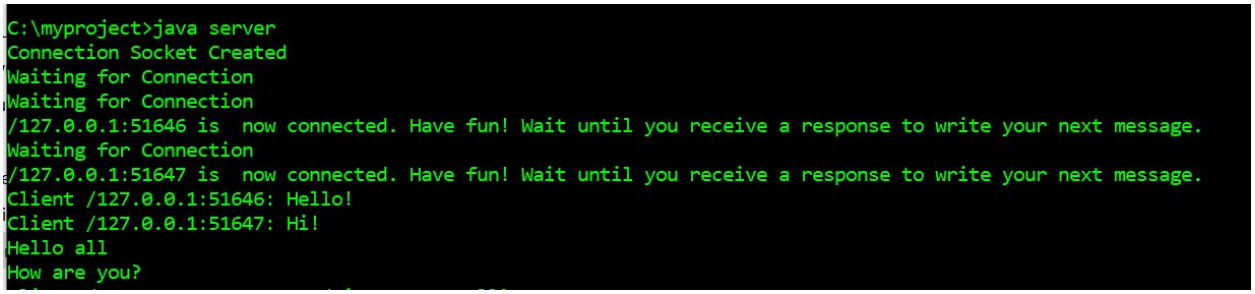
```
C:\myproject>javac server.java

C:\myproject>java server
Connection Socket Created
Waiting for Connection
Waiting for Connection
/127.0.0.1:51527 is  now connected. Have fun! Wait until you receive a response to write your next message.
Client /127.0.0.1:51527: Hello!
Hello Client, how are you?
Client /127.0.0.1:51527: I am doing wonderful, thank you for asking. How is the weather over there?
Absolutely fantastic
Client /127.0.0.1:51527: I have to go, but I will check in with you later
Thats fine! I will be here. Bye!
Client /127.0.0.1:51527: Bye.
```

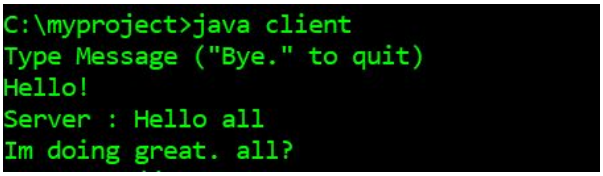
**Example 2**

In the following example, the server interacts with two different clients. The process is the same as example 1.

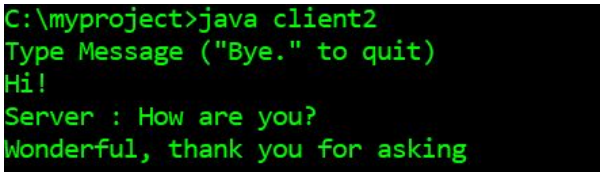
The server perspective:




Client 1 perspective:




Client 2 perspective:




## Results and Discussion

This project ran smoothly and was effective for its purpose. It would be better if the server included identifiers for which client it was speaking back to. Typically the top answer would go to Client 1 and the bottom answer would go to Client 2. This could get confusing if too many clients are connected to the server at the same time. It would also be easier for user experience purposes to allow the clients to define their own usernames.

In the future it would be interesting to take this concept to the next level by allowing the clients to chat with the server as well as themselves.

## Conclusion

This project used Java to create a multi client chat program that allowed multiple different clients to chat directly with the server using socket programming. The program differentiated the clients using unique identifiers and the server sent and received messages efficiently.

## References

[1] Stevens, W. Richard., "*UNIX Network Programming,"* Addison-Wesley, (2004).