Design & Implemenation: Internet Voice & Text Chatting Program

Tan Wei Xuan (49003140) tanweixuan@postech.ac.kr

Zhang Xin Yue (49003143) xyzhang@postech.ac.kr

April 1, 2019

1 Project Overview

The **Internet Text and Voice Messaging Program** that I have implemented is a **Multi-threaded Chat Application** that utilises the Transmission Control Protocol *(TCP)* and allows for mutiple clients to send/receive text and voice messages over a server, to one another, at the same time. We switched from using *Java* to *Python* as *Python* provides more API support for voice chat. The source code files are as follow:

1. **Server Files**
   - server.py
2. **Client Files**
   - client.java

2 Environment and Dependencies

My program is written in **Python 3.6.7** and is developed on **Windows running the Ubuntu 18.04.2 bit subsystem**, using **PyCharm** as the **Integrated Development Environment**. *The program may not work as inteneded if it is ran on other environments.* ***Ensure that these dependencies are being installed on your system.***

1. PyAudio
2. Threading
3. libasound

3 Implementation - Text Chat

3.1 Client - Receving Text

Below is the handler for how a client receives text sent from other clients through the server. The Client is binded to the Server Text Port on 3001.

```
#Client.py
def receive_text():
  while True:
    try:
      msg = client_socket_text.recv(BUFSIZ).decode("utf8")
      print(msg)
    except OSError:
      break
```

3.2 Client - Sending Text

Below is the handler for how a client sent a text to the server and to other clients on the server. The Text and Video Port connection will be closed if the Client inputs *"quit"*.

```
#Client.py
def send_text():
  while True:
    msg = input()
    print('\033[A\033[A')
    client_socket_text.send(bytes(msg, "utf8"))
    if msg == "{quit}":
      client_socket_text.close()
      client_socket_voice.close()
      quit()
```

3.3 Server - Accept Connection to Text Port

Below is the handler for how the server accepts a connection from the client onto the Text Port. The Text Port that the client connect to is /textitP3001. Whenever a client is connected to the port, a thread if attached to the client to handle it.

```
#Server.py
#For handling client's connection to TEXT PORT, assign a thread for each
    connection
def accept_text():
  while True:
    client, client_address = SERVER_TEXT.accept()
    print("%s:%s␣has␣connected␣from␣." % client_address)
    client.send(bytes("Enter␣your␣name:␣", "utf8"))
    addresses_text[client] = client_address
    Thread(target=handle_client_text, args=(client,)).start()
```

3.4 Server - Handling Text

Below is the handler for how the Server recevies a message from a client and subsequently broadcasts the message to all the other clients on the server.

```
#Server.py
def handle_client_text(client):
  #Handles a message from the given socket.
  #Receive Message
  name = client.recv(BUFSIZ).decode("utf8")
  welcome = 'Welcome␣%s!␣Type␣{quit}␣to␣exit␣the␣chat␣room.' % name
  client.send(bytes(welcome, "utf8"))
  msg = "%s␣has␣entered␣the␣chat␣room!" % name
  broadcast(bytes(msg, "utf8"))
  clients_text[client] = name

  while True:
    msg = client.recv(BUFSIZ)
    if msg != bytes("{quit}", "utf8"):
      #Append name to front of message
      broadcast(msg, name + ">>␣")
    else:
      client.send(bytes("{quit}", "utf8"))
      client.close()
      erase_client(client)
      broadcast(bytes("%s␣has␣left␣the␣chat␣room." % name, "utf8"))
      break
```

4 Implementation - Voice Chat

We utlisied the PyAudio library to implement the voice chat portion of our application.

4.1 Client - Declaring the Input/Output Streams

We have to declare the Input and Output Stream for sending and receiving audio on our client. *stream send* is for sending our voice input to the server, on onto the other clients on the server. *stream receive* is for receving and playing the voice output from the server, which is sent from the other clients on the server.

```
#Client.py
#For sending stream over to the server
stream_send = p.open(
  format=pyaudio.paInt16,
  channels=CHANNELS,
  rate=RATE,
  input=True,
```

```
  frames_per_buffer=CHUNK)

#For receving stream from the server and playing the stream
stream_recv = p.open(
  format=p.get_format_from_width(WIDTH),
  channels=CHANNELS,
  rate=RATE,
  output=True,
  frames_per_buffer=CHUNK)
```

4.2 Client - Sending/Receiving Audio

The client have to handles both sending/receiving voice simulatanuously. The below two functions are for receiving and playing the voice sent from the server and for sending voice inputs from the client to the server.

```
#Client.py

# For Receiving Voice
def receive_voice():
  while True:
    try:
      data = client_socket_voice.recv(BUFSIZ)
      stream_recv.write(data)
    except OSError:
      break

# Send voice to the server
def send_voice():
  while True:
    try:
      data = stream_send.read(CHUNK)
      client_socket_voice.sendall(data)
    except OSError:
      break
clientSocket.close();
```

4.3 Server - Accept Connection to Voice Port

Below is the handler for how the server accepts a connection from the client onto the Voice Port. The Voice Port that the client connect to is /textit50007. Whenever a client is connected to the port, a thread if attached to the client to handle it.

```
#Server.py
 For handling client's␣connection␣to␣VOICE␣PORT,␣assign␣a␣thread␣for␣each␣
```

```
    connection
def␣accept_voice():
␣␣while␣True:
␣␣␣␣client,␣client_address␣=␣SERVER_VOICE.accept()
␣␣␣␣addresses_voice[client]␣=␣client_address
␣␣␣␣Thread(target=handle_client_voice,␣args=(client,)).start()
```

4.4 Server - Handling Voice

Below is the handler for how the Server recevies a voice message from a client and subsequently broadcasts the message to all the other clients on the server.

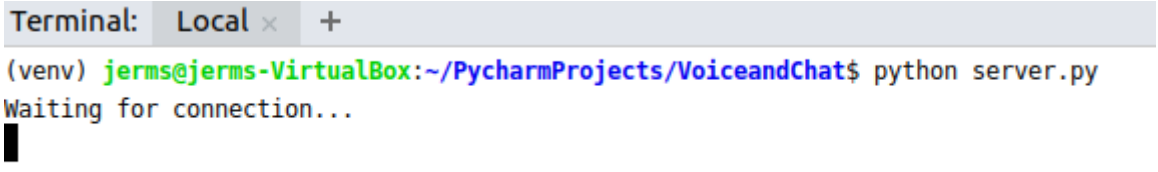
```
#Server.py
def handle_client_voice(client):
  """
  Handles Voice from the given socket.
  """
  # Receive Voice
  clients_voice[client] = 1

  while client in clients_voice:
    try:
      data = client.recv(BUFSIZ)
      broadcast(data, dtype='voice')
    except Exception as _:
      client.close()
      del clients_voice[client]
```

5 Running the Program

*On the client, the input for host should be **127.0.0.1** unless a specific host has been defined.*

5.1 Terminal

1. In the terminal, run the command ***python Server.py*** to start the **server**.



2. In the terminal, run the command ***python Client.py*** to start the **client**. (Run ***n*** times to start ***n*** clients)

```
(venv) jerms@jerms-VirtualBox:~/PycharmProjects/VoiceandChat$ python client.py
Enter the Host: 127.0.0.1
Enter your name:
Welcome Jerms! Type {quit} to exit the chat room.
Jerms>> Hi there
```

3. Voice is constantly is being sent & received, so you do not have to input any additional commands to enable voice chat.

4. In the terminal, type **{quit}** to exit the from the server.

```
(venv) jerms@jerms-VirtualBox:~/PycharmProjects/VoiceandChat$ python client.py
Enter the Host: 127.0.0.1
Enter your name:
Welcome Jerms! Type {quit} to exit the chat room.
Jerms>> Hi there
{quit}
(venv) jerms@jerms-VirtualBox:~/PycharmProjects/VoiceandChat$
```