Design & Implemenation: Internet Voice & Text Chatting Program

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

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

April 9, 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.py

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**

```
#On Ubuntu Terminal
$ sudo apt-get install python-pyaudio python3-pyaudio
```

2. **Threading**

```
#On Ubuntu Terminal
$ sudo apt install python3-pip #Only if pip is not installed
$ pip install threaded
```

3. **libasound**

```
#On Ubuntu Terminal
$ sudo apt-get install libasound2-devf
```

4. **Pillow**

```
#On Ubuntu Terminal
$ pip install Pillow==2.2.1
```

5. **OpenCV**

```
#On Ubuntu Terminal
$ pip install opencv-python
```

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 *2222*.

```
#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 *2222*. 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. The Client is binded to the Server Voice Port on *55555*.

```
#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 *55555*. 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')
```

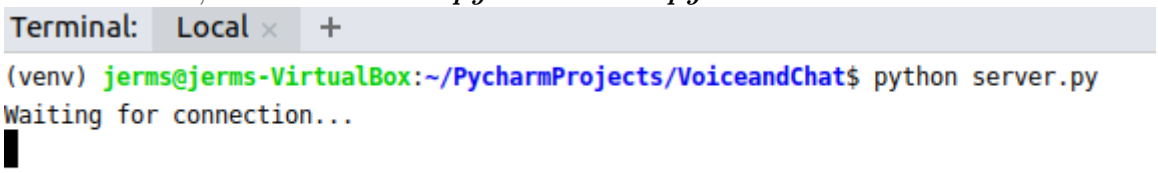
```
    except Exception as _:
      client.close()
      del clients_voice[client]
```

5 Running the Program

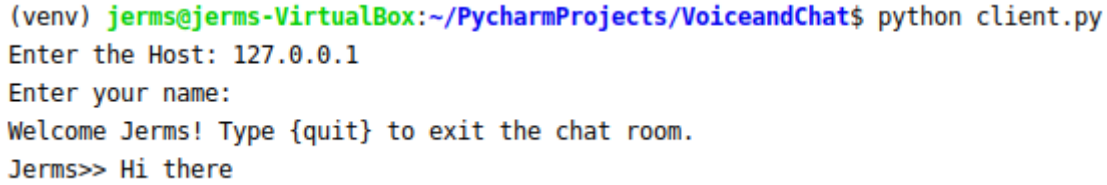
*On the client, the input for host should be **127.0.0.1** unless a specific host has been defined.*

5.1 Terminal

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

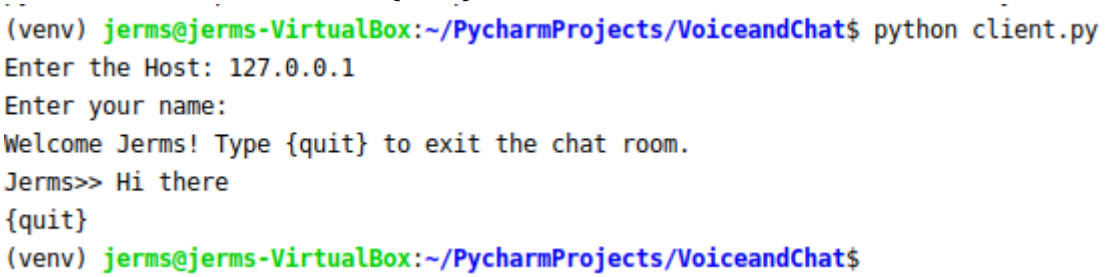


2. In the terminal, run the command ***python Client.py #hostname*** to start the **client**. (Run ***n*** times to start ***n*** clients). By default the host name will be **127.0.0.1**



3. When more than 1 user has connected to the server, both **Video** and **Voice** will be automatically enabled. You do not have to input any additional commands to enable them. **Video, Voice and Text** can be sent/received simultaneously.

4. In the **GUI Text Box**, type **{quit}** to exit from the server.



6 Limitations

Our Program currently only allow peer-to-peer video communications (between two parties). Further improvement can be made to allow more than 2 clients to connect to the server and chat through video communications at any point of time. The Video Stream sent are compressed through the conversion into JPEG images and thus the resolution for then recevied Video

Stream will not be of the best quality. We could implement better compression algorithms for the Video Stream to improvde the Video quality in the future.