

2013CSB1021 - Pradeep Kumawat
2013CSB1027 - Rehmat Aulakh
2013CSB1030 - Ali Jafri

Group Name: G2

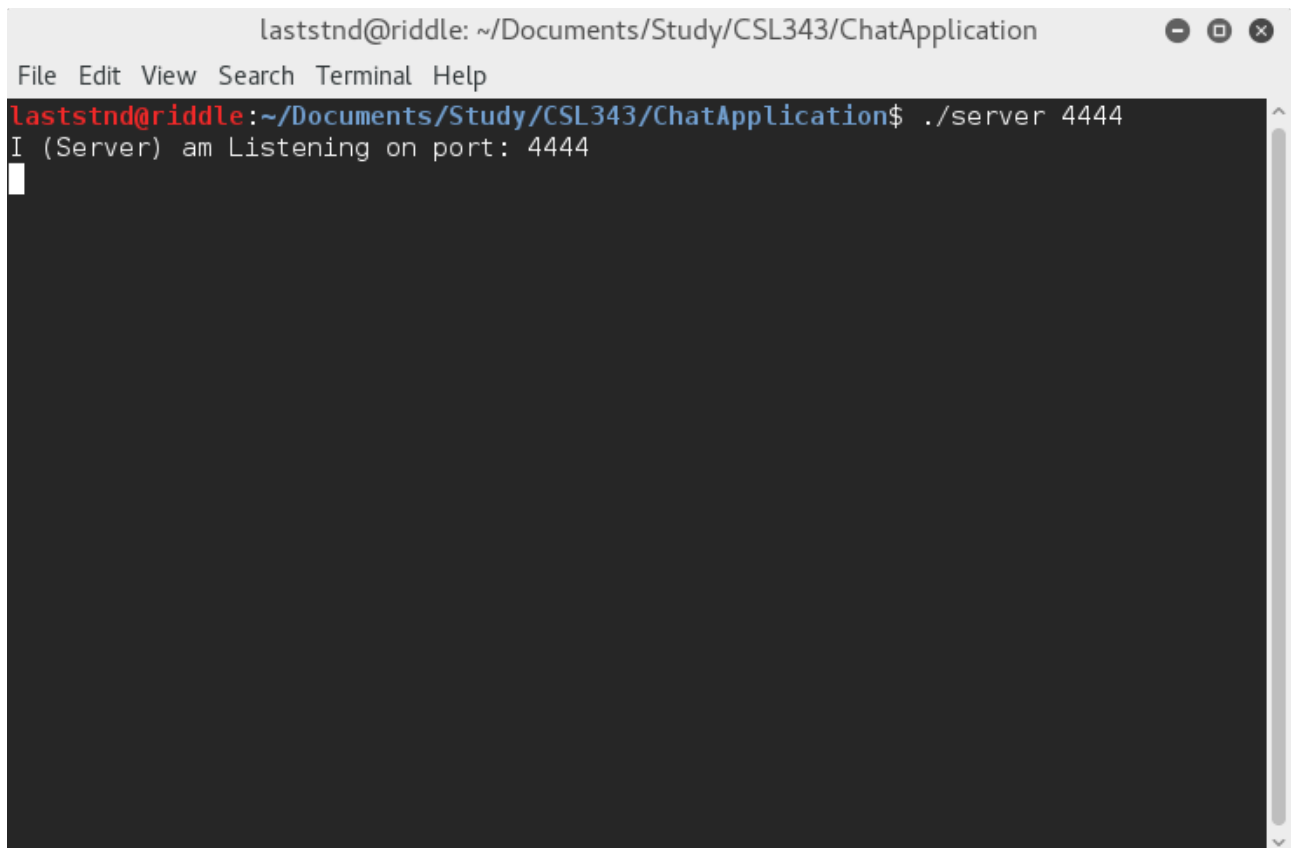
Problem Statement: Create a Chat Application that allows a client to connect to chat server and thus establishes a two way communication using sockets

Methodology:

- We have used Sockets **<sys/socket.h>** available in C Language to first set up a server that accepts connection from clients on any machine (including localhost) using the command:

\$/server <port_no>

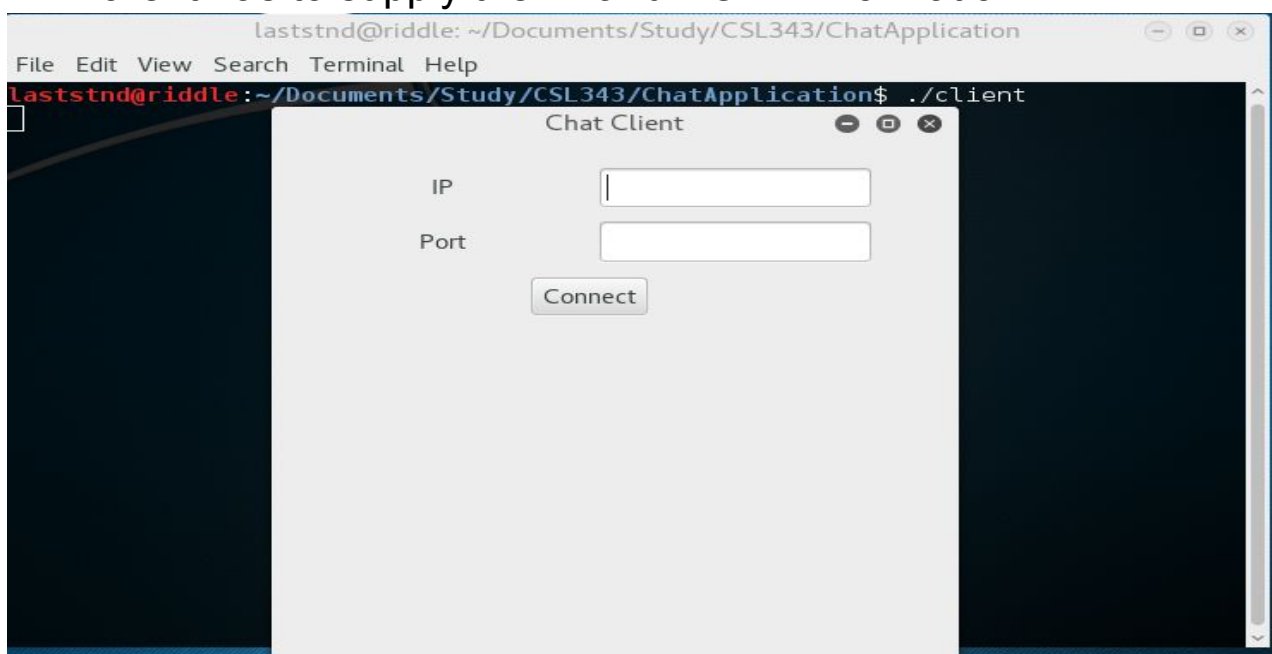
e.g. **\$/server 4444**

A screenshot of a terminal window titled "laststnd@riddle: ~/Documents/Study/CSL343/ChatApplication". The terminal shows the command `./server 4444` being executed, followed by the output `I (Server) am Listening on port: 4444`. The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The background is dark, and the text is light-colored.

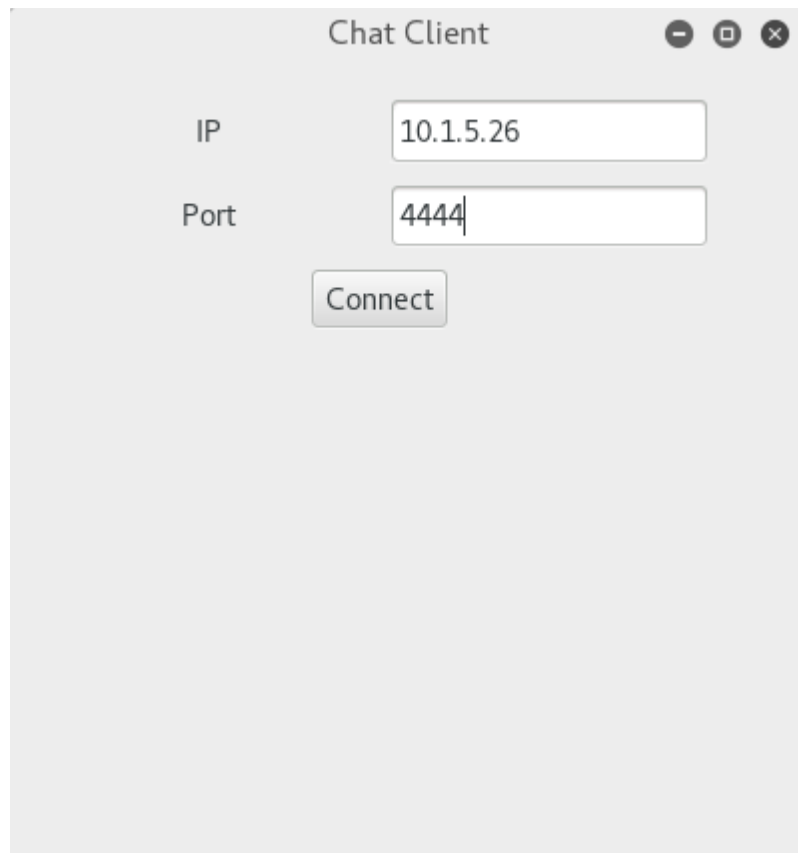
- The **IP** of the server was found by **`\$ip addr`** command on linux

```
laststnd@riddle: ~/Documents/Study/CSL343/ChatApplication
File Edit View Search Terminal Help
laststnd@riddle:~/Documents/Study/CSL343/ChatApplication$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:26:8a:db:dd:94 brd ff:ff:ff:ff:ff:ff
    inet 10.1.5.26/22 brd 10.1.7.255 scope global dynamic eth0
        valid_lft 425146sec preferred_lft 425146sec
    inet6 fe80::c191:c9af:d06a:75b7/64 scope link
        valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 48:d2:24:3a:c5:23 brd ff:ff:ff:ff:ff:ff
laststnd@riddle:~/Documents/Study/CSL343/ChatApplication$
```

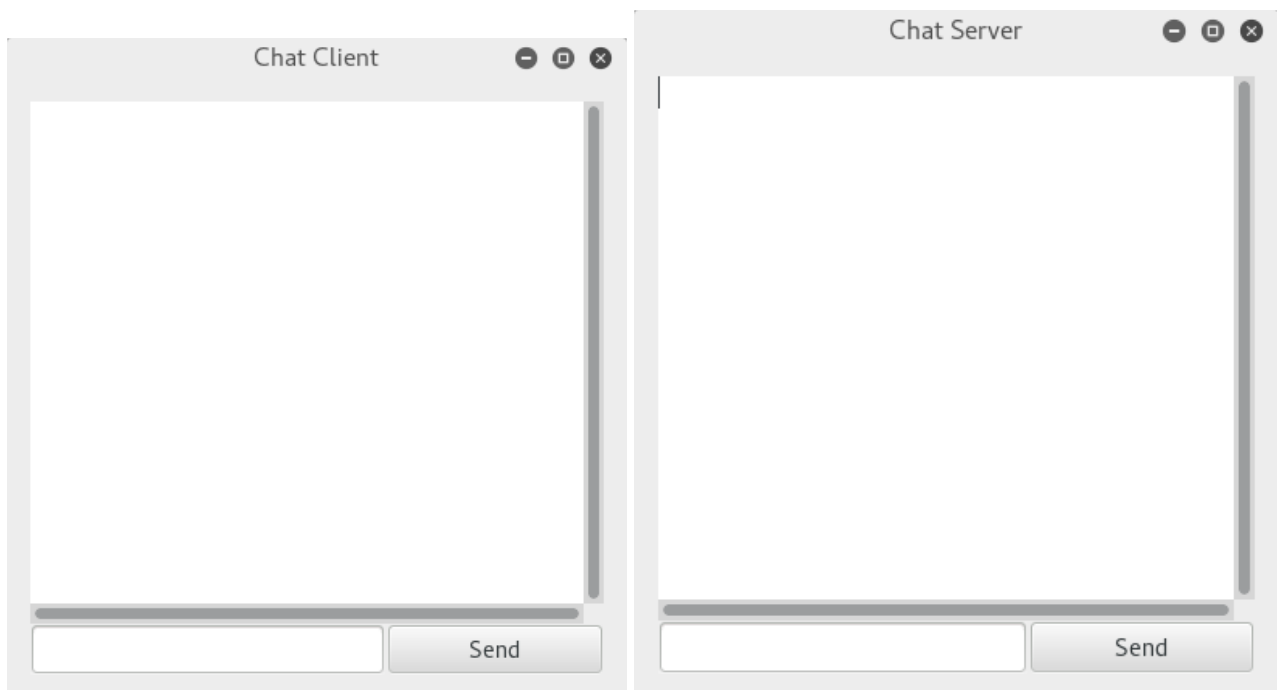
- The client can now communicate with this server using its **IP** and **PORT** with the command
- The client process was initiated using the command **\$/client**
- This opens up a GUI interface (created using GTK2+) wherein the client has to supply the IP and PORT information



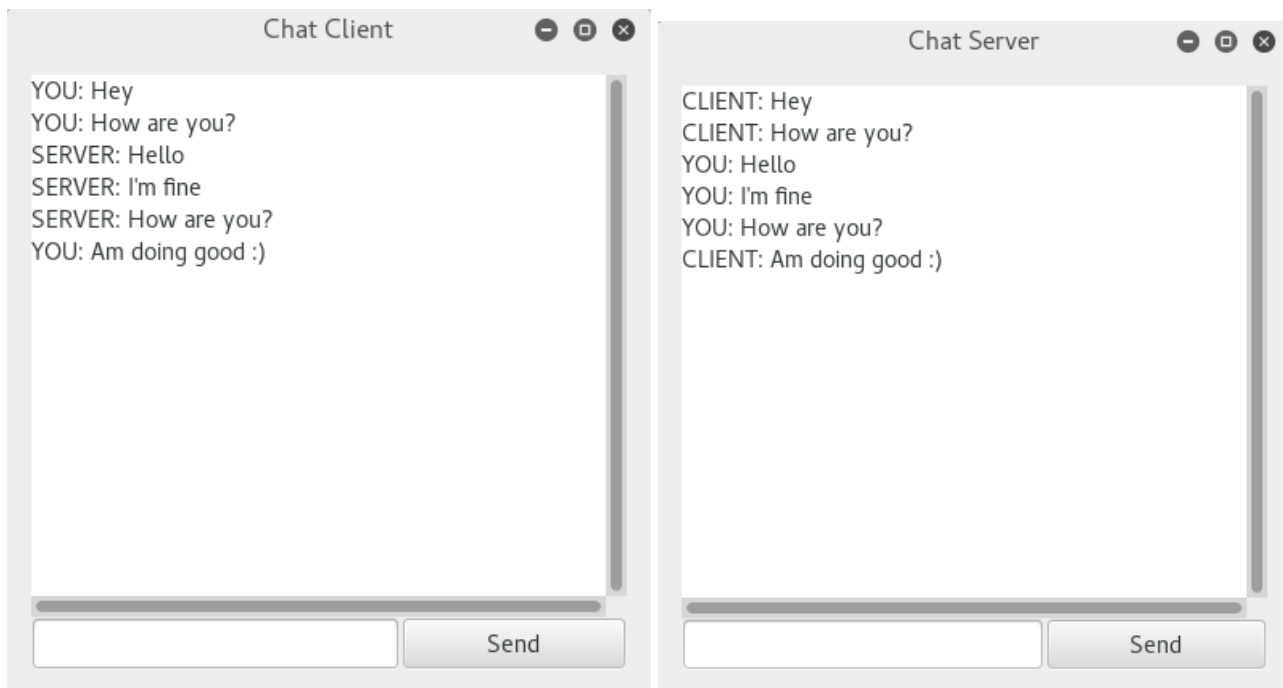
- After this, the IP and PORT details were entered in the client GUI



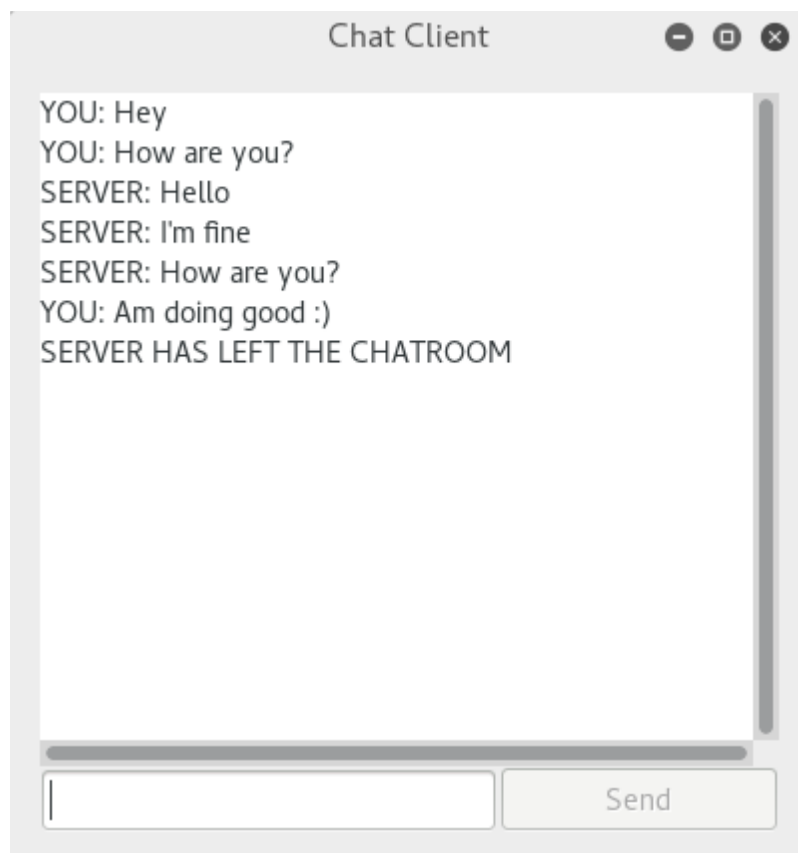
- **Connect** button was clicked to establish a connection with the server at given address.
- As soon as the connection gets established both the client and server are welcomed with a chat GUI wherein they can begin to chat with each other.



- Here is the screenshot of a random chat between the client and server.



- Now when the Server Quits the client is sent a message of abortion and the client app informs the user that **`SERVER HAS LEFT THE CHATROOM`**



Drawbacks:

- A server can chat with only one client at a time. Well we can create it to communicate with multiple clients in iteration, so that's not a drawback.

Problems you faced:

- Initially we made a simple command line interface for communication (chat) between server and client. It seemed too boring to our group & we had time, therefore we decided to go with a GUI model/prototype for the same.
- The problem of sending and receiving the data to and from both the client and server was an headache because without threading only one task can take place at a time. The solution was to go with threading.
- Initially we tried creating 2 sub processes, one to handle send and one to handle receive function for sockets. The problem left was of closing the socket which wasn't possible as unless every sub-process (forked) closes the socket the socket remained open. This was resolved with threading as there was no such issue of closing multiple socket copies in threading.
- In GUI, to scroll the chats as new chat arrives was a problem too which was later resolved by using a mark (the cursor) technique to scroll the text automatically as new text is received.

Discussion and Conclusion:

We had fun developing a chat application in GUI using SOCKETS. We got to know about a lot of stuff while doing that.

- When multiple messages are sent then they are kept in a message queue at the recipient side unless the recipient tries to fetch the message(s) from the queue.
- We need to specify the maximum length of the message that would be passed.
- Threading comes to rescue when dealing with parallel computations.
- In order to communicate between two processes a communication

channel needs to be set up and that will send data in bytes. Which can be decoded in C using proper struct assigned at receiver side to decode the messages into readable data by user.

Overall it was fun to develop such a chat application in C.