Individual portfolio assignment

Johan H. Hansen

S349880

DATA2410

Table of contents

Assignment overview

To make a simple chat bot, all you need is a function that takes a string as input (what somebody said) and returns another string (the bot's response). A more complex bot would have a longer memory but responding to a single sentence is more than hard enough, so let's stick with that. The main part of this assignment is to make a little chat server with a couple of bad chat bots on it. To make it simpler, let's say that your bots (your functions) never have to take the initiative but only respond to what someone else is suggesting. In the simplest case, the suggestion is a single verb—a suggestion for something to do to pass the time **" What do you like doing in your spare time? "** or **"What day is it today? "** or **"Let's eat!"**. How your bots respond to these suggested actions is **completely up to you**, but you should have **FOUR** different ones. Remember, your code should be more elegant.

For this assignment, Bots are functions that take a string or two as input and return one or two strings:

3. Task 1: TCP Client

Implement a TCP client program, **"client.py"**, that takes 3 command line parameters: *ip, port,* and *bot.* The idea is that each client can connect from its own terminal and run a single bot each. Several terminal windows can run in parallel, connecting different bots to the same server. You can provide command line parameters if you like, but they must then be explained if you call the program with "--help" or "-h".

The client will do the following:

1. Create a TCP socket and connect to ***(ip, port)***
2. Read from the socket, line by line.
    a. If the line is from the host, you can expect it to be a suggestion, e.g. "Let's take a walk" or "Why don't we sing?". Extract the suggested action from the line. E.g. "walk" or "sing".
        i. Call the function *bot* to create a response. Send the response back over the socket.
        ii. You can choose to remember the suggested action as alternative 1.
    b. If the line is from one of the other participants, you can choose to ignore it,

OSLOMET

or pass it to your bot as alternative 2 if there's already a suggested action.

3. You are free to decide how and when to end the connection.
4. sending a message that might be dropped if the client is not ready to receive messages (optional)

4. Task 2: TCP Server

Implement a TCP server program, "**server.py**" that takes a single parameter *port*. The idea is that the server acts as a chat room and that it initiates rounds of dialogue to make the protocol (yes, you're making a protocol!) easier to implement. You can provide command lineparameters if you like, but they must then be explained if you call the program with "--help" or "-h". The server will do the following:

1. Accept any connection. You can expect all connections to be a bot, e.g. that they willnot be the first to speak, but that they will always respond. This gives you the option of waiting for them. You can also decide to make your program more robust by reading from clients in parallel, or if you're a real pro, use select or poll to makethe clients non-blocking. But keep in mind: it's better to have something simple that works than something sophisticated that doesn't.
2. Initiate a round of dialogue by suggesting an action. Send the suggestion to each ofyour connected clients. The action can be random, provided as user input for each.
3. All responses should be sent back out to all clients except the one who sent it.
4. Maintain a list of connected clients. If you want, you can let new connections wait untilyou've completed one round of dialogue. A good program will check if clients are still connected before trying to interact with them. If they're not, or if you decide that they're taking too long to respond, you can remove them from the list of connections.
5. You are free to decide when and how to disconnect the clients (you can even kickthem out if they misbehave) and how to gracefully terminate the program.
6. Make a "bot" that takes its response from the command line. That way you or other users can interact with the bots and make the dialogue more interesting.
7. Don't nag your users. It's sometimes nice to let the user add choices and options, butyour defaults should work well without user interaction. Don't make them fill out forms.

introduction

To my understanding, the assignment wants me to have a minimum of 2 files. I have decided to make a third file, response.py which is used to generate responses from the client.py-file. The first one, called server.py, will take one parameter as input, and that's the port it should run on. "server.py" should act as a sort of host, that sends out messages to all the connected clients and receives their responses.

The second file, client.py, acts as a participant in the chat. This file takes 3 parameters, IP, port and bot. When running the program, it should be possible to run multiple instances of client.py, all connected to the socket created in server.py. "client.py" should listen for suggestions from the host(server) and generate responses to these suggestions. Use of AI and tools such as NLTK are not allowed for this assignment, which indicates we are supposed to make it simplistic without too much focus on the responses themselves, but rather making it functional and able to handle errors etc. There are many creative ways to get the conversation flowing, which could be used to show of skills in data management. However, this doesn't seem to be the focus of the assignment.

Assumptions

Although the goal of this assignment is clear, I have noticed a few conflicting messages.
First off, in the PPT-slide published by Aws (Aws, 2022) [1], one of the slides says that the solution could be 100% single threaded. However, in server requirement 1 it's indicated that we can read from the clients in parallel, which to my knowledge means that we have 1 thread running for each client. Therefore, I assume that it's not a requirement for our solution to be single-threaded. While writing this I'm not quite sure what approach I

Additionally, Aws suggested on slack (Aws, 2022)[2] that we can let the bots respond to each other, for example by correcting each other's grammar. But in requirement 1 for the server, it says; "You can expect all connections to be a bot, e.g., that they will not be the first to speak, but that they will always respond. This gives you the option of waiting for them" This is somewhat conflicting, since the bots should only respond to suggestions from the host, and not provide unsolicited messages about the other bots.

Requirements walkthrough

Client Requirements

Requirement 1

"1. Create a TCP socket and connect to (ip, port)"

```
ip = sys.argv[1]
port = int(sys.argv[2])
botname = sys.argv[3]

socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.connect((ip, port))
socket.send(botname.encode())
```

The assignment specifies the fact that we should take 3 input-parameters in the client program. This is done with the help of the sys-module (Python, 2022)[3]. The last 3 lines creates a socket, connects it to the specified ip and port, and then sends its name for the server to handle. Every time I discover a new connection from the server side, a message is sent with the client's name.

Requirement 2

```
while True:
    x = socket.recv(1024).decode()
    # responsex29 signifies a response from another client.
    # Before printing the message, it cuts out the identifier 'responsex29'
    if "responsex29" in x:
        x = x.replace("responsex29", "")
        print(x)

    else:
        x = x.replace("?", "") # Removes the question mark while processing the suggestion and finding response
        sending = botname + ": " + findbot(botname, x) # Calls on findbot() to generate the response
        print("New sequence... ")
        print("\nHost:" + x)
        socket.send(sending.encode())
        sending = sending.replace(botname + ":", "") # Formats the response before printing it out
        print("Me:", sending)
```

The 'while True' runs constantly, and allows the client to listen for messages from the host. A similar loop is made for the bot Soria, which I explain further in server requirement 6.

2a

"If the line is from the host, you can expect it to be a suggestion".

In my program, 2 kinds of messages are to be expected. It's either a suggestion from the host, or a response from another client. To separate these two, I have chosen to give responses from other clients an appendix, "responsex29". If the message contains that code, we know it's a response from a different client. When it's a suggestion from the host, the function sendbot() is called. It finds the correct botname, and gives a response from that.

```
def findbot(name, msg):
    name = name.lower()
    if name == "chuck":
        return chuck(msg);
    elif name == "dora":
        return dora(msg)
    elif name == "alice":
        return alice(msg)
    elif name == "bob":
        return bob(msg)
    else:
        socket.send("{}:I am not a bot, goodbye".format(name).encode())
        socket.close()
        exit()
```

The name in this function is taken from input, as shown in requirement 1. As we see from the screenshot above, each bot has a different function:

```
def alice(a):
    # List of verbs that the bot likes, dislikes and hates
    dislikes = ["cry", "hike", "run", "walk", "swim", "sleep", "cry", "fight", "drink", "party", "climb", "kick"]
    likes = ["shoot", "kill", "work", "think", "look", "find", "sing", "drive", "perform", "build", "create"]
    hates = ["develop", "code", "laugh", "fly", "cook", "wash", "talk", "hang", "paint", "dive", "buy"]
    return response.findresponse(likes, dislikes, hates, a)
```

This is Alice as an example. All it does is list what the bot likes, dislikes, and absolutely hates. From that it generates a response. This function generates the response:

```
def findresponse(likes, dislikes, hates, a):
    # Makes an iterable list containing all the words sent from the server
    wordlist = a.split()
    for word in wordlist:
        # 3 next if statements checks if the word is in my list of actions and returns an appropriate response from
        # the bot
        if word in likes:
            previous_suggestions.append(word)
            return random.choice(positiveresponses).format(word + "ing")
        elif word in dislikes:
            return random.choice(neutralresponses).format(word + "ing")
        elif word in hates:

            # If the bot hates the suggestion, there is a 1 in 25 chance the bots response will be a swear word
            if random.randint(0, 25) == 1:
                return "f you"
            else:
                # If the bot hates this suggestion, and liked a previous suggestion
                # it will suggest that instead
                if len(previous_suggestions) > 0:
                    sugg1 = word + "ing"
                    sugg2 = random.choice(previous_suggestions) + "ing"
                    return "I don't really like {}, but i liked your previous suggestion" \
                           " {} much better. I would rather do that".format(sugg1, sugg2)

                else:
                    return random.choice(negativeresponses).format(word + "ing")

    return "I don't understand, sorry"
```

As you can see, I have not done too much in regards to the kinds of responses generated.

```
# Lists of responses
positiveresponses = ["{} sounds great, i would love that", "I havent gone {} in a long time, sounds fun",
                     "Good idea, i would love to go {}",
                     "I like your idea, maybe we can do something else after {}?"]
negativeresponses = ["I'm usually open to doing most things but I really hate {}",
                     "I will never do that, i hate {}",
                     "{} sucks", "If you don't have any other suggestions, I'm in"]
neutralresponses = ["{} is not my cup of tea, but I guess I can give it a try",
                    "I'm not sure if i'm up for {} today",
                    "If {} is the only option, sure.. but I would prefer to do something else"]
```

Due to the limitations of the task (No AI or NLTK), there is not much else you can do than my solution. Though I have put in some creativity. For example, if the suggestion is something the bot hates, there is a 1 in 25 chance the response will be ‘f you’ which causes the server to kick the client out. Here again, I could randomize it more, and give more options, but I didn’t see the necessity since it’s not the main focus of the task. If a previous suggestion is something the bot likes, the bot will suggest that instead.

2b

All I have done with responses from other clients is print them out.

```
if "responsex29" in x:
    x = x.replace("responsex29", "")
    print(x)
```

All we do is remove the identifier ‘responsex29’, and then print the message.

Requirement 3

“You are free to decide how and when to end the connection”

The only way to disconnect from the client side, is if you take too long to respond. I have given options for the user to disconnect a client from input from the server.

Requirement 4

“Sending a message that might be dropped if the client is not ready to receive messages”

I’m not sure what this means. So since it’s optional, I decided to drop it.

Server requirements

Requirement 1

“Accept any connection. You can expect all connections to be a bot, e.g. that they will not be the first to speak, but that they will always respond.”

On this requirement, I struggled a bit with choosing the best solution for my program. At first, I did attempt to create a list with all connections, and perform operations on that list iteratively. This worked out fine, but since I finished this quite early, I decided to go for the proposed select-module even though I’m new to socket programming. This allowed me to make a 100% single-threaded solution, instead of dealing with multiple threads like an iterative approach would.

```
while True:

    if len(outputs) == 0:
        print("No clients are connected, the program will wait until you connect clients")
        # Creates r-list, w-list and x-list and updates it each iteration in the loop
    read, write, exception = select.select(inputs, outputs, inputs)
    # Allows user to connect up to 2 bots between sequences, and not just 1 per msg sequence
    for o in range(4):
        for s in read:
            if s is sock:
                # Accepts connection from socket
                c, addr = s.accept()
                try:
                    name = c.recv(1024).decode()
                except Exception as e:
                    pass

                if name in botlist: # The program only allows 1 instance of each bot
                    c.close()
                    read, write, exception = select.select(inputs, outputs, inputs)
                    print("Bot {} is already connected, you can't have 2 instances of one bot in the program".format(
                        name))
                    time.sleep(1)
                else:
                    # Updates the lists with the new connection
                    inputs.append(c)
                    outputs.append(c)
                    botlist.append(name)
                    read, write, exception = select.select(inputs, outputs, inputs)
                    print("New connection", addr)
                    time.sleep(1)
                    print(name + " will take part in the conversation now")
                    time.sleep(1)
```

This is the needed code for the requirement. The select.select statement creates 3 iterable lists, read is a list of sockets ready to read, for example if a client is waiting to connect to the socket, the read-list will have 1 item. write is a list of sockets ready to receive messages, and exception is a list of sockets throwing an exception. However, I won’t be needing the exception list as I deal with exceptions in other ways.

Requirement 2

“Initiate a round of dialogue by suggesting an action. Send the suggestion to each of your connected clients. The action can be random, provided as user input for each.”

```
    # The program needs at least 2 connections to be able to start
    if len(outputs) < 2:
        print("Please connect more clients to continue")
        time.sleep(3)
    else:
        # The user decides on input whether he wants a sequence of random messages,
        # kick a user or write their own message to the bots
        print("You can send a new message now\n'r' = sequence of randomly generated messages(5)\n"
              "'n' = message from input\n'kick [botname]' "
              "to kick bot from conversation\n'q' to disconnect and terminate the program")
        x = input()
        if x == "r":
            if len(outputs) > 0:
                for t in range(5):
                    global_message = gensuggestions()
                    broadcast(sock, global_message)
                    if len(outputs) > 0:
                        print("Sequence done...")
                        time.sleep(1)
                time.sleep(1)

        elif x == "n":
            print("Input message")
            global_message = input()
            broadcast(sock, global_message)
            print("Sequence done....")
            time.sleep(1)
        elif "kick" in x:
            kickbot(x)
        elif x == "q":
            sock.close()
            break;
```

Within the same while(True)-loop, the program checks if we have at least 2 connections. If we do, the user gets the option of writing their own message, generating random or kicking out a user. Whenever a message is getting sent, we will go to the broadcast function, which looks like this;

```
def broadcast(c, message):
    global read, write, exception
    if len(outputs) == 0:
        return
    # If the message is coming from the host
    if c is sock:
        print("Me: " + message)
        time.sleep(1)
        # Iterates through sockets ready to receive a message
        for f in write:
            # When we reach a client
            if f is not sock:
                # A good program will check if clients are
                # still connected before trying to interact with them, this is done with a try-except
                try:
                    f.send(message.encode())
                    f.settimeout(2)
                except Exception as e:
                    kickcon(f, "unknown reason")
                    if len(outputs) == 0:
                        sendback()
                        return
                # If the sending was successful,
                # we will receive the response from the client. try-except in case of a timeout
                else:
                    try:
                        x = f.recv(1024).decode()
                        response_list.append(x)
                        # If the response is a bad word, the client will be kicked out
                        if "f you" in x:
                            kickcon(f, "misbehaving")
                            if len(outputs) == 0:
                                return
                    # In case of a timeout, we will kick the client
                    except socket.timeout as e:
                        kickcon(f, "taking too long to respond")
```

```
                    kickcon(f, "taking too long to respond")
                    if len(outputs) == 0:
                        sendback()
                        return
sendback()
read, write, exception = select.select(inputs, outputs, inputs)
```

Having the “if c is sock:” statement might be unnecessary in this case, as we can assume only the sock will use this function. We will send back the responses to other clients by using a different function “sendback()” which I will cover later. The broadcast function handles 2 exceptions: first try-except checks if the client is still connected. If not, it will be removed from the outputs list through the kickcon() function which I will cover later. The other exception is if the client is taking too long to respond. We also have a special case, if the message from a client is “f you”. Here I could complicate things more by adding a function to check against a list of banned words. However, I tried to keep it simple and show how it could be done with 1 simple message.

At the end of the function, the sendback-function is called to send out all responses to all clients.

Requirement 3

“All responses should be sent back out to all clients except the one who sent it.”

```
def sendback():  # Sends back all responses to all clients except the one it came from

    for f in range(len(response_list)):  # Iterates through the response list
        print(response_list[f])
        time.sleep(1)
        sending = response_list[f] + " responsex29"  # Before sending, we append the code responsex29 so the client
        # knows it's a response from another client
        for s in write:
            # A response should not be sent back to where it came from
            if write.index(s) != f:
                # Try-except in case the client has disconnected
                try:
                    s.send(sending.encode())
                except:
                    pass
    for x in errors:
        print(x)
        time.sleep(1)
    # Clears the lists of responses and errors for the next msg sequence
    response_list.clear()
    errors.clear()
```

To fulfil this requirement, we can choose to send the responses out immediately after the message is received. However, I have chosen to put them in a list of responses and send them out first after I have collected all responses. I also chose to print responses in this function, as well as error messages. This is done with the function above.

Requirement 4

“Maintain a list of connected clients. If you want, you can let new connections wait until you've completed one round of dialogue. A good program will check if clients are still connected before trying to interact with them. If they're not, or if you decide that they're taking too long to respond, you can remove them from the list of connections.”

Most of these requirements are already covered by using the select module. The list of connected clients is made and regularly updated through the ‘read, write exception = select.select(…)’. If a message sequence has started, the new connection has to wait for it to finish. I have also set a limit of 4 new connections per new sequence by having the ‘for o in range(4)’ (read requirement 1).

Since I have the try-except in the broadcast-function, it doesn't really check if the client is connected BEFORE it sends a message. However, it does catch the exception that occurs if it's not, and I consider that to be good enough. See requirement 2.

I also have the try-except to check if the client uses too much time before it responds. See requirement 2.

Requirement 5

"You are free to decide when and how to disconnect the clients (you can even kick them out if they misbehave) and how to gracefully terminate the program."

I have partly covered this already. The user gets to decide if it wants to kick a specific client out, or terminate the program altogether. There are also options for kicking out a client if it misbehaves, but as I specified I decided to not complicate that bit too much, and just show 1 example of it.

Requirement 6

"Make a "bot" that takes its response from the command line. That way you or other users can interact with the bots and make the dialogue more interesting"

I'm not entirely sure what this means exactly. However, I interpreted it as a bot that receives messages from the host and responds from input. This is a good opportunity to test if the timeout-exception is working properly. As with the other bots, this one is constantly listening for messages:

```
def soria():
    while True:
        msg = socket.recv(1024).decode()
        if "responsex29" in msg:
            msg = msg.replace("responsex29", "")
            print(msg)
        else:
            print(msg)
            print("reply: ")
            reply = botname + ": " + input()
            socket.send(reply.encode())
            reply = reply.replace(botname + ":", "")
            print("Me:", reply)
```

When it receives a message, all it does is ask the user for input and sends it right back to the host.

Requirement 7

"Don't nag your users."
This requirement is open for interpretation. I have chosen to give options for the user regularly. However, when the user requests a sequence of random messages, 5 are generated automatically.

Demonstration of program

To test the program properly I should try out a couple of different things and see if I encounter any problems.

Before I start, these are the -help pages for the server and client, in that order:

```
PS C:\Users\Johan\PycharmProjects\individual-portfolio-assignment-1-Githansen> python server.py -h
The program must be invoked by taking in the port number as a parameter
Example: 'python server.py 8080'
After starting the program, you have to connect at least 2 clients before starting
When you are ready, you get the option of sending your own messages, or make the program generate some at random
Don't worry, instructions will be given along the way.
```

```
PS C:\Users\Johan\PycharmProjects\individual-portfolio-assignment-1-Githansen> python client.py -h
The program must be invoked by giving 3 parameters -> IP-address, port number and bot name
Example: python client.py 192.168.56.1 8080 bob
For it to work, the bot must be one of bots available, or else the server will kick you out
Bot list:
bob
alice
dora
chuck
soria (all replies from user input)
```

First, I will try a few iterations with all bots connected, except Soria which requires user input for every reply.

When starting, the user gets a message that at least 2 clients need to be connected:

```
PS C:\Users\Johan\PycharmProjects\individual-portfolio-assignment-1-Githansen> python server.py 8080
No clients are connected, the program will wait until you have at least 2 connected clients
New connection ('192.168.56.1', 61859)
alice will take part in the conversation now
Please connect more clients to continue
New connection ('192.168.56.1', 61860)
bob will take part in the conversation now
New connection ('192.168.56.1', 61861)
dora will take part in the conversation now
```

When clients connect, a message will be printed that for example ‘bob will take part in the conversation now’ as well as a statement showing where the connection is from.

```
You can send a new message now...
'r' = sequence of randomly generated messages(5)
'n' = message from input
'kick [botname]' to kick bot from conversation
'q' to disconnect and terminate the program
```

Next, a message gets printed and asks for user input. The user can choose to input their own message, generate some random messages, kick out a bot or end the program. To start off, we generate some random messages.

```
Me: How do you feel about going to drive?
alice: driveing sounds great, i would love that
bob: f you
dora: I will never do that, i hate driveing
chuck: driveing is not my cup of tea, but I guess I can give it a try
bob has been kicked out for misbehaving
Sequence done...
Me: How do you feel about going to drive?
alice: driveing sounds great, i would love that
dora: I will never do that, i hate driveing
chuck: If driveing is the only option, sure.. but I would prefer to do something else
Sequence done...
Me: I feel like going to run , how about you?
alice: I'm not sure if i'm up for runing today
dora: runing sounds great, i would love that
chuck: I havent gone runing in a long time, sounds fun
Sequence done...
Me: Do you want to drink?
alice: I'm not sure if i'm up for drinking today
dora: Good idea, i would love to go drinking
chuck: drinking sounds great, i would love that
Sequence done...
Me: How do you feel about going to create?
alice: Good idea, i would love to go createing
dora: I don't really like createing, but i liked your previous suggestion runing much better. I would rather do that
chuck: I'm not sure if i'm up for createing today
Sequence done...
You can send a new message now...
'r' = sequence of randomly generated messages(5)
'n' = message from input
'kick [botname]' to kick bot from conversation
'q' to disconnect and terminate the program
```

As you can see, the program worked out fine. The bot bob triggered the 1 in 25 chance rude response and got kicked out. After he got kicked, the program continued without issues. As we can see, there are some grammatical errors, which could be fixed somewhat, but as mentioned I focused more on the functionality than the content itself, since that seems to be the main focus of the assignment. After the sequence of messages is over,

For the next round of messages, I will close one of the client-programs forcefully, so that it disconnects so I can see how the server handles it.

```
Me: How do you feel about going to buy?
alice: I don't really like buying, but i liked your previous suggestion working much better. I would rather do that
dora: I'm not sure if i'm up for buying today
chuck has been kicked out for a client side error or disconnection
Sequence done...
Me: How do you feel about going to paint?
alice: I don't really like painting, but i liked your previous suggestion working much better. I would rather do that
dora: I'm not sure if i'm up for painting today
Sequence done...
Me: How do you feel about going to perform?
alice: I havent gone performing in a long time, sounds fun
dora: I don't really like performing, but i liked your previous suggestion drinking much better. I would rather do that
Sequence done...
Me: I feel like going to talk , how about you?
alice: I don't really like talking, but i liked your previous suggestion performing much better. I would rather do that
dora: I'm not sure if i'm up for talking today
Sequence done...
Me: I feel like going to develop , how about you?
alice: I don't really like developing, but i liked your previous suggestion working much better. I would rather do that
dora: developing is not my cup of tea, but I guess I can give it a try
Sequence done...
You can send a new message now...
'r' = sequence of randomly generated messages(5)
'n' = message from input
'kick [botname]' to kick bot from conversation
'q' to disconnect and terminate the program
```

As we can see, the program notifies the user that Chuck has been disconnected, and then continues as usual. In the requirements walkthrough, I mentioned that the bot can suggest an activity that was suggested earlier. Here, Alice suggests working, since she remembers it from the first iteration of messages.

In the next iteration, I will do the same, just with all the bots so no one are connected when the messages start.

```
r
Me: How do you feel about going to dive?
alice has been kicked out for a client side error or disconnection
dora has been kicked out for a client side error or disconnection
No clients are connected, the program will wait until you have at least 2 connected clients
```

Here, a message is sent out the list of connections. As stated in the requirements, a good program will check if the clients are still connected before sending it. This is done with a try-except, as stated in requirement 4 for the server.

Since no clients are connected, the program stops and tells the user that more clients are needed to continue.

Now, I will reconnect a few bots, among them Soria which takes all replies from user input. The requirements state I can kick out a bot if it takes too long to respond. Having Soria gives me a good opportunity to test it.

```
Me: Do you want to party?
chuck: I havent gone partying in a long time, sounds fun
dora: Good idea, i would love to go partying
soria has been kicked out for taking too long to respond
Sequence done...
Me: I feel like going to fight , how about you?
chuck: Good idea, i would love to go fighting
dora: I havent gone fighting in a long time, sounds fun
Sequence done...
Me: I feel like going to sleep , how about you?
chuck: Good idea, i would love to go sleeping
dora: sleeping sounds great, i would love that
Sequence done...
Me: I feel like going to dive , how about you?
chuck: I don't really like diveing, but i liked your previous suggestion partying much better. I would rather do that
dora: diveing is not my cup of tea, but I guess I can give it a try
Sequence done...
Me: How do you feel about going to cook?
chuck: I don't really like cooking, but i liked your previous suggestion fighting much better. I would rather do that
dora: cooking is not my cup of tea, but I guess I can give it a try
Sequence done...
```

Here, Soria takes too long to respond and gets disconnected from the program. After that it continues without problems.

Now I will try to kick a client through using the kick-command.

```
You can send a new message now...
'r' = sequence of randomly generated messages(5)
'n' = message from input
'kick [botname]' to kick bot from conversation
'q' to disconnect and terminate the program
kick chuck
chuck has been kicked out
Please connect more clients to continue
```

Chuck gets kicked out, and we now only have 1 connection. The program waits until at least 2 are connected, again.

My program only allows 1 instance of each bot. Let's see what happens if I connect another instance of a bot. Let's try with chuck:

```
Bot chuck is already connected, you can't have 2 instances of one bot in the program
```

This message shows up on the server-side, and the program continues.

Next, I'll try a message with user input:

```
Bot chuck is already connected, you can't have 2 instances of one bot in the program
You can send a new message now...
'r' = sequence of randomly generated messages(5)
'n' = message from input
'kick [botname]' to kick bot from conversation
'q' to disconnect and terminate the program
n
Input message
Let's go to work together!
Me: Let's go to work together!
dora: I don't really like working, but i liked your previous suggestion fighting much better. I would rather do that
chuck: working is not my cup of tea, but I guess I can give it a try
alice: I like your idea, maybe we can do something else after working?
Sequence done...
```

Finally, I will try to connect a bot mid-conversation.

```
Me: How do you feel about going to sleep?
dora: sleeping sounds great, i would love that
chuck: I like your idea, maybe we can do something else after sleeping?
alice: I'm not sure if i'm up for sleeping today
Sequence done...
Me: I feel like going to cry , how about you?
dora: Good idea, i would love to go crying
chuck: Good idea, i would love to go crying
alice: If crying is the only option, sure.. but I would prefer to do something else
Sequence done...
Me: How do you feel about going to find?
dora: I don't really like finding, but i liked your previous suggestion sleeping much better. I would rather do that
chuck: If finding is the only option, sure.. but I would prefer to do something else
alice: I havent gone finding in a long time, sounds fun
Sequence done...
Me: Do you want to build?
dora: I don't really like building, but i liked your previous suggestion crying much better. I would rather do that
chuck: building is not my cup of tea, but I guess I can give it a try
alice: I havent gone building in a long time, sounds fun
Sequence done...
Me: Do you want to swim?
dora: I havent gone swiming in a long time, sounds fun
chuck: I havent gone swiming in a long time, sounds fun
alice: I'm not sure if i'm up for swiming today
Sequence done...
New connection ('192.168.56.1', 62862)
bob will take part in the conversation now
You can send a new message now...
```

I made the connection after the second message in this sequence, and as we can see, the program waited until the sequence was over before allowing bob to take part in the conversation.

On the client side of the program, the responses are printed, but doesn't do much more than that:

```
New sequence...

Host:I feel like going to cry , how about you?
Me:  crying is not my cup of tea, but I guess I can give it a try
chuck: I like your idea, maybe we can do something else after crying?
```

I believe I have demonstrated most properties of the program.

Conclusion

As I have demonstrated, although the program is simple in its responses and messages, it works as required by the task. The focus was mostly on the functionality of the program, and not the content of messages etc. as it was not listed as a requirement for the task. I did attempt a few solutions before landing on using the select.select in python. I tried using multiple threads, and simply iterating

through lists to send and receive messages. Although these solutions worked, I wanted to do something more challenging, and went with the select instead. All in all, I believe it worked out fine.

[i]

[1] https://oslomet.instructure.com/courses/23100/pages/no-lecture-8-portfolio-1-guidance-session?module_item_id=397071 , Portfolio 1 guidance session

[2] https://app.slack.com/client/T02TBGWFFPY/C02TLJHE60L, Slack channel for DATA2410

[3] https://docs.python.org/3/library/sys.html, (Python, 2022)Python documentation, sys module