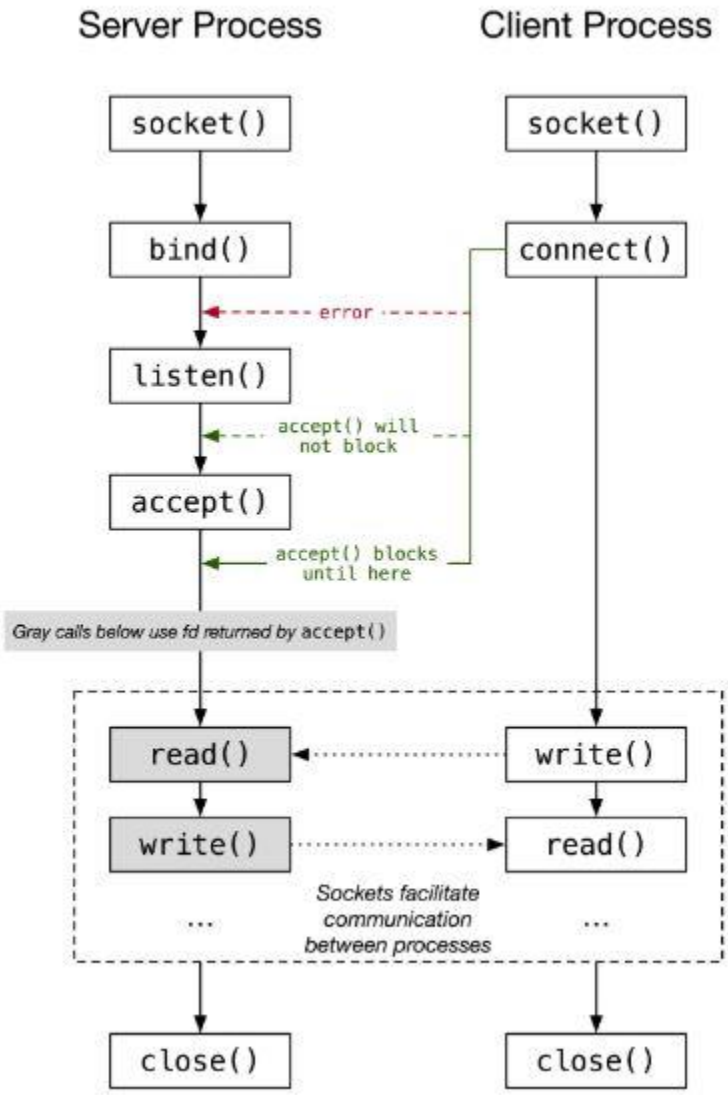
CMSC 137 Lab Activity 5

For the past month, you have learned about connecting computers to a network and configuring their IP addresses. By doing so, you were able to play and transfer different files between computers. For this lab activity, we will discuss how a simple communication is done between two nodes in a network.

Client-Server model is something that you have probably learned in your CMSC 10 but if you forgot, this is just a relationship in which one program, the client, requests a service or resource from another program, the server. Commonly, there are multiple client but only one server. For this activity, we will just focus on a simple communication (Sending/Receiving messages). When the server is initialized (turned on) with its current IP address and port to listen at, it will wait for the client to connect to it. Once it sees that the client is connecting, it will accept it and start communication. A visualization of this process is provided below.



If you notice, there is a socket function in the diagram. This is just initializing the socket, otherwise known as an endpoint of a communication. An implementation of this is called Socket programming. Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

The code snippet below shows the implementation of the server model. The first block is getting and setting the necessary information for a socket to be used as a server. The IP address along with the port number is saying that the server device is waiting for communication at this particular port. A common port number is 80 which is the default port for browser access. The 2nd and 3rd block waits for a client and then accepts the connection. In the 4th block, the server sends a message and waits for the message from the client. This process is looped until the client sends "[bye]".

```python
import time, socket, sys
#Get the hostname, IP Address from socket and set Port
soc = socket.socket()
host_name = socket.gethostname()
ip = socket.gethostbyname(host_name)
port = 1234
soc.bind((host_name, port))
print(host_name, '({})'.format(ip))
name = "Referee"

#Waiting for client
soc.listen()
print('Waiting for incoming connections...\n')

#Client Connecting
connection, addr = soc.accept()
print("Received connection from ", addr[0], "(", addr[1], ")")
print('Connection Established. Connected From: {}, ({})'.format(addr[0], addr[0]))
client_name = connection.recv(1024)
client_name = client_name.decode()
print(client_name + ' has connected.\n')
connection.send(name.encode())

#first message
message = "Hi"
#Convo loop
while True:
    if message == '[bye]':
        message = 'Good Night...'
        connection.send(message.encode())
        print("")
        break

    connection.send(message.encode())
    message = connection.recv(1024)
    message = message.decode()
    print(client_name, '>', message)
    message = input(f"Server > ")
```

Console output for the server model is shown below.

```
Server turning on
DESKTOP-SQ4HKEC (10.49.[redacted])
Waiting for incoming connections...

Received connection from  10.49.[redacted] ( 59797 )
Connection Established. Connected From: 10.49.[redacted], (10.49.[redacted])
Person1 has connected.

Person1 > Hello
Server > What can I do for you today?
Person1 > Dunno
Server > So
Person1 > Leaving the Chat room
Server >
```

The following is an implementation of the client model. It is similar to the server model but it does not listen to incoming connections, it simply connects to the server. It is also important that the port number is the same with the server so that communication can occur.

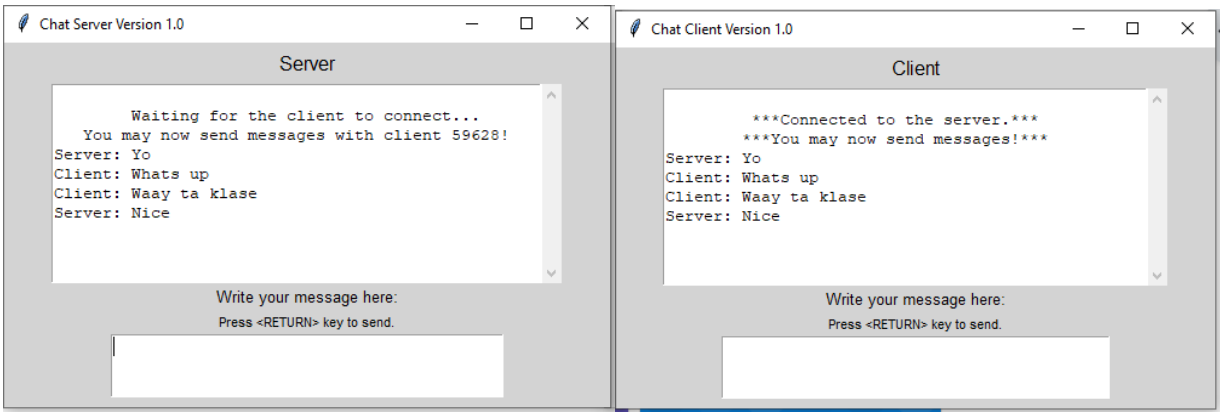
```python
import time, socket, sys
#Get the hostname, IP Address from socket
soc = socket.socket()
shost = socket.gethostname()
ip = socket.gethostbyname(shost)

#get information to connect with the server
print(shost, '({})'.format(ip))

#Setting up names and connection and port
server_host = input('Enter server\'s IP address:')
name = input('Enter Client\'s name: ')
port = 1234

#Connecting to Server
print('Trying to connect to the server: {}, ({})'.format(server_host, port))
time.sleep(1)
soc.connect((server_host, port))
print("Connected...")
soc.send(name.encode())

#listening to server
server_name = soc.recv(1024)
server_name = server_name.decode()
print('{} has joined...'.format(server_name))

#Convo loop
while True:
    message = soc.recv(1024)
    message = message.decode()
    print(server_name, ">", message)
    message = input("Me > ")
    if message == "[bye]":
        message = "Leaving the Chat room"
        soc.send(message.encode())
        print("")
        break
    soc.send(message.encode())
```

Below is the console output of the client model.

```
Client Server...
DESKTOP-SQ4HKEC (10.49.[redacted])
Enter server's IP address:10.49.[redacted]
Enter Client's name: Person1
Trying to connect to the server: 10.49.[redacted] (1234)
Connected...
Referee has joined...
Referee > Hi
Me > Hello
Referee > What can I do for you today?
Me > Dunno
Referee > So
Me > [bye]
```

This implementation however has a setback. The first is that each one has to wait for the other to send a message before sending another message. Even if you want to send multiple messages, you will not be able to as long as the other has not sent one back. Another is that this is limited to only one client. The server will not be able to accept any other connection as long as the current client has not ended the communication.

Hence, for this activity, I want you to work by pairs (preferably but individual is fine) to **create a Client-Server model using socket programming**. The servers should be able to accept multiple clients and see what each client has sent. The client should also be able to send messages freely (whenever and how many they want). There should also be a GUI for this output. I do not expect you to have the same design as the ones below, they are just an example. Basically, I want you to **create a group chat**.



You may explore the socket, tkinter, and threading modules in python for a start. Submit the codes as a ZIP file on LMS with the format ***Person1&Person2-GC.zip***. After which, you may have your code defense.

Deadline: November 13, 2024