Multithreaded Python Chat Server

With GUI

Michael Ivain and Abhilash Bhabad

Description

In this project, we used sockets and threading to create a chat server in Python. We also used tkinter, the Python binding to the Tk GUI toolkit, to create a GUI for the chat server application.

Our code consists of two files, server.c and client.c, providing the server side functionality and client side functionality.

Our chat server allows for up to 5 client connections, who are all able to chat with each other in a group chat environment.

After walking through the source code, we will demonstrate the code on the local host.

Server Side

Receive Connection Function

```
"""Server for multithreaded chat application"""
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread


def accept_connections():
    """Accepts connection from clients"""
    while True:
        user, user_adr = SERVER.accept()
        print("%s:%s has connected." % user_adr)
        user.send(bytes("Type your username and press enter to begin", "utf8"))
        addresses[user] = user_adr
        Thread(target=handle_user, args=(user,)).start()
```

Handle Connection Function

```
def handle_user(user):  # Takes user socket as argument.
    """Handles user connection"""

    username = user.recv(BUFFER_SIZE).decode("utf8")
    welcome = 'Welcome %s! If you ever want to quit, type {quit} to exit.' % username
    user.send(bytes(welcome, "utf8"))
    message = "%s has joined the chat!" % username
    broadcast(bytes(message, "utf8"))
    users[user] = username

    while True:
        message = user.recv(BUFFER_SIZE)
        if message != bytes("{quit}", "utf8"):
            broadcast(message, username+": ")
        else:
            user.send(bytes("{quit}", "utf8"))
            user.close()
            print("%s:%s has disconnected." % addresses[user])
            del users[user]
            broadcast(bytes("%s has left the chat." % username, "utf8"))
            break
```

Broadcast Message Function

```python
def broadcast(message, prefix=""):  # prefix is for username identification.
    """Broadcasts a message to all users"""

    for sock in users:
        sock.send(bytes(prefix, "utf8")+message)
```

When a user types a message into their client program, the server will receive the message and use this function to broadcast to all currently connected clients

Main Function

```
users = {}
addresses = {}

HOST = ''
PORT = 33000
BUFFER_SIZE = 1024
ADDR = (HOST, PORT)

SERVER = socket(AF_INET, SOCK_STREAM)
SERVER.bind(ADDR)

if __name__ == "__main__":
    SERVER.listen(5)
    print("Waiting for connection...")
    ACCEPT_THREAD = Thread(target=accept_connections)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()
```

The main function manually sets the host and port for clients to connect.

Then a socket is created and binded to the address.

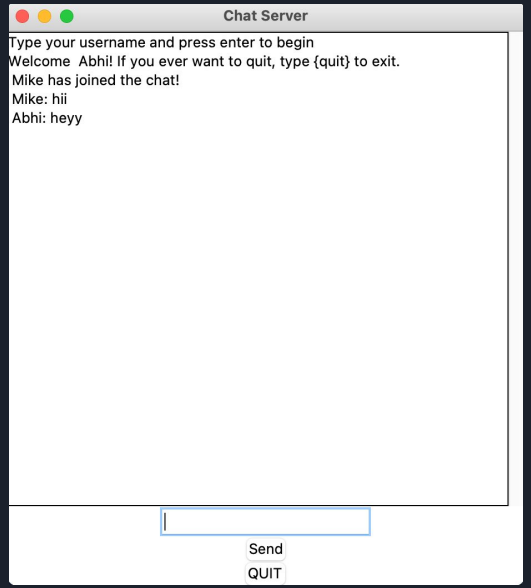
When a client connects, the server creates a thread and starts it.

When the client quits, the server will join the thread to close.

Client Side

GUI implementation

```
top = tkinter.Tk()
top.title("Chat Server")

messages = tkinter.Frame(top)
msg_value = tkinter.StringVar()  # For the messages to be sent
msg_value.set(" ")
scrollbar = tkinter.Scrollbar(messages)  # To navigate through past messages


all_msg = tkinter.Listbox(messages, height=25, width=50, yscrollcommand=scrollbar.set) # List of all prior messages
scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
all_msg.pack(side=tkinter.LEFT, fill=tkinter.BOTH)
all_msg.pack()
messages.pack()

message_field = tkinter.Entry(top, textvariable=msg_value)
message_field.bind("<Return>", send)
message_field.pack()


send_button = tkinter.Button(top, text="Send", command=send)
send_button.pack()

quit_button = tkinter.Button(top, text="QUIT", command=quitting)
quit_button.pack()

top.protocol("WM_DELETE_WINDOW", quitting)
```



This code uses tkinter to create a GUI interface, utilizing functions created (recieve, send, quitting) to create a view of all the past messages, an entry field, a send button and a quit button.

Receive message function

```
# Recieve Messages
def receive():
    while True:
        try:
            val = server_connection.recv(BUFSIZ).decode("utf8") # Recieve message from server
            all_msg.insert(tkinter.END, val) # Insert new message into message list all_msg
        except OSError:
            break
```

This function runs continuously to receive messages from the server side.

Once a message is received, it inserts the message into the message list all_msg to be displayed in the GUI

Send Message function (send button in GUI)

```
# Send Messages
def send(event=None):
    val = msg_value.get() # Get value from entry field
    msg_value.set("") # Reset it to nothing
    server_connection.send(bytes(val, "utf8")) # Send the message to the server
    if val == "{quit}": # If the message is 'quit' close the connection with the server
        server_connection.close()
        top.quit()
```

This function sends messages for the client. It gets the message from the msg_value field from the GUI after the send button is hit. It then sets the field back to nothing. Then, it sends the message to the server through the socket so the server can broadcast it to all the clients. If the msg value is ‘quit’, it closes the socket and quits the GUI.

Quitting function (quit button in GUI)

```
#When you quit
def quitting(event=None): # Uses the send function with the message value
    msg_value.set("{quit}") # Sets the msg_value to 'quit'
    send() # Uses send function to bring it to if statement in send()
```

This function quits for the client and is called by the QUIT button on the GUI. Essentially, the function uses the if statement in the send function to quit the client as shown in last slide.

Main code (Sockets)

```
HOST = input('Enter host: ')
PORT = input('Enter port: ')

if not HOST:
    HOST = '127.0.0.1'
if not PORT:
    PORT = 33000
else:
    PORT = int(PORT)

BUFSIZ = 1024
ADDR = (HOST, PORT)

server_connection = socket(AF_INET, SOCK_STREAM)
server_connection.connect(ADDR)

begin_recieving = Thread(target=receive)
begin_recieving.start()
tkinter.mainloop()  # Starts GUI execution.
```

This code initializes the host and port to connect to the server on in the client's terminal.

We implemented it so simply entering when prompted defaults HOST, PORT to the local host and port 33000

It then uses these values to connect to the server, and begins a thread called on the receiving function, which runs indefinitely to receive messages from other clients broadcasted by the server.

Demonstration