P2P chatting

Computer Network

2018008904 이성진

2019-10-2

Project #0: TCP Server & Client

Project Explanation :

수업시간에 배운 TCP 서버 / 클라이언트와 조건, 분기, 반복문을 이용해 서버와 클라이언트가 동시에 존재하는 1:1 peer-to-peer 채팅 프로그램을 구현해본다.

채팅 프로그램은 Python 3.7 버전을 이용해서 만들었으며, 서버가 포트를 개방하고 연결을 대기하고 있으면 클라이언트가 연결할 수 있으며, 서버와 클라이언트가 연결되면 실시간으로 메시지를 주고받을 수 있다.

수업 시간에 조교님께서 알려주신 예시 코드를 활용하는 채팅 프로그램이지만, 반복문과 분기를 추가하여 한 번 연결되면 채팅이 끊기지 않고 연속적으로 메시지를 주고받을 수 있게 했고, 클라이언트가 특정 메시지를 입력하면 연결이 해제되도록 하는 기능을 추가했다.

Source Files:

채팅 서버와 클라이언트가 서로 독립된 PC 에서도 각각 동작할 수 있도록, 소스 코드는 포트를 개방하고 연결을 준비하는 Server.py 와 서버의 IP 와 포트를 입력하면 서버와 연결을 해 주는 Client.py 로 구분되어있다.

Server.py 의 내용은 다음과 같다.

```python
import socket

HOST = '127.0.0.1'
PORT = 50000
BUFFER = 4096
server_enabled = True

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((HOST, PORT))
sock.listen(0)
print('TCP Server listens at: %s:%s\n----------------------------\r' % (HOST, PORT))

while True:
    if not server_enabled:
        input('\nPress Any Key to Continue...')
        break

    print('\nWating for connection with Client...\n')
    client_sock, client_addr = sock.accept()

    while True:
        recv = client_sock.recv(BUFFER)
        if not recv:
            client_sock.close()
            break

        msg_recv = str(recv.decode('cp949'))
        if msg_recv == '#exit':
            print('Client was disconnected from connection')
            server_enabled = False
        else:
            print('Client %s:%s said: %s' % (client_addr[0], client_addr[1], msg_recv))
            msg = str(input('Your message to the Client> '))
            client_sock.send(msg.encode())

sock.close()
```

우선 Python 에서 기본 제공하는 socket 을 import 하여 서버용 소켓을 먼저 생성했고, Host IP 127.0.0.1 과 port 50000 으로 bind 한 후 listen(0)을 호출하여 연결에 대기한다.

클라이언트와 연결하고 메시지를 전달받는 코드는 수업 시간에 배운 코드와 거의 유사하지만, 두 가지 차이점이 존재하는데, 첫 번째는 클라이언트가 보낸 메시지를 수신한 후 클라이언트에게 답장을 할 수 있다는 점이며, 두 번째는 한 번 메시지를 주고받은 후에 반복을 통해 연결을 다시 한 번 진행하여 연속적으로 클라이언트와 대화할 수 있다는 것이다.

또한 후술할 클라이언트가 보내는 메시지 중 특정 메시지 '#exit'가 서버에 전달될 경우, server_enabled 에 False 값을 할당한다. Server_enabled 가 False 일 경우 다음 반복 루프에서 서버는 클라이언트와 연결을 하지 않고 break 로 도중에 빠져나오게 되며, 프로그램이 종료된다.

Client.py 의 내용은 다음과 같다.

```
import socket

BUFFER = 4096
connected_before = False
HOST = '127.0.0.1'
print('Host IP: '+HOST)
PORT = 50000
print('PORT: '+str(PORT))

while True:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((HOST, PORT))
    if not connected_before:
        print('Server connection enabled!\n\n\r')
        connected_before = True

    msg = str(input('Your message to server> '))
    sock.send(msg.encode())
    if msg == '#exit':
        break

    recv = sock.recv(BUFFER)
    msg_recv = str(recv.decode('cp949'))
    print('TCP Server said: %s\n' % msg_recv)

sock.close()

input('\nPress Any Key to Continue...')
```

Client 코드 역시 socket 을 import 한 다음, Host IP 와 port 에 연결을 시도하는 코드를 사용했다. 이때 반드시 서버가 미리 작동중이어야 연결이 가능하다. 서버가 연결 대기중이 아니라면 Client 프로그램은 에러 메시지를 내고 종료한다. 그런데 예시 코드와는 다르게, 무한 반복되는 반복문 안에서 소켓을 생성하고 연결을 하고 메시지를 주고 받는 과정을 진행함으로써, 한 번 연결되고 대화를 주고받는 과정 이후에도 다시 연속을 시도하게 됨으로써 서버가 연결을 받아 대화를 지속할 수 있게 되었다. 또한 서버와 미리 약속되어 있는 종료 메시지인 '#exit'를 보내면 클라이언트 역시 자동으로 종료될 수 있도록 했다.

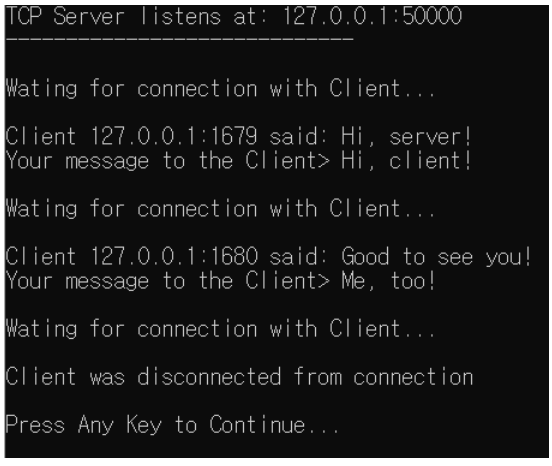
두 소스 코드 모두 메시지를 전달할 때 바이너리로 인코딩하여 전달한 후 메시지를 수신한 뒤에 cp949 로 다시 디코딩하여 출력했다. Python 의 str 형식 객체 자체를 전송할 수는 없었기 때문이다.

Instructions:

Server.py 와 Client.py 두 파일은 각각 독자적으로 작동한다. Python 3.x 버전이 설치되어 있다면, 각각의 파일을 Python 으로 실행함으로써 서버-클라이언트 p2p 채팅을 할 수 있다. 단, Server.py 를 먼저 실행시켜야 하며, Server.py 가 실행 중일 때 Client.py 를 실행하면 서버와 클라이언트 양쪽에서 연결되었다는 메시지가 뜨며 서로 메시지를 주고받을 수 있다.

How the program works:

Server.py 를 실행시키면 서버가 본인의 포트와 HOST IP 를 출력해주며, 클라이언트가 접속하기 전까지 계속해서 대기하게 된다. Listen(0)을 호출한 상태에서는 client 의 연결 시도에 대기하게 되며, 클라이언트가 IP 와 port 를 정상적으로 입력하여 연결이 성공하면 클라이언트에서 연결 성공 메시지가 출력된다. 두 프로그램 모두 while True: 이하에서 소켓 연결 및 메시지 전송 코드가 실행되도록 되어있어 클라이언트가 연결 해제를 원할 때까지 계속해서 반복해서 메시지를 주고받을 수 있게 된다. HOST IP 와 PORT 를 입력하는 기능은 없지만, 각자 독립적인 객체로 설계했기 때문에 필요시 input() 함수를 활용하면 직접 IP 와 port 를 입력 받아서 사용할 수 있도록 변경도 가능하다.



Server.py 실행 화면

Client.py 실행 화면