# 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.