**PES University, Bengaluru**

**Computer Science & Engineering Department**

**Session: Aug – Dec, 2019**

**Computer Networks Lab – UE17CS304**

**Mini Project**

**Fifth Semester**

**Title of the project: Creating an Asynchronous Multithreaded Chat Application**

**Team members:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. #** | **Name** | **SRN** | **e–mail id** |
| **1** | **Aishwarya Pramod** | **PES1201700770** | **aishwarya.1999@gmail.com** |
| **2** | **Saioni Chatterjee** | **PES1201700118** | **saionichatterjee1999@gmail.com** |
| **3** | **Sujeeth Anil** | **PES1201700958** | **sujeeth.anil@gmail.com** |

**ABSTRACT:**

In this project we create an Asynchronous messaging application using python. Asynchronous means both client and server can independently send each other messages without waiting for response from the other side. We used the concept of multithreading to implement sending and receiving text messages between the processes with the help of DatagramPackets. Datagrams are bundles of information passed between applications. Once the datagram has been released to its intended target, it is independent and there is no assurance that it will arrive or even that some application will be there to receive it.

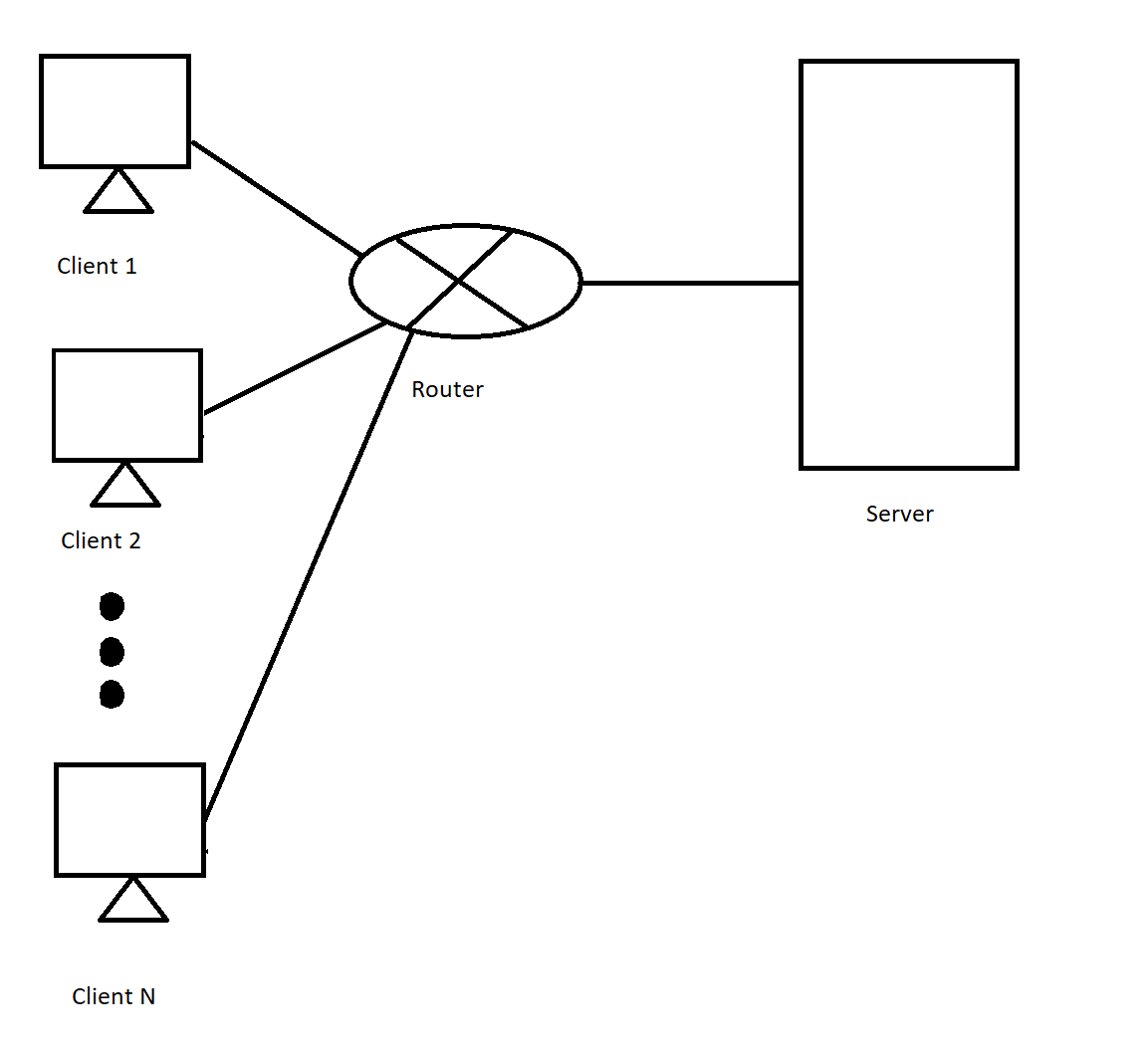
**INDEX:**

|  |  |
| --- | --- |
| **Introduction** | **4** |
| **Requirements Specification** | **5** |
| **Implementation Details** | **6** |
| **Screen Shots** | **7** |
| **Results** | **8** |
| **Conclusion** | **9** |

**INTRODUCTION:**

The aim of the project is to create a platform where multiple users can chat where all are independent of each other. A chat server is created through which incoming requests can be received from clients wanting to communicate. For this sockets and multithreading are being used. We will be using TCP sockets for this purpose, and therefore we use AF\_INET and SOCK\_STREAM flags. We use them over UDP sockets because they’re more telephonic, where the recipient has to approve the incoming connection before communication begins.

**TOPOLOGY:**

****

**REQUIREMENTS SPECIFICATION:**

1. Socket: A socket is used in a client-server application framework. It allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor.
2. Threading Module: Python threading allows you to have different parts of your program run concurrently and can simplify your design. Using threading in programs help to make the design cleaner and easier to reason about.
3. Tkinter: Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

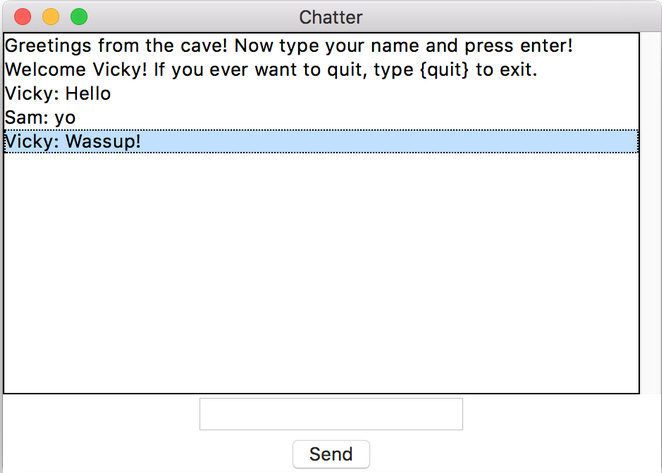
**IMPLEMENTATION DETAILS:**

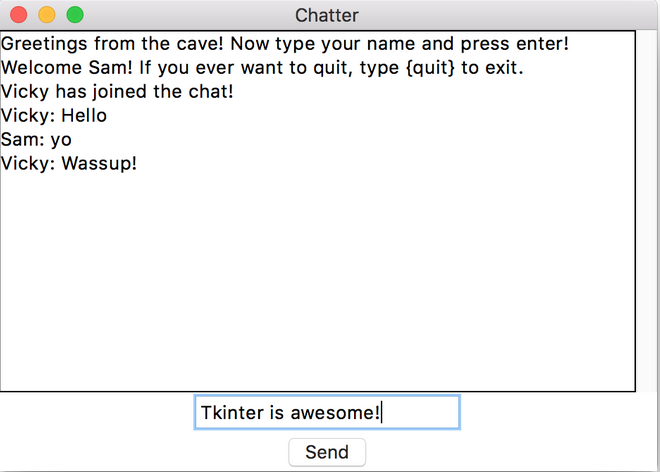
We will be using TCP sockets for this purpose, and therefore we use AF\_INET and SOCK\_STREAM flags. So we receive further messages from the client and if a message doesn’t contain instructions to quit, we simply broadcast the message to other connected clients (we’ll be defining the broadcast method in a moment). If we do encounter a message with exit instructions (i.e., the client sends a {quit}), we echo back the same message to the client (it triggers close action on the client side) and then we close the connection socket for it.

This is just a loop that waits forever for incoming connections and once it does it logs the connection, it stores the client’s address in the addresses dictionary and later starts the handling thread for that client. In the handle\_client() function, the name is saved and then send another message to the client about the instructions. In the main loop for communication we receive further messages from the client and if a message doesn’t contain instructions to quit, we simply broadcast the message to other connected. If we do encounter a message with exit instructions, we send back the dame message to client and then close the connection socket for it. We then do some clean up by deleting the entry for the client, and finally inform other connected users that this particular person has left the conversation.

The recv()function is the blocking part. It stops execution until it receives a message, and when it does, we move ahead and append the message to msg\_list. we extract the message to be sent using msg=my\_msg.get(). After that, we clear the input field and then send the message to the server, which, as we’ve seen before, broadcasts this message to all the clients (if it’s not an exit message). If it is an exit message, we close the socket. This sets the input field to {quit} and then calls send(), which then works as expected. Later we create a GUI for this application.

**SCREENSHOTS:**

****

****

**RESULTS:**

We can see how the server and the client architecture can be implemented in order to create a logical communication between multiple clients.

The screenshots above and implementation depict the way multiple users communicate with each other with no dependency after establishing a connection among them with the server via sockets.

**CONCLUSION:**

The major learning in this project is we came to know how one of the real-world applications of computer networks which is texting works and how all the clients communicate with each other. The server and the client can be running on the same machine for testing but seeing the communication happen in real time among different computers is great.