Assignment #2 Text Chatting

20140427 컴퓨터공학과 김민경

20140790 컴퓨터공학과 이서림

1. Introduction

이번 과제는 소켓을 이용해 텍스트로 채팅 하는 프로그램을 만드는 것이다. 소켓 라이브러리를 이용해 서버/클라이언트 간에 text 형태의 정보를 주고받을 수 있도록 구현해야한다.

1. Design

1대 n 채팅 프로그램을 가능하게 한다.

서버는 사용자가 접속할 경우 사용자의 이름을 입력 받아 저장하고, 각 사용자가 전송한 메시지를 다른 사용자들에게 broadcast하도록 한다. 또한, 각 사용자마다 thread를 생성해 메시지 전송 및 broadcast가 독립적으로 진행될 수 있게 한다.

이번 과제에서는 터미널을 통해 채팅이 진행되도록 했으며, 추가 GUI는 구현하지 않는다.

1. Implementation

**<server.py>**

#!/usr/bin/env python3

from socket import socket, SOCK\_STREAM, AF\_INET

from threading import Thread, Lock

먼저 Socket과 thread 기능을 사용하기 위해 import한다.

def accept\_incoming\_connections():

while True:

client, client\_addr = SERVER.accept()

print("{} has connected".format(client\_addr))

lock.acquire()

addresses[client] = client\_addr

lock.release()

Thread(target=handle\_client, args=(client,)).start()

서버에 접속하는 client를 다루기 위한 함수이다. accept() 함수를 통해 서버에 접속한 client의 소켓과 주소를 받아와 저장한 수, 터미널에 접속한 client의 주소를 출력한다.

Client 주소를 addresses 변수에 저장한다.(전역 변수인 address와 clients를 수정할 때는 반드시 lock을 acquire한 후 수정하고, 수정 후에는 release한다.) 이 client를 관리하는 thread를 생성, 독립적으로 client를 관리한다. 해당 thread는 handle\_client함수를 호출하며, argument로 client 소켓을 넘긴다.

def handle\_client(client):

client.send(b"Hello! Now you are in a chat room. Tell us your name!")

name = client.recv(BUFSIZ).decode('utf-8')

welcome\_msg = 'Welcome {}! Type <quit> to exit.'\

.format(name)

client.send(welcome\_msg.encode('utf-8'))

msg = "{} has joined the chat.".format(name)

broadcast(msg)

lock.acquire()

clients[client] = name

lock.release()

while True:

msg = client.recv(BUFSIZ).decode('utf-8')

if msg != '<quit>': # client send message

broadcast(msg, name)

else: # client tried to quit

client\_addr = addresses[client]

lock.acquire()

del clients[client]

del addresses[client]

lock.release()

client.close()

broadcast("{} has left the chat.".format(name))

break

print("{} has disconnected".format(client\_addr))

client로부터 사용자 이름을 입력받아 이를 name변수에 저장한다. 해당 name을 포함한 안내 메시지를 전송한 후, 해당 name의 사용자가 채팅에 접속했다는 메시지를 모든 client에게 broadcast함수로 전달한다. Clients 변수에 사용자의 name을 저장한다.

이후, client에게 받은 메시지는 <quit>일 경우 접속을 끊고(client.close()), 해당 사용자가 채팅을 떠났다는 메시지를 broadcast하며, clients 에서도 지운다.

<quit>이 아닐 경우, 해당 메시지를 broadcast 하며, 다음 전송을 기다린다.

def broadcast(message, sender=''):

if sender:

message = '{}: {}'.format(sender, message)

for c in clients:

c.send(message.encode('utf-8'))

Broadcast 함수는 메시지와, 보통 사용자 이름을 변수로 받아 이 둘을 합친 메시지를 clients 변수 내에 저장된 모든 소켓에 전송한다.

clients = {}

addresses = {}

HOST = ''

PORT = 12345

BUFSIZ = 1024

ADDR = (HOST, PORT)

SERVER = socket(AF\_INET, SOCK\_STREAM)

SERVER.bind(ADDR)

Clients, addresses 라는 이름의 두 dictionary 변수를 선언한다.

HOST, PORT, BUFSIZ, ADDR을 선언 및 초기화 하고, 서버의 소켓을 생성하고 주소를 설정한다.

if \_\_name\_\_ == "\_\_main\_\_":

SERVER.listen(5)

print("Waiting for connection...")

ACCEPT\_THREAD = Thread(target=accept\_incoming\_connections)

ACCEPT\_THREAD.start()

ACCEPT\_THREAD.join()

SERVER.close()

accept\_incomint\_connections 함수(client의 접속을 관리)를 호출하는 thread를 생성한다.

**<client.py>**

#!/usr/bin/env python3

import os

from socket import AF\_INET, socket, SOCK\_STREAM

from threading import Thread

BUFSIZ = 1024

우선 server측과 마찬가지로 socket과 thread를 사용하기 위해 import한다.

def receive():

while True:

try:

msg = client\_socket.recv(BUFSIZ).decode('utf-8')

if msg == '<quit>':

break

print(msg)

except OSError:

break

이 함수는 client가 받은 메시지를 터미널 상에 출력해주는 함수이다.

def send():

CLEAR\_LINE = '\x1b[1A\x1b[2K'

while True:

try:

msg = input()

if msg == "<quit>":

client\_socket.send(b'<quit>')

break

if os.name != 'nt':

print(CLEAR\_LINE, end='')

client\_socket.send(msg.encode('utf-8'))

except OSError:

break

client\_socket.close()

사용자가 입력한 메시지를 받아 서버로 전송한다. <quit>을 입력받은 경우 소켓을 닫는다.

if \_\_name\_\_ == '\_\_main\_\_':

HOST = input('Enter host: ')

PORT = input('Enter port: ')

if not PORT:

PORT = 12345

else:

PORT = int(PORT)

ADDR = (HOST, PORT)

client\_socket = socket(AF\_INET, SOCK\_STREAM)

try:

client\_socket.connect(ADDR)

except ConnectionRefusedError:

print("Server is not found.")

else:

receive\_thread = Thread(target=receive)

receive\_thread.start()

send\_thread = Thread(target=send)

send\_thread.start()

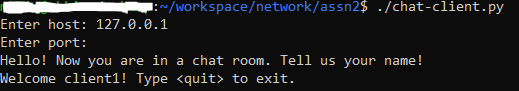
사용자가 접속하려는 서버의 host/port 값을 입력받아 ADDR 값을 정하고, client\_socket 소켓을 생성한 후 이 소켓과 ADDR값을 가진 서버를 연결한다. 서버가 존재하지 않을 경우 해당 메시지를 출력하며, 서버와 연결된 경우 receive함수를 호출하는 thread를 생성한다.

1. 프로그램 실행 방법 및 화면

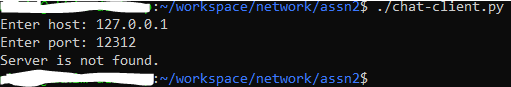
* 실행 환경 : Ubuntu 18.04.2 LTS, python 3.6 이상 / Window 10, python 3.7 환경에서 구동 확인
* 서버 실행



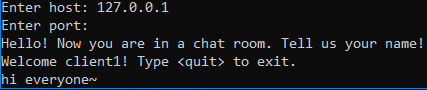
* 클라이언트 실행



* 클라이언트에서 연결을 시도한 서버가 열려 있지 않거나 host 혹은 port가 잘못되었을 경우 아래와 같은 메시지가 출력된다.



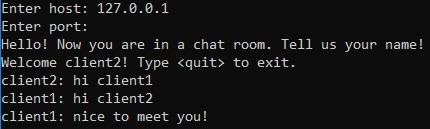
이름을 입력할 경우, 보내는 모든 메시지가 다음과 같이 이름을 포함한 형태로 출력됨을 볼 수 있다.



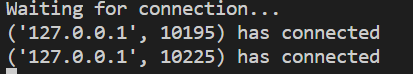
클라이언트가 접속할때마다 **서버**에는 다음과 같이 해당 클라이언트의 주소를 포함한 메세지가 출력된다.



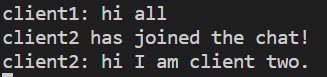
* 2명 이상의 클라이언트 접속



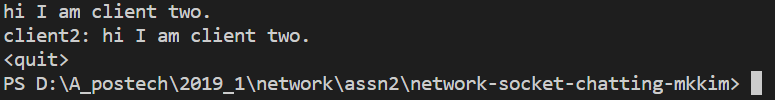
이처럼 새로운 클라이언트가 접속할 경우 다음처럼 서버에 새 클라이언트가 접속했음을 알리는 메세지가 출력된다.



기존 서버에 접속되어 있는 클라이언트의 화면에도, 다른 사용자가 접속했음을 알리는 메시지가 출력된다. 또한 Client2 가 메시지를 보내면, 서버에 접속된 기존 클라이언트 화면에도 해당 메시지가 출력된다.



<quit> 입력해 채팅 프로그램을 종료할 수 있으며, 이 경우 다른 클라이언트의 화면에 접속을 종료했다는 메시지가 출력된다.





또한 서버에는 해당 클라이언트가 disconnect했다는 메시지가 출력된다.



1. 결론 및 개선점

* Python의 socket 라이브러리를 이용해 서버/클라이언트로 이루어진 채팅 app을 구현하였다. 클라이언트의 다중 접속이 가능하도록 thread라이브러리를 사용, 잘 작동되는 것을 확인하였다.
* 더 많은 기능을 포함하기 위해선 클라이언트 부분을 GUI로 구현해야 할 필요가 있었다. 다음 과제부터는 추가 기능을 진행하기 이전에 클라이언트가 사용할 app GUI를 구현할 것이다.