Computer Networks Assignment 3 Report

Internet Text/Voice Chatting Program

정건화 Geonhwa Jeong, CiTE, 20130392

김준영 Junyoung Kim, CiTE, 20140263

1. *Introduction*

The goal of this assignment is to implement a fully functional one to one text and voice chatting program. The key function of this program is that both features (text and voice chatting) need to happen simultaneously, and are non-blocking, which means that users can send messages regardless of whether the other user is sending or not.

This report is made up of three major parts, design, implementation, discussion. In the design part, we will explain the basic plan we used when structuring our program. In the implementation part, we show how we actually implemented our design. In the discussion part, we bring up some difficulties we encountered, and explain how we solved them.

By completing this assignment, we were able to understand the basic nuances of sending different types of data. We were also able to study about various ways to implement non-blocking chatting such as using threads.

2. *Design*

This program should be able to handle both voice and text data at the same time, so that the users can fully enjoy text/voice chatting simultaneously. At first, we tried to extend our program that we implemented for text chatting in assignment 2, but it was pretty inefficient since the program was based on c++ language.

After searching for libraries that we can use for handling I/O devices, we decided to implement from the very beginning using python with pyaudio library. PyAudio make us use python bindings for PortAudio easily so that we can handle I/O devices.

The main feature that this program should provide is the availability to speak, listen, send text, and receive text simultaneously. To achieve this goal, we decided to use separate threads for each function so that one doesn't block another. The main process handles receiving voice data, and create three other threads for handling other functionalities: sending voice data, sending text data, and receiving text data.

We used server-client model for this program. Server user can talk to client user, and client user also can talk to server user. Therefore, this program is based on 1-1 chatting. Even though we use different term for 'server' user and 'client' user, users can enjoy same functionalities mentioned above.

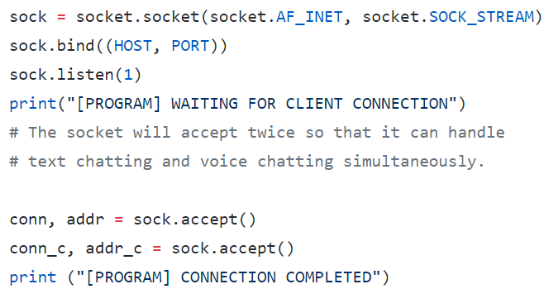
3. *Implementation*

First of all, we decided to use PyAudio to handle I/O devices. Documentation for PyAudio can be found in the following URL.

http://people.csail.mit.edu/hubert/pyaudio/docs/

Our implementation has two files for execution (***client.py*** and ***server.py***). Both the client and the server can send and receive voice and text. Since we used socket for communication, the server and the client should play different role. That's the reason why we built separate files, but they have same functionality in perspective of users.

Since we can't specify host ip address, users should set the host ip address in both ***client.py*** and ***server.py***. Also, available port should be set by the users. We let the client connect to the server twice using same host ip address and port. Each connection will be used for handling text data and voice data respectively. Following is the corresponding ***server.py*** code.



```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((HOST, PORT))
sock.listen(1)
print("[PROGRAM] WAITING FOR CLIENT CONNECTION")
# The socket will accept twice so that it can handle
# text chatting and voice chatting simultaneously.

conn, addr = sock.accept()
conn_c, addr_c = sock.accept()
print ("[PROGRAM] CONNECTION COMPLETED")
```

In server.py, we implemented send_txt and receive_txt functions, and created a thread for each function. Therefore, one thread is created and designated only for sending text data and the other thread is created and designated only for receiving text data. Similarly, receive_voice and send_voice functions are implemented for receiving and sending voice data. To let the voice data go through the user's speaker, we used pyaudio library. Of course, pyaudio is used for getting voice data from the user's input device(microphone). With pyaudio, we were able to deal with I/O devices easily. Following is the code for configuration of pyaudio.

```
# Configuration for pyaudio

CHUNK = 512
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 20000

pyaudio_obj = pyaudio.PyAudio()

rcv_stream = pyaudio_obj.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                output=True,
                #output_device_index=1,
                frames_per_buffer=CHUNK)

snd_stream = pyaudio_obj.open(format=FORMAT,
                 channels=CHANNELS,
                 rate=RATE,
                 input=True,
                 #input_device_index=0,
                 frames_per_buffer=CHUNK)
```

Then rcv_stream send data to the output device which is the user's speaker. snd_stream get data from the input device which is the user's microphone. Then, the data is sent and received by the sockets. Since the voice chatting and text chatting could be done simultaneously, we created a thread which keeps sending voice data. Main program will keep receiving the voice data. When the connection is not available, the program gets terminated.

4. *Discussion*

At first, we tried to follow tutorials to learn how to use pyaudio library. Even though we follow all the steps illustrated, our program didn't get data from the microphone of the laptop. After many try-and-error steps, we could figure out that it was the problem of OS. The function that we were trying to use was working fine, but the input device and output device were not recognized by OS. Therefore, we made some code to check which devices are recognized by OS.

```
# This test is checking input/output devices of your computer

import pyaudio
po = pyaudio.PyAudio()

for index in range(po.get_device_count()):

    desc = po.get_device_info_by_index(index)

    #if desc["name"] == "record":

    print ("DEVICE: %s  INDEX:  %s  RATE:  %s "
            %(desc["name"], index, int (desc["defaultSampleRate"])))
```

The code above checks the devices recognized using pyaudio functions. By doing so, we could figure out that Ubuntu doesn't recognize the input/output devices, so we used Mac OS – Mac OS and Mac OS – Window OS to check.

5. *Conclusion*

We successfully implemented a fully functional duplex text and voice chatting program. Through this assignment we were able to gain a deeper understanding of socket programming in python, such as opening multiple sockets, and detecting dead sockets.

One key point of our implementation of non-blocking chatting was the use of threads. This is in contrast to our last assignment, where we used the select function to monitor whether a socket is ready for input or output. In the process we learned that there are various ways of implementing non-blocking communication.