Computer Network Assn#2

20170302 CSED 김대희

20170354 CSED 백승훈

**1. 프로그램 구조**

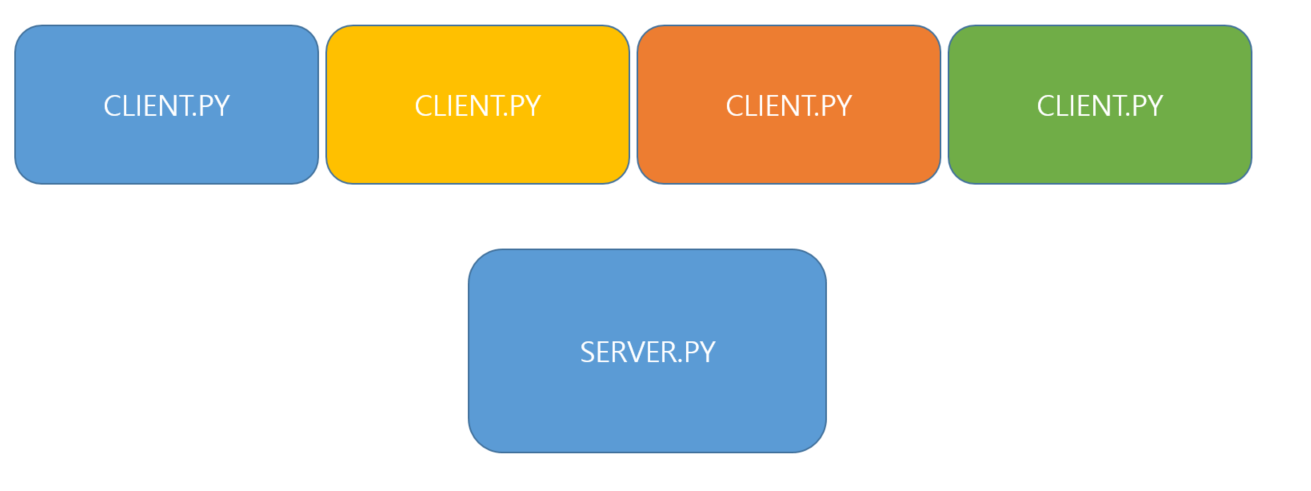


Figure 1 프로그램 구조

이번 과제에서 구현한 채팅 프로그램은 서버와 클라이언트를 각각 실행하여 서버의 IP주소로 접속하여 간단한 텍스트를 주고받을 수 있는 프로그램이다. 프로그램의 구조는 위와 같다. 1번에서는 프로그램 실행 시에 서버와 클라이언트를 실행했었는데 여기서는 오디오 서버와 텍스트서버, 그리고 오디오와 텍스트 클라이언트를 각각 실행시킨다. 나머지 구조는 기존의 프로그램과 같다.

**2.소스코드**

Text server와 client의 소스코드 부분은 지난 보고서에 자세히 적혀있기 때문에 제외하고 새로 추가된 부분의 소스코드만 보고서에 넣었습니다.

2.1audioserver.py

|  |
| --- |
| import socket  import select  class ChatServer:  def \_\_init\_\_(self):  self.CONNECTION\_LIST = []  self.chat\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  self.chat\_server\_socket.bind(("0.0.0.0", 33300))  self.chat\_server\_socket.listen(5)  self.CONNECTION\_LIST.append(self.chat\_server\_socket)   * 클래스를 초기화해주는 부분이다. 소켓을 연결하고 클래스를 초기화해주어 서버에 사용할 수 있는 형태로 저장해둔다.   print("Server Started!")  def broadcast(self, sock, data):  for current\_socket in self.CONNECTION\_LIST:  if current\_socket != self.chat\_server\_socket and current\_socket != sock:  try:  current\_socket.send(data)  except:  pass   * Try except형태로 서버안에서 소켓간 충돌을 방지한다. 소켓을 연 후 data를 send해주는 방식으로 서버에다가 데이터를 전송해주게 하는 부분이다.   def run(self):  while True:  rlist, wlist, xlist = select.select(self.CONNECTION\_LIST, [], [])   * 서버에 연결이 들어오는 클라이언트들을 관리하기 위해 리스트로 관리하기 위한 구조를 잡는다.   for current\_socket in rlist:  if current\_socket is self.chat\_server\_socket:  (new\_socket, address) = self.chat\_server\_socket.accept()  self.CONNECTION\_LIST.append(new\_socket)  print("%s connected to the server" % str(address))  else:   * 소켓에 클라이언트의 접속을 받아서 확인한 후 데이터의 전송을 해줄 수 있게 해주는 부분이다.   try:  data = current\_socket.recv(1024)  self.broadcast(current\_socket, data)  except socket.error:  print("%s left the server" % str(address))  current\_socket.close()  self.CONNECTION\_LIST.remove(current\_socket)  if \_\_name\_\_ == "\_\_main\_\_":  ChatServer().run()   * 실행기 부분이다. 메인 부분에서 run함수를 이용해서 init된 클래스 내의 메소드를 실행시키는 방법이다. |

2.2 audioclient.py

|  |
| --- |
| import socket  import pyaudio   * 오디오의 입출력을 담당할 수 있게 해주는 pyaudio를 추가한다.   import \_thread  # record  CHUNK = 1024 # 512  FORMAT = pyaudio.paInt16  CHANNELS = 1  RATE = 20000   * 입출력 오디오에 관한 기본 설정을 해준다.   host=input('Enter the address:')  port=33300   * 호스트의 IP 주소를 시작하면 적게 되어있다. Port는 33000으로 고정해두었다.   s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  s.connect((host, port))   * 호스트의 IP:33000으로 connect 시도한다.   p = pyaudio.PyAudio()  receive\_stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE, output=True, frames\_per\_buffer=CHUNK)   * 다른 사용자로부터 받을 음성 파일에 대해서 설정을 해준다.   send\_stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE, input=True, frames\_per\_buffer=CHUNK)   * 다른 사용자한테 보낼 음성 파일에 대해서 설정을 해준다.   print("Voice chat running")  def receive\_data():  while True:  try:  data = s.recv(1024)  receive\_stream.write(data)   * 스트림을 받는다.   except:  pass  def send\_data():  while True:  try:  data = send\_stream.read(CHUNK)  s.sendall(data)   * 스트림을 내보낸다.   except:  pass  \_thread.start\_new\_thread(receive\_data, ())  \_thread.start\_new\_thread(send\_data, ())   * 동시에 말을 하고 들어야 하기 때문에, Thread를 생성해서 각각 처리할 수 있게 해준다.   while True: |