LINUX CHAT

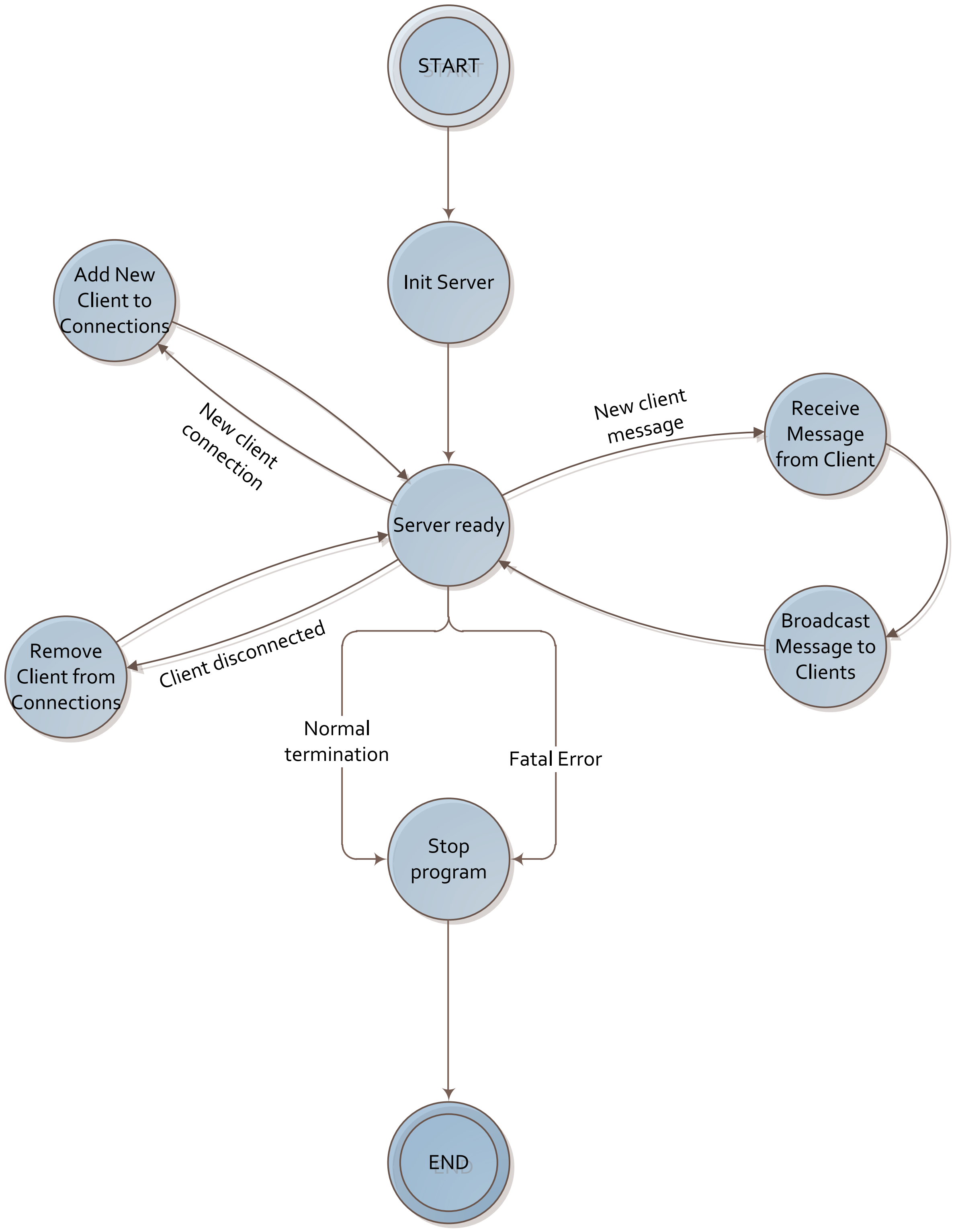
Local Area Communication

The Simple chat program that allows you to communicate with your peers in a free and friendly environment.

Contents

Linux Chat Systems Server

Server Visual Design



Server Pseudocode Design

Note: Each message being sent is appended with NULL and EOT at the very end of the message. This allows to have packets of variable size over TCP by detected what the last character set was. To counter-act this freedom, messages over the maximum buffer size of 2048 are thrown out.

Initialize Server

- Grab the user specified port if available, otherwise default to port 9654.
- Create the TCP listening socket and allow other sockets to connect to this port.
- Bind the socket and listen to all incoming connections.
- Any errors in any of the methods above will terminate the program.

Server Ready

- Initialize the file descriptors and add in the listening socket to the list of file descriptors.
- While the server is running
    - Listen on all connected sockets (initially only the listen socket is available)
    - If there is any socket activity, determine what type of activity has occurred:
        - New client: ADD NEW CLIENT TO CONNECTIONS
        - New connected client message: RECEIVE MESSAGE FROM CLIENT
        - Client disconnection: REMOVE CLIENT FROM CONNECTIONS
    - Restart this loop when the activity has been handled
- Since the server has stopped because of fatal error or user choice, STOP PROGRAM.

Add New Client to Connections

- Accept the new connection on a new socket to allow communicate but **do not add them to the list of connections yet.**
- Request the client to set their username and receive their desired username.
    - Any errors in this step will refuse the connection to this user and close the newly opened socket.
- Add the user to the list of connections.
- Return to SERVER READY loop

Receive Message from Client

- While the message has not been fully received
    - Continue receiving until EOT has been detected.
    - If the max buffer size has been reached, discard all received information. We are assuming that the packet is corrupted and return to SERVER READY.
- On valid message, append the username to the message and BROADCAST MESSAGE TO CLIENTS.
- Return to SERVER READY loop.

Broadcast Message to Clients

- Prepend the message with the client's username.
- Packetize the message with a NULL and EOT.
- Send the message to each client, skipping the original sender.
  - Ignore any send errors because maybe the receiver has disconnected during this time. We will handle that later.
- After the broadcast, return an ACK message to the original sender indicating that the message was sent to all possible clients.
- Return to RECEIVE MESSAGE FROM CLIENT.

Remove Client from Connections

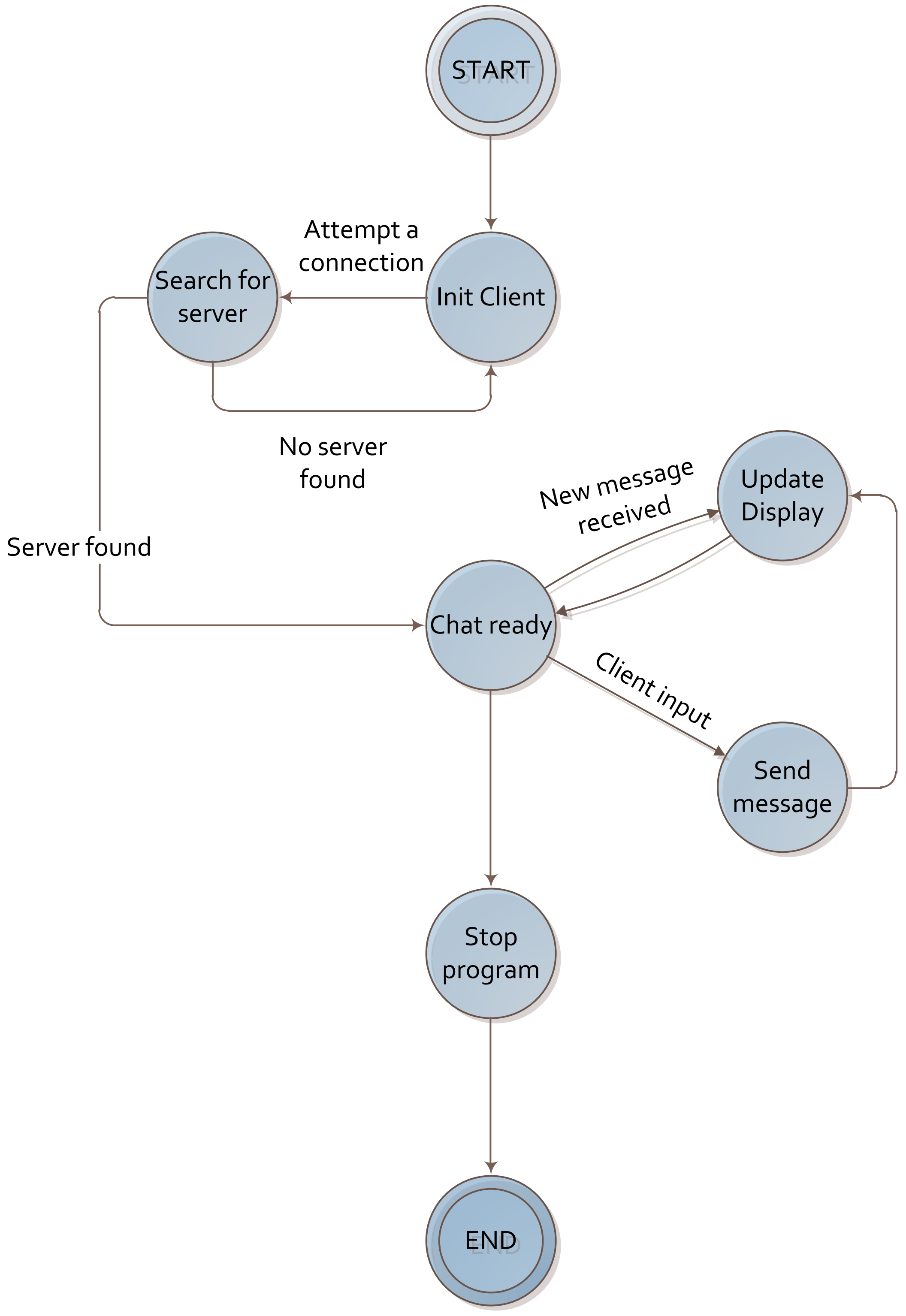
- Close the client and remove them from the list of connections.
- Return to the SERVER READY loop.

Stop Program

- Close all client connections and deallocate all used resources.
- Terminate the program.

Linux Chat Systems Client Design

Client Visual Design



Client Pseudocode Design

Note: Each message being sent is appended with NULL and EOT at the very end of the message. This allows to have packets of variable size over TCP by detected what the last character set was. To counter-act this freedom, messages over the maximum buffer size of 2048 are thrown out.

Initialize Client

- Acquire the port and server address from the user (default the port to 9654 if no port specified).
- Create a TCP Socket to be used and allow other sockets to bind that port.

Search for Server

- Find the server and connect to the server if available.
- If the server cannot be found
    - close the application.
- Once a connection is accepted, let the user set their username.
- Chat is now ready

Chat Ready

- Create a thread that will receive all incoming data from the server.
- While the chat is active
    - If the thread received a new message, UPDATE DISPLAY
    - If the user inputs a new chat message, SEND MESSAGE

Update Display

- Get the new message and remove the EOT from the end of the message.
- Add in the message to the display

Send Message

- Packetize the message by appending a NULL character and an EOT character at the very end of the message.
- Send the message to the server.

Stop Program

- Close the socket being used and free all allocated resources.
- Terminate the program.

Tyler Trepanier Vivek Kalia

Technical Report

Summary

Our chat program works very well handling multiple client connections, communication and disconnections. Having the server forcefully exit results in undesired behavior in the clients where they have to send a message to discover that the server does not exist anymore.

General Test Evaluations:

Our tests relied on having multiple machines connected to a server that was running. These machines were connected on a local area network and were used multiple times to ensure the validity of the tests. Our greatest concerns for testing were the edge cases.

Iteration 1: Client Connects to Server

Test Environment:

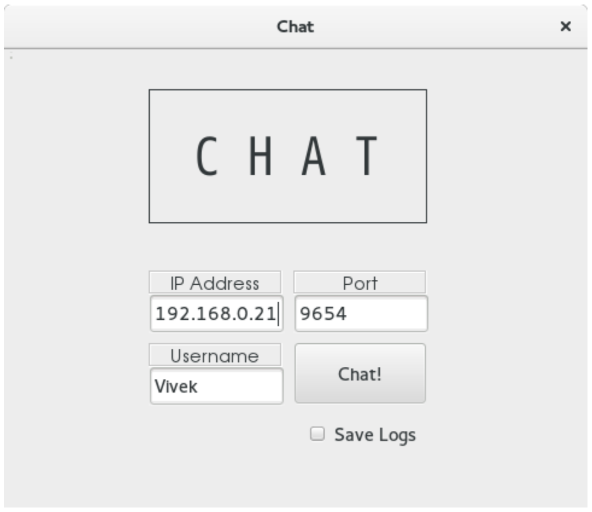
On a Fedora machine, we are running a local server while having another computer acting as a client. We will enter in our username, the server's IP Address and our desired username.

Test Purpose:

We will test the basic functionality of a Client connected to a running server.

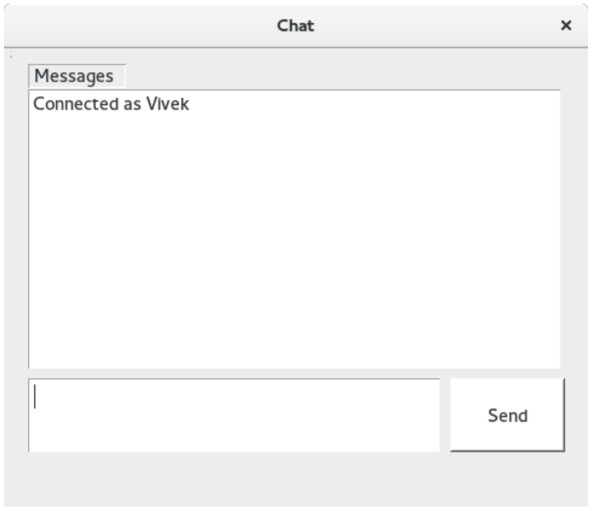
Test Results:

Test 1 – Fig 1



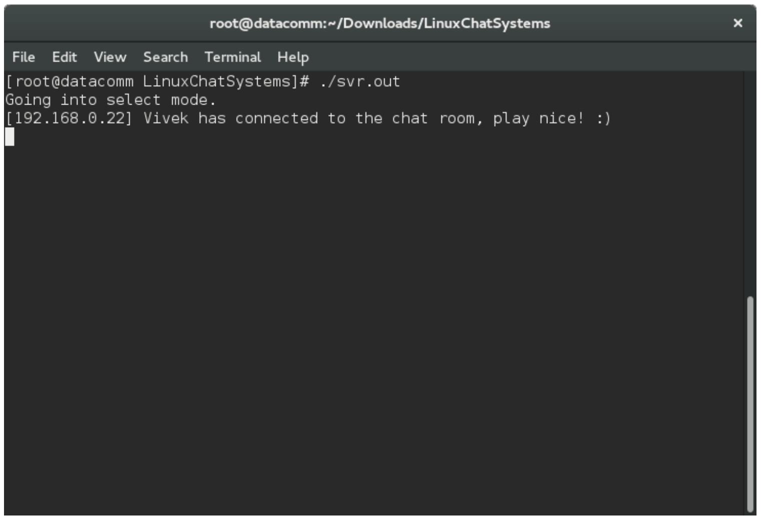
IP Address: 192.168.0.21
Port: 9654
Username: Vivek

Test 1 – Fig 2



Connected as Vivek

Test 1 – Fig 3


root@datacomm:~/Downloads/LinuxChatSystems
File Edit View Search Terminal Help

```
[root@datacomm LinuxChatSystems]# ./svr.out
Going into select mode.
[192.168.0.22] Vivek has connected to the chat room, play nice! :)
```

Test Conclusion:

Connecting to an active server is functional. The first “chat” message seen on the client window after connecting will be the window announcing that you are connected as “your username.” In addition, our server announces that the client has connected successfully, displaying the fact that they connected.
This is success.

Iteration 2: Client Sends a Message

Test Environment:

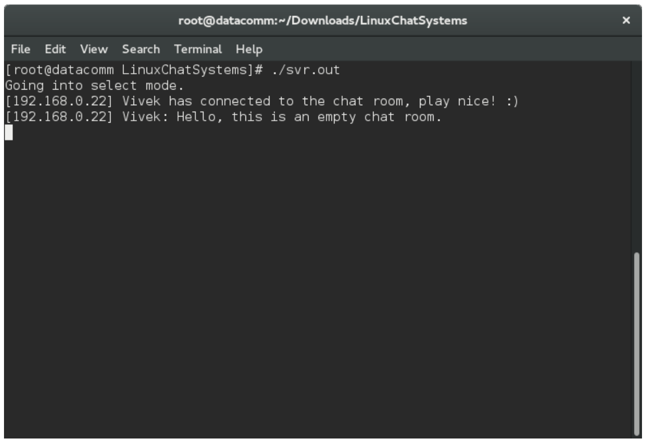
On a Fedora machine, we are running a local server while having another computer acting as a client. After a successful connection, we are going to send a message to the server.

Test Purpose:

We will test the basic functionality of a Client connected to a running server and sending a message to the server.

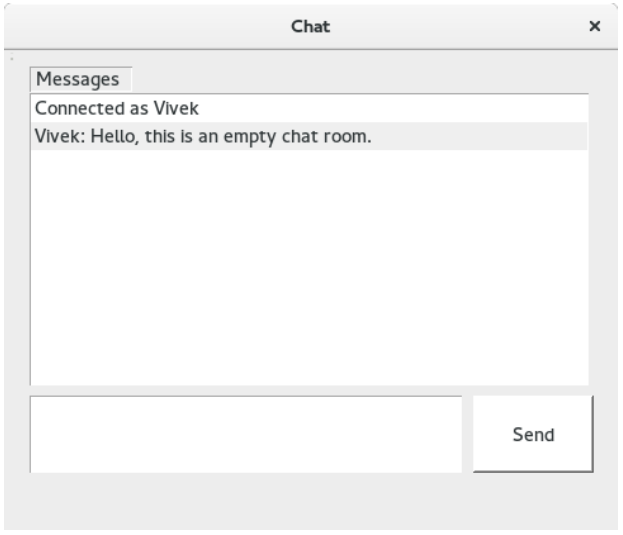
Test Results:

Test 2 – Fig 4


root@datacomm:~/Downloads/LinuxChatSystems
File Edit View Search Terminal Help

```
[root@datacomm LinuxChatSystems]# ./svr.out
Going into select mode.
[192.168.0.22] Vivek has connected to the chat room, play nice! :)
[192.168.0.22] Vivek: Hello, this is an empty chat room.
```

Test 2 – Fig 5



Connected as Vivek
Vivek: Hello, this is an empty chat room.

Test Conclusion:

Sending a message in the chat room functions properly when there is no other client available. The server will display all messages that have been sent to all the “connected” clients. In this circumstance however, the message about Vivek connecting to the chat room is not displayed to “Vivek.” This is intentional, and as such, **this test is a success.**

Iteration 3: Client Receives a New Message

Test Environment:

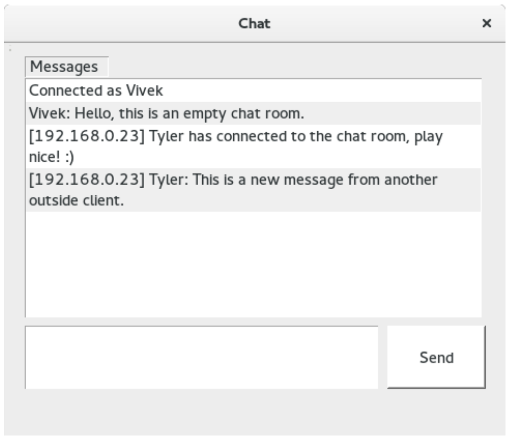
On a Fedora machine, we are running a local server while having two computers acting as a client. After a successful connection with both clients, we will remain connected until the other client sends a message.

Test Purpose:

We will test the basic functionality of a Client connected to a running server and how an active chat operates.

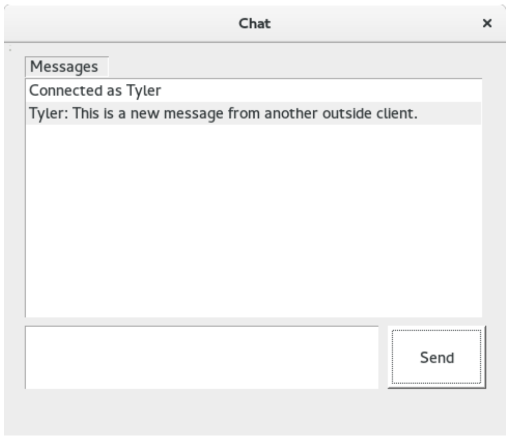
Test Results:

Test 3 – Fig 6



```
Connected as Vivek
Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play
nice! :)
[192.168.0.23] Tyler: This is a new message from another
outside client.
```

Test 3 – Fig 7



```
Connected as Tyler
Tyler: This is a new message from another outside client.
```

Test Conclusion:

When we are connected as Vivek, we can see that the new user "Tyler" has connected and Tyler's message which includes his IP Address. This is a success on Vivek's end. Tyler sent Vivek a message and we notice that the message that the sender sends, does not show the sender's IP Address.

This is intended because we want to have other users to display their IP Address, not the sender. This is a success.

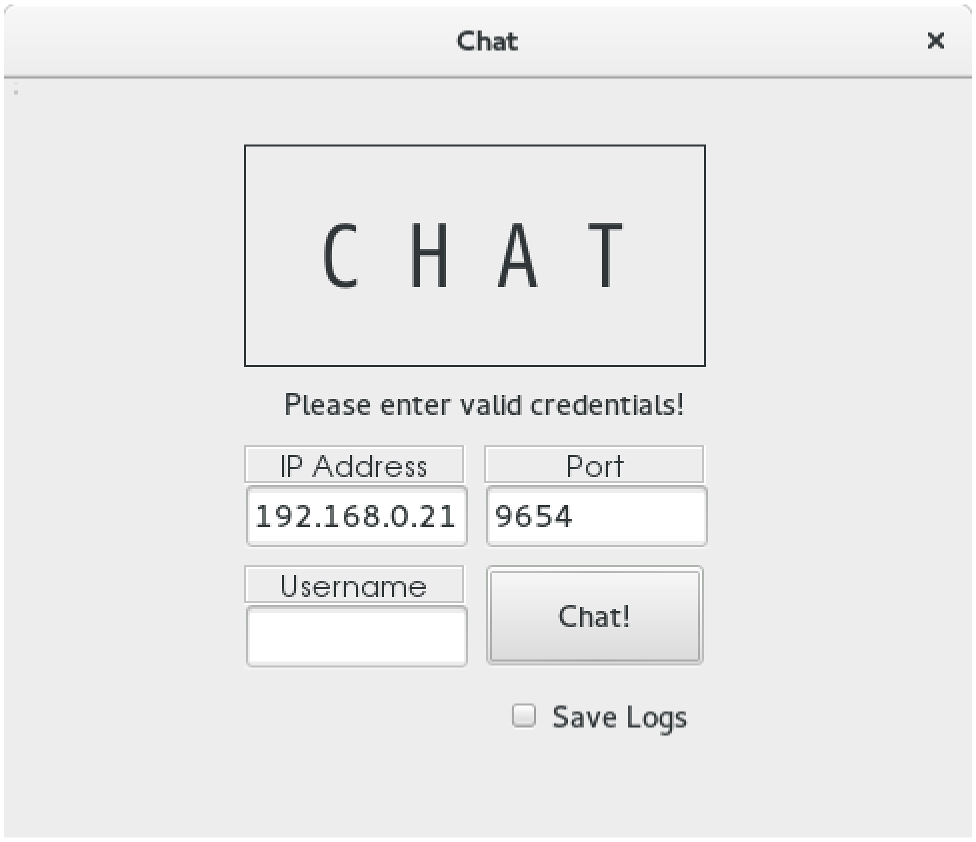
Iteration 4: Client Has a Bad Username

Test Environment:
On a Fedora machine, we are running a local server while having two computers acting as a client. We will attempt to connect to a client with no username.

Test Purpose:
We are testing to see if, essentially, a client successfully connects and specifies no username. A connection will occur initially but will be refused when there is no name.

Test Results:

Test 4– Fig 8



Please enter valid credentials!

IP Address: 192.168.0.21

Port: 9654

Username:

Test Conclusion:
The client refuses to connect to the server and displays an error message indicating that user has inputted invalid credentials. **This is an intended failure and will be declared as a success!**

Iteration 5: Client Disconnects

Test Environment:

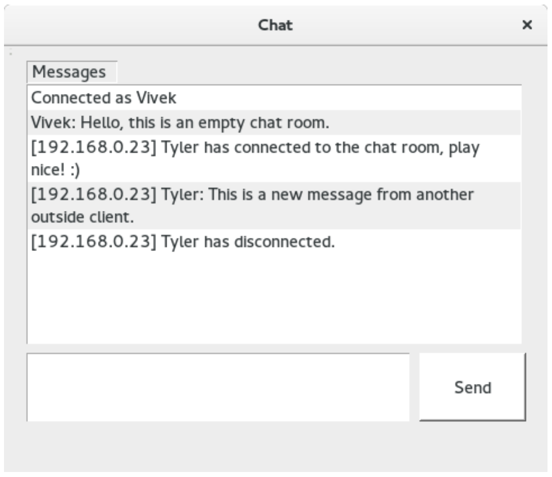
On a Fedora machine, we are running a local server while having two computers acting as a client. After a successful connection, we will disconnect after a chat session.

Test Purpose:

We are checking to see if a server properly handles a client's disconnection and sends a message to all the clients when those connections end.

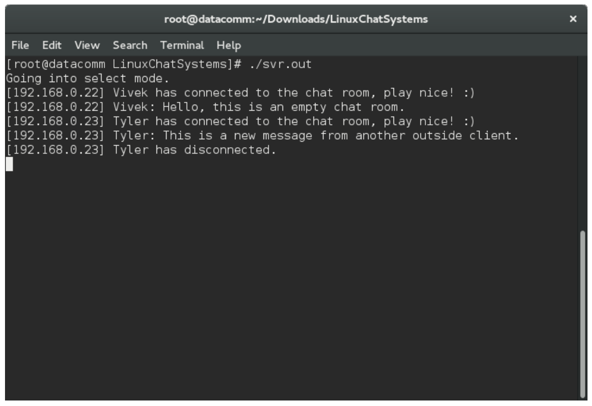
Test Results:

Test 5 – Fig 9



```
Connected as Vivek
Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play nice! :)
[192.168.0.23] Tyler: This is a new message from another outside client.
[192.168.0.23] Tyler has disconnected.
```

Test 5 – Fig 10


root@datacomm:~/Downloads/LinuxChatSystems
File Edit View Search Terminal Help

```
[root@datacomm LinuxChatSystems]# ./svr.out
Going into select mode.
[192.168.0.22] Vivek has connected to the chat room, play nice! :)
[192.168.0.22] Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play nice! :)
[192.168.0.23] Tyler: This is a new message from another outside client.
[192.168.0.23] Tyler has disconnected.
```

Test Conclusion:

The server detects a user disconnection and sends a message to the other client "Vivek" that Tyler has disconnected. We can see in the server window that we do receive an indication that Tyler did indeed disconnect.

This test is a success.

Iteration 6: Client Reconnects

Test Environment:

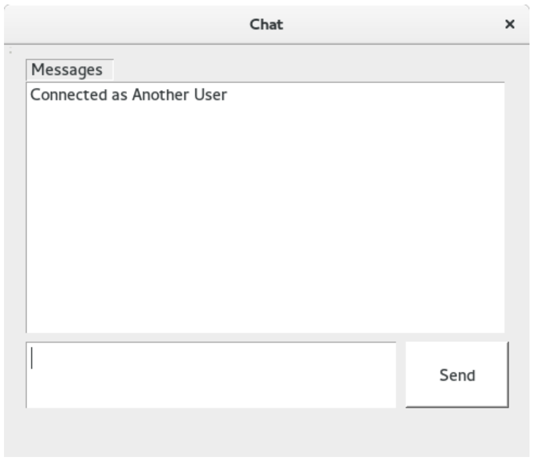
On a Fedora machine, we are running a local server while having two computers acting as a client. After previously being disconnected, we will re-enter the chat as a new user with the same IP Address as before.

Test Purpose:

We are checking to see if a server properly destroys a disconnected client's information and then recreates the client with a new username and chat instance.

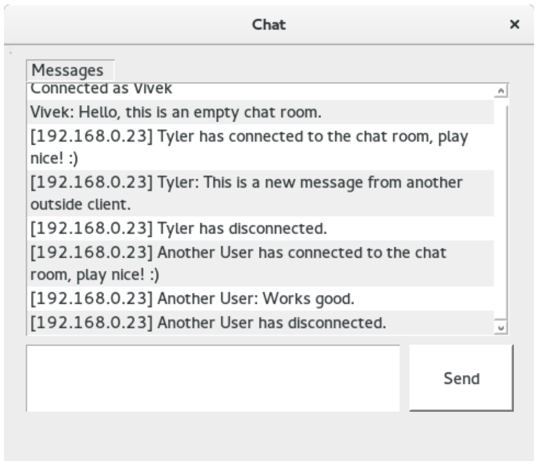
Test Results:

Test 6 – Fig 11



```
Connected as Another User
```

Test 6 – Fig 12



```
Connected as Vivek
Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play nice! :)
[192.168.0.23] Tyler: This is a new message from another outside client.
[192.168.0.23] Tyler has disconnected.
[192.168.0.23] Another User has connected to the chat room, play nice! :)
[192.168.0.23] Another User: Works good.
[192.168.0.23] Another User has disconnected.
```

Test Conclusion:

Iteration 7: Client Logs the Chat Session

Test Environment:

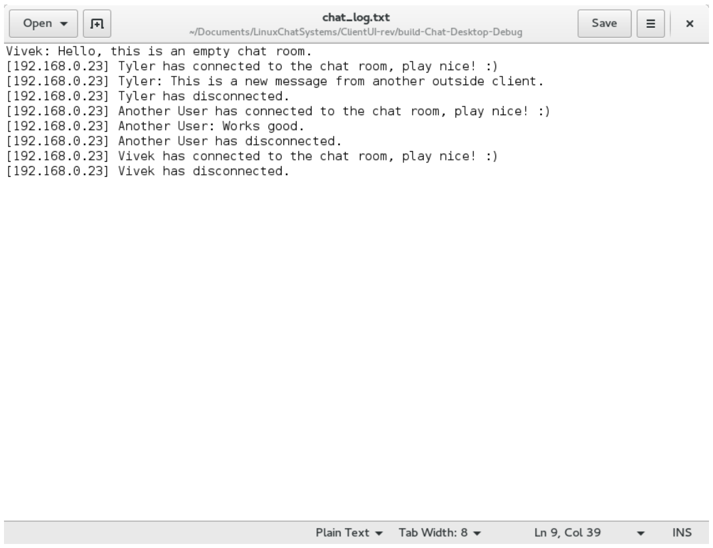
On a Fedora machine, we are running a local server while having two computers acting as a client. Right before the client connects, the client specifies that they want to log this chat session. Afterwards a normal chat session will occur.

Test Purpose:

We want to see the result of how the chat log appends every new chat message until a client session ends. Any non-ascii characters will be regarded as errors.

Test Results:

Test 7 – Fig 13


Open
chat_log.txt
~/Documents/LinuxChatSystems/ClientUI-rev/build-Chat-Desktop-Debug
Save
Plain Text
Tab Width: 8
Ln 9, Col 39
INS

```
Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play nice! :)
[192.168.0.23] Tyler: This is a new message from another outside client.
[192.168.0.23] Tyler has disconnected.
[192.168.0.23] Another User has connected to the chat room, play nice! :)
[192.168.0.23] Another User: Works good.
[192.168.0.23] Another User has disconnected.
[192.168.0.23] Vivek has connected to the chat room, play nice! :)
[192.168.0.23] Vivek has disconnected.
```

Test Conclusion:

When the user initially selects to log a chat session, they are capable of having their entire chat session logged into a file until they are able to delete it. The “chat_log.txt” will be where the program was executed.

This is a success.

Iteration 8: Client Has Bad Credentials

Test Environment:

On a Fedora machine, we are running a local server while having two computers acting as a client. The client will attempt to connect to a server with:

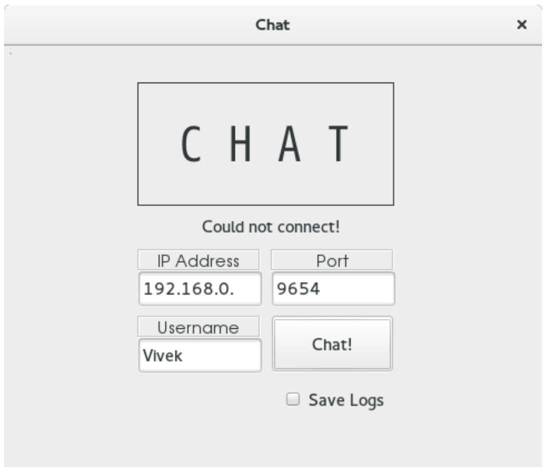
1. An invalid IP Address : IP Address = 192.168.0.
2. The wrong port number : Port = *1000*

Test Purpose:

A client that attempts to connect with the wrong credentials should immediately fail and return to the login page.

Test Results:

Test 8 – Fig 14



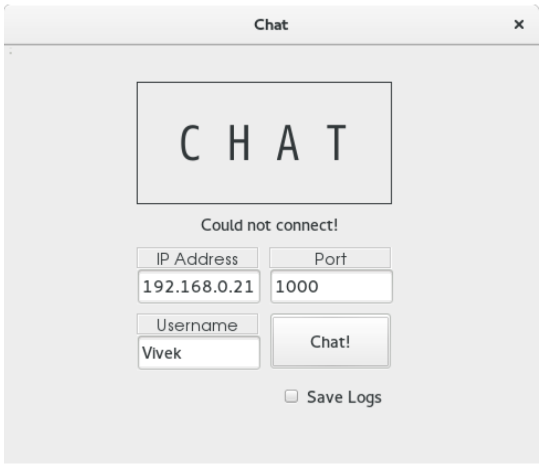
Chat

CHAT

Could not connect!

IP Address: 192.168.0.
Port: 9654
Username: Vivek

Test 9 – Fig 15



Chat

CHAT

Could not connect!

IP Address: 192.168.0.21
Port: 1000
Username: Vivek

Test Conclusion:

As we can see from both of the screenshots, both attempts resulted in a failure to connect. Since we can't connect, we are unable to proceed to the chat message screen.

This is working as intended and is a success.

Iteration 9: Creation of a New Server Instance

Test Environment:

On a Fedora machine, we will start the server with no clients.

Test Purpose:

Checking to see if a server can initialize.

Test Results:

Test 9 – Fig 16

Test Conclusion:

Creating a new instance of the Server causes no errors and announces that it is going into select mode.
This is a success.

Iteration 10: Normal Server Operation

Test Environment:

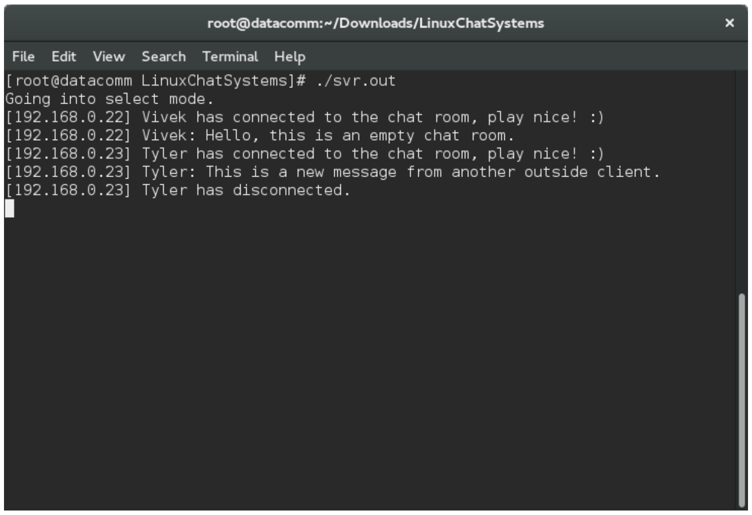
On a Fedora machine, we will see the normal operation of a server when multiple clients connect, disconnect, reconnect and interact with each other.

Test Purpose:

We want to see normal client interaction and how the system handles it. Anytime the server crashes will result in a failure.

Test Results:

Test 10 – Fig 17


root@datacomm:~/Downloads/LinuxChatSystems
File Edit View Search Terminal Help

```
[root@datacomm LinuxChatSystems]# ./svr.out
Going into select mode.
[192.168.0.22] Vivek has connected to the chat room, play nice! :)
[192.168.0.22] Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play nice! :)
[192.168.0.23] Tyler: This is a new message from another outside client.
[192.168.0.23] Tyler has disconnected.
```

Test Conclusion:

We can see from normal server operation that there are no errors to be seen from users connecting, disconnecting and interacting with each other.

This is a success.

Iteration 11: Server Stops Running

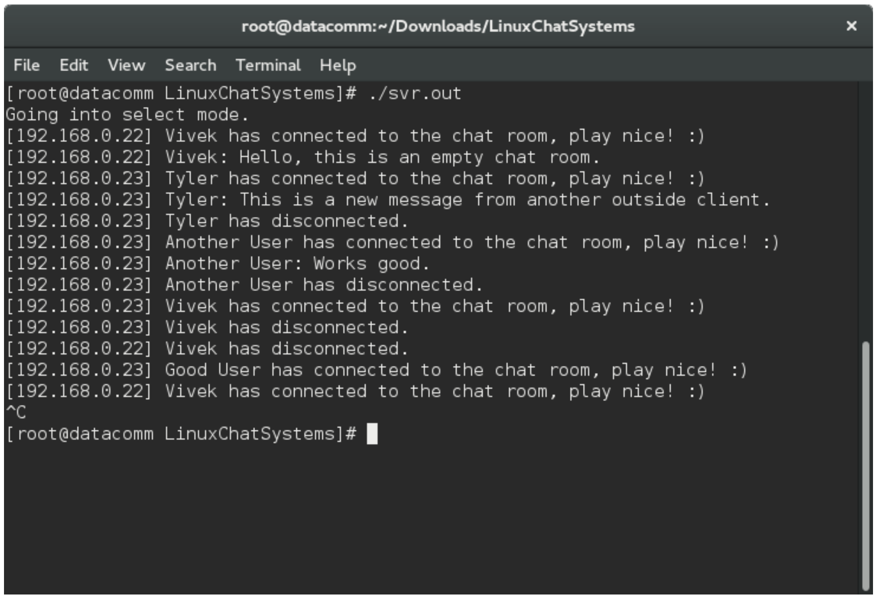
Test Environment:

On a Fedora machine, we are running a local server while having two computers acting as a client. After normal user interaction, the server will suddenly stop. This will destroy all current client connections. We first made

Test Purpose:

We want to see what happens on the client end when we can't send and receive messages to a server. A client instance should terminate when there are too many failures.

Test Results:


root@datacomm:~/Downloads/LinuxChatSystems
File Edit View Search Terminal Help

```
[root@datacomm LinuxChatSystems]# ./svr.out
Going into select mode.
[192.168.0.22] Vivek has connected to the chat room, play nice! :)
[192.168.0.22] Vivek: Hello, this is an empty chat room.
[192.168.0.23] Tyler has connected to the chat room, play nice! :)
[192.168.0.23] Tyler: This is a new message from another outside client.
[192.168.0.23] Tyler has disconnected.
[192.168.0.23] Another User has connected to the chat room, play nice! :)
[192.168.0.23] Another User: Works good.
[192.168.0.23] Another User has disconnected.
[192.168.0.23] Vivek has connected to the chat room, play nice! :)
[192.168.0.23] Vivek has disconnected.
[192.168.0.22] Vivek has disconnected.
[192.168.0.23] Good User has connected to the chat room, play nice! :)
[192.168.0.22] Vivek has connected to the chat room, play nice! :)
^C
[root@datacomm LinuxChatSystems]#
```

Test Conclusion:

At first the user attempted to send the message which did not give a socket error and the user's chat room message screen updated without issue. Another attempt to send caused the application to quit because the server failed didn't receive the acknowledge from the server.

The result is a desired error and forced application quit. This is intentional, success.

Conclusion of the resulting tests:

This chat program handles connections, user interactions and disconnections from clients without issue. Server disconnections will eventually force the client applications to close because that's how they were designed.