Codey Sivley

CS415 Operating Systems

Dr. Lewis

Spring 2022

Chatting the Evil Way

Intro

This assignment is to create a chat client utilizing ports and multithreading. In my solution, the files are named spcServer and spcClient, for Simple Python Chat. Of course, I found the assignment a little more challenging than "simple" would imply.

spcServer.py

This is the server-side script. Major functionality includes:

- Calling the script
    - For simplicity, I let the IP and Port number be args here.
- Setup
    - I don't understand a lot of networking stuff, so I followed an online tutorial pretty heavily here.
    - Since I was operating from the home IP, I used port numbers wherever I could to represent different users.
- Interpreter
    - Upon receiving a message, the message is split for parsing
    - A simple if/elif loop looks for the following leading keywords:
        - USERS: prints a list of users directly to the requesting port client
        - DISCONNECT: kills the connection, but isn't clean on the client side
        - MESSAGE: passes the message to the broadcast function, which sends it to every connected port that isn't the sender port.
            - The MESSAGE keyword is omitted from the send.

Missing functionality:

- I didn't like that the users were listed as port numbers. I wanted to add a "NICK" command that would pair port numbers to a string, then pass the string on message broadcasts, but I ran out of time.
- There is no direct logging functionality, but ideally all of the "print()" statements would just be routed to a file instead of / in addition to the console.
- Did not include whisper functionality, kick functionality, or pwn functionality.
    - Whisper/kick: I imagine that it would be done the same way: Detect the keyword, then trigger the related function but with a target other than the client port.
    - Pwn: I did not figure out how I'd access the system file structure from here.

spcClient.py

Client side script. Functionality includes:

- commandInterpreter()
    - Mirroring the server-side interpreter, this interpreter splits the command string into tokens.
        - CONNECT: looks to tokens 1 and 2 for IP and Port. This allows the client to connect to anywhere.
        - MESSAGE: Sends the message to the server, and echos to the stdout
        - USERS: Simply sends the string USERS to the server, which returns a list.
            - Server side interpreter protects against a MESSAGE beginning with "USERS" causing a false trigger by interpreting the raw string instead of the trimmed string.
        - DISCONNECT: Asks the server to boot it.
        - HELP: A little helper function that lists acceptable commands. Every command line needs one!
- Chat()
    - Loops indefinitely, looking for either incoming data which it prints, or command line data that it interprets.

Missing functionality:

- I don't like the for/if/else loop to check for input and output. I would rather make two threads in a reader/writer pattern, then mutex the print statement as needed for each to echo to cmd. I couldn't figure it out though.

Pitfalls

- Windows I/O
    - I struggled a lot with Python blocking the stdin/stdout pipes.
    - I eventually moved the project to a linux box where I knew Windows shenanigans weren't getting in the way.
- Message passing
    - Python made it pretty easy to parse out a command, but that didn't keep it from giving me a hard time. When adding interpreter functionality to the server-side, I got stuck where my try/except flow was breaking off before my broadcast could trigger, and was doing so silently.
- Universal Ports
    - I got stuck for a while because I was running the code in both cmd.exe and a python shell. However, each of these will connect to the same port with no complaints. Therefore, my messages were being intercepted by an old run of the server! I wasn't able to figure it out until I went back and saw that I had a multitude of connections on that run.

Source Code:

spcServer.py

```
# Codey Sivley

# For CS415 Operating Systems

# Dr Lewis

import socket

import select

import sys

import string


from _thread import *


"""The first argument AF_INET is the address domain of the

socket. This is used when we have an Internet Domain with

any two hosts The second argument is the type of socket.

SOCK_STREAM means that data or characters are read in

a continuous flow."""

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)


if len(sys.argv) != 3:

	print ("Correct usage: script, IP address, port number")

	exit()


IP_address = str(sys.argv[1])

Port = int(sys.argv[2])

server.bind((IP_address, Port))

server.listen(100) #max connections
```

```
list_of_clients = [] #list of currently connected client objects


def clientthread(conn, addr):


        #successful connection greeting.
        conn.send(("You have entered the EVIL chatroom!").encode())


  #each client thread runs ad infinitum
        while True:
                try:
                        message = conn.recv(2048)
                except:
                        print("Message recieve error")
                        conn.close()
                        break #closes thread on server side, but client crashes
                else:
                        if message:
                                message = message.decode()
                                messageParse = message.split()
         #message is parsed here for user commands.
                                if messageParse[0] == "USERS":
           #send only to requesting user
                                        userlist = str(list_of_clients)
                                        conn.send("USERS ONLINE\n".encode())
                                        for each in list_of_clients:
                                                conn.send(str(each.getpeername()).encode())


                                elif messageParse[0] == "DISCONNECT":
                                        print(str(conn) + " is disconnecting.")
                                        conn.send("Goodbye.".encode())
                                        remove(conn)
                                        conn.close()
        break
                                elif messageParse[0] == "MESSAGE":
                                        outgoing = " ".join(messageParse[1:])
                                        print ("<" + str(addr[1]) + "> " + outgoing)
                                        # Calls broadcast function to send message to all
                                        message_to_send = (str("<" + str(addr[1]) + "> " + outgoing +
"\n"))
                                        broadcast(message_to_send.encode(), conn)
                                else:
                                        print ("message not understood")
                                        print (message) #for debug mostly

"""Using the below function, we broadcast the message to all
clients who's object is not the same as the one sending
the message """
def broadcast(message, connection):
        print ("Broadcasting message: " + str(message))
        for clients in list_of_clients:
                if clients!=connection:
                        try:
                                clients.send(message)
                        except:
                                clients.close()

                                # if the link is broken, we remove the client
                                remove(clients)
```

```
def remove(connection):
    if connection in list_of_clients:
        list_of_clients.remove(connection)


print("Ready for clients.")
while True:


    """Accepts a connection request and stores two parameters,
    conn which is a socket object for that user, and addr
    which contains the IP address of the client that just
    connected"""
    conn, addr = server.accept()

    """Maintains a list of clients for ease of broadcasting
    a message to all available people in the chatroom"""
    list_of_clients.append(conn)

    # prints the address of the user that just connected
    print (str(addr[1]) + " connected")

    # creates and individual thread for every user
    # that connects
    start_new_thread(clientthread,(conn,addr))

conn.close()
server.close()
```

spcClient.py

```
# Codey Sivley

# For CS415 Operating Systems

# Dr Lewis

import socket

import select

import sys


#global stuff for port connections

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


def commandInterpreter(cmd):

        #not sure how to do this other than an if tree?

        cmdParse = cmd.split()

        if cmdParse[0] == "CONNECT":

                try:

                        server.connect((str(cmdParse[1]), int(cmdParse[2])))

                except:

                        print("Connection failed. Please check address and try again.")


        elif cmdParse[0] == "MESSAGE":

                outgoing = " ".join(cmdParse[1:])

                server.send(cmd.encode()) #send the whole thing!

                sys.stdout.write("<You>")

                sys.stdout.write(outgoing + "\n")

                sys.stdout.flush()


        elif cmdParse[0] == "USERS":

                server.send("USERS".encode())
        elif cmdParse[0] == "DISCONNECT":
                server.send("DISCONNECT".encode()) #graceful disconnect

        elif cmdParse[0] == "HELP":
                print("Accepted commands:\n CONNECT [ip-address][port]\n MESSAGE [message]\n
USERS\n DISCONNECT")

        else:
                print("Unrecognized command. Type \"HELP\" for help.")
##################### splash page ###################
print("Welcome to evilChat!")
commandInterpreter(sys.stdin.readline())

def chat():
        while True:
                # maintains a list of possible input streams
                sockets_list = [sys.stdin, server]

                read_sockets,write_socket, error_socket = select.select(sockets_list,[],[])

                for socks in read_sockets:
                        if socks == server:
                                message = socks.recv(2048)
                                print (message.decode())
                        else:
                                message = sys.stdin.readline()
                                commandInterpreter(message)
                                sys.stdout.flush()
        server.close()
chat()
```

Sources:

Heavily referenced the geeksforgeeks page on this:

https://www.geeksforgeeks.org/simple-chat-room-using-python/