

# Socket Programming

## Project #1

TCP/UDP 소켓 프로그래밍을 활용한 채팅 Application 개발  
1단계 : 문자열 전송 기능 구현

2019134006 김준섭  
2019161011 박수연  
tekcos

학정번호-분반 : CSI4106.02-00

## I - 개요

Python을 이용, TCP, UDP 소켓프로그래밍을 활용한 채팅 application 개발을 목표로 프로젝트를 수행했다. 프로젝트 목표 1단계에 해당하는 문자열 전송 기능을 구현하였다.

## II 프로젝트 진행 상황

### 1. 현황 요약

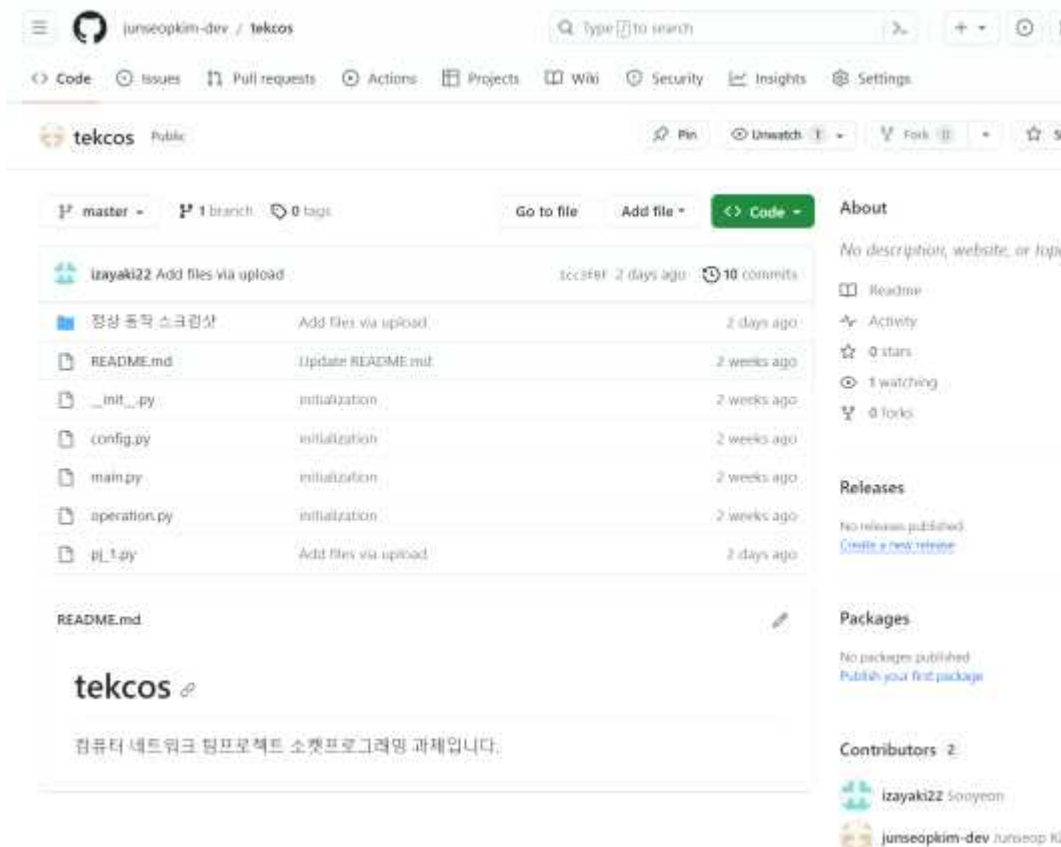
단계	핵심 구현 기능 목표	완료 여부 (마감 일자)
1	문자열 전송 기능	완료 (2023.10.30)
2	파일 전송 기능	-
3	주변 device 인식 가능	-

### 2. 역할 분담

- 김준섭 : Git 관리, 코드 테스트, 보고서 작성
- 박수연 : 소켓프로그래밍 코드 작성, 코드 테스트 및 스크린샷 캡처

### 3. 협업을 위한 Github repository 구축

- Github 주소 : <https://github.com/junseopkim-dev/tekcos>



### III 사전지식

#### 1. 네트워크 계층구조

유지관리 및 시스템 업데이트를 쉽게 하고자 네트워크 시스템은 계층구조를 이룬다.

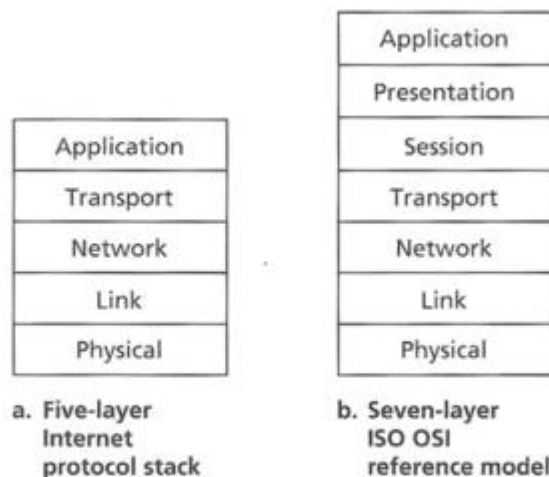
크게 2가지의 모델이 존재한다 : Internet Protocol Stack, ISO/OSI Reference Model

##### a. Internet Protocol Stack : 5개의 계층으로 구성

- (1) Application : Supporting network applications.  
ex) FTP, SMTP, HTTP
- (2) Transport : Process-Process data transfer.  
ex) TCP, UDP
- (3) Network : Routing of datagrams from source to destination.  
ex) IP, Routing Protocols
- (4) Link : Data transfer between neighboring network elements.  
ex) Ethernet, 802.11(Wifi), PPP
- (5) Physical : Move the individual bits within the frame from one node to the next.

##### b. ISO/OSI Reference Model : 7개의 계층으로 구성

- (1) Application : Supporting network applications.  
ex) FTP, SMTP, HTTP
- (2) Presentation : Allow applications to interpret meaning of data
- (3) Session : Synchronization, checkpointing, recovery of data exchange
- (4) Transport : Process-Process data transfer.  
ex) TCP, UDP
- (5) Network : Routing of datagrams from source to destination.  
ex) IP, Routing Protocols
- (6) Link : Data transfer between neighboring network elements.  
ex) Ethernet, 802.11(Wifi), PPP
- (7) Physical : Move the individual bits within the frame from one node to the next.



**Figure 1.23** ♦ The Internet protocol stack (a) and OSI reference model (b)

## 2. Transport Layer Protocol : TCP & UDP

### a. TCP (Transmission Control Protocol)

#### (1) 특징 :

- (가) reliable (rdt 1.0, 2.0, 2.1, 2.2, 3.0 등)
- (나) in-order delivery
- (다) 연결 지향적 프로토콜 (3-way handshake)

#### (2) 장단점

- (가) 장점 : rdt 3.0을 통해 높은 신뢰성을 보장한다.
- (나) 단점 : UDP보다 속도가 느리다.

### b. UDP (User Datagram Protocol)

#### (1) 특징 :

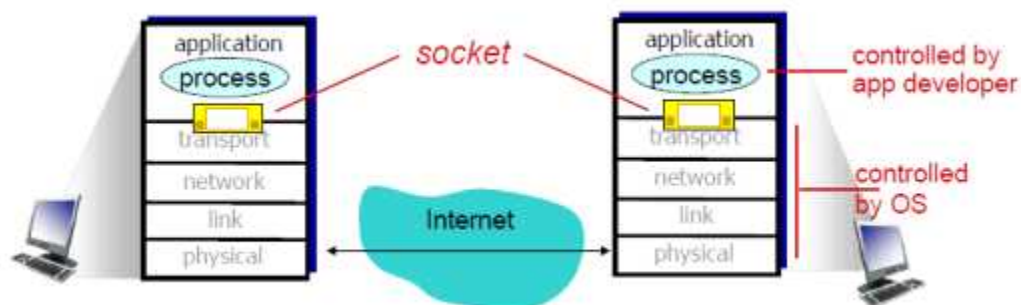
- (가) unreliable
- (나) unordered delivery
- (다) 비연결형 프로토콜

#### (2) 장단점

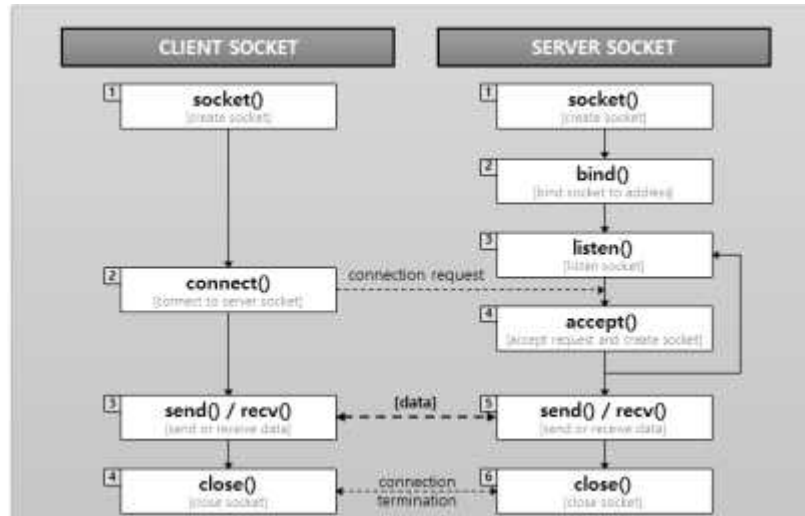
- (가) 장점 : 빠른 속도로 전송 가능하다.
- (나) 단점 : 신뢰성을 보장하지 못한다.

## 3. Socket

- a. A process sends messages into, and receives messages from, the network through a software interface called a **socket**.
- b. 프로세스는 집, 소켓은 문에 비유할 수 있다.

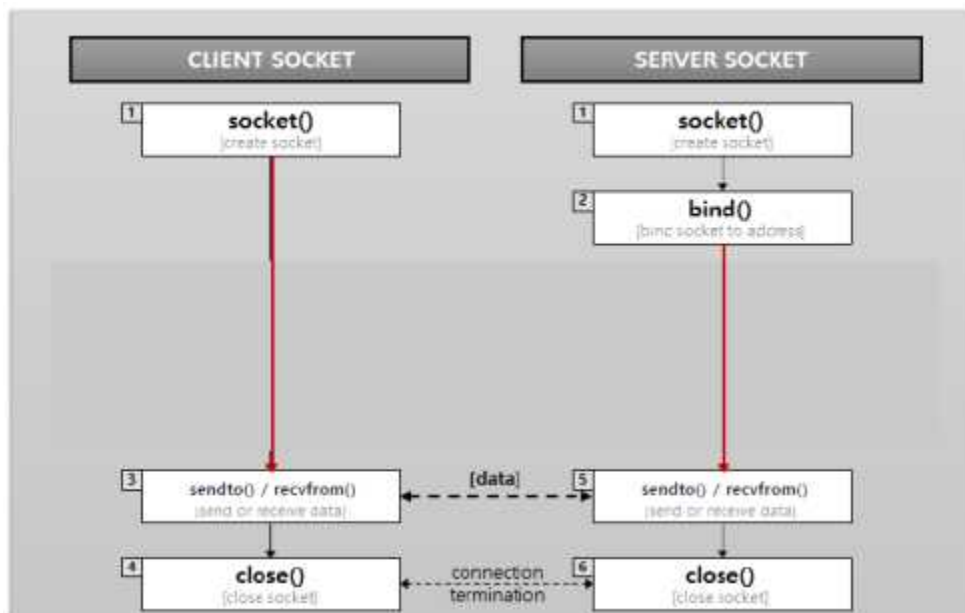


## c. Socket Programming with TCP



- (1) TCP는 연결 지향성 프로토콜이므로, 데이터 패킷을 주고받기 전에 서로를 연결하는 절차가 필요하다. (3-way handshake)
- (2) 서로 데이터를 주고받기 전 서버는 `listen()`을 통해 클라이언트로부터의 연결요청이 오기를 대기한다.
- (3) 서버가 listening하는 사이, 클라이언트는 `connect()`를 통해 서버에게 연결요청을 보낸다.
- (4) 클라이언트로부터 연결요청이 온 경우, 서버는 `accept()`를 통해 연결을 허가하는 절차를 거친다.
- (5) 서버-클라이언트 간 연결이 수립된 이후, `send()/recv()`를 통해 서로 데이터를 주고받을 수 있게 된다.

## d. Socket Programming with UDP



- (1) UDP는 TCP와는 달리 비연결형 프로토콜로, 이 때문에 TCP와는 달리 클라이언트의 `connect()`, 서버의 `listen()`, `accept()`가 존재하지 않는다.
- (2) 서버-클라이언트 간 `sendto()`, `recvfrom()`을 통해 data를 주고받는다.

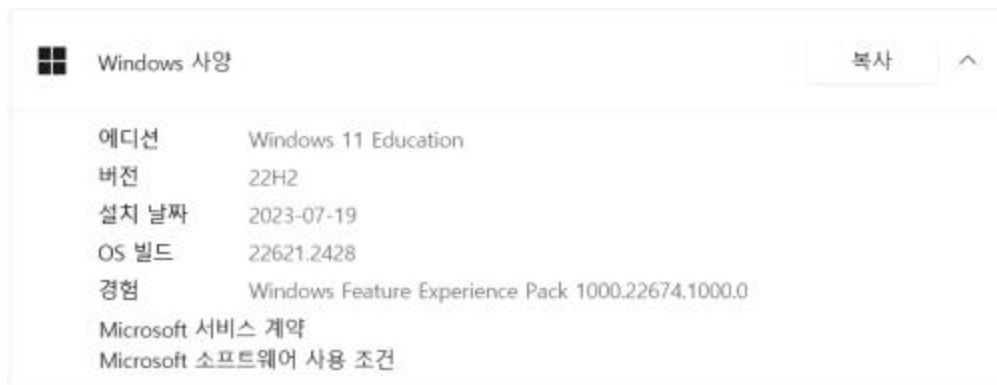
## IV 구현

1. 구현 목표 : 파이썬을 이용해 TCP, UDP 소켓 프로그래밍을 이용해 문자열 전송 기능이 구현된 채팅 어플리케이션 개발

2. 구현 환경

a. OS 정보

- (1) 에디션 : Windows 11 Education
- (2) 버전 : 22H2
- (3) 설치 날짜 : 2023-07-19
- (4) OS 빌드 : 22621.2428
- (5) 경험 : Windows Feature Experience Pack 1000.22674.1000.0



b. Python 버전 : 3.11.5



## 3. 구현 코드

a. 전체 코드 : pj\_1.py외에는 코드의 변경이 없으므로, pj\_1.py의 코드만 기재하였다.

```

from time import sleep
from typing import Tuple
from config import *
from threading import Thread
import socket

class NetworkSocket:
    def __init__(self)->None:
        self.tcp_socket = None
        self.udp_socket = None
        self.target_tcp_addr = None
        self.target_udp_addr = None

    @staticmethod
    def tcp_server_socket(host:str,port:int)->socket.socket:
        # TCP server socket 생성

        sock =socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        sock.bind((host,port))
        sock.listen(10)

        return sock

    @staticmethod
    def tcp_server_connect(server_socket:socket.socket)->Tuple[socket.socket,any]:
        # Server_socket을 통해 client와의 connection 생성
        # 생성된 connection socket(conn)과 client의 address(tcp_client_addr) 반환

        conn_sock,addr =server_socket.accept()

        return conn_sock,addr

    @staticmethod
    def tcp_client_socket(host:str,port:int)->socket.socket:
        # TCP client socket 생성
        # Server에 connection을 요청하고, server와 client 간 tcp socket(tcp_client_socket)
        반환

        sock =socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        sock.connect((host,port))

        return sock

```

```

@staticmethod
def udp_server_socket(host:str,port:int)->socket.socket:
    # UDP server socket 생성

    sock =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    sock.bind((host,port))

    return sock

@staticmethod
def udp_server_connect(udp_server_socket:socket.socket):
    # udp_client_socket 함수가 전송한 packet으로부터 client의 udp
address(udp_client_addr) 반환

    data,addr =udp_server_socket.recvfrom(1024)
    return addr

@staticmethod
def udp_client_socket(host:str,port:int)->socket.socket:
    # UDP client socket 생성
    # UDP 통신으로 server에 packet을 전송하고 udp client socket(udp_client_socket) 반
환

    sock =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    sock.sendto(b'HELO',(host,port))

    return sock

def tcp_send(self,data:bytes)->None:
    # TCP socket(tcp_socket)을 통해 입력 받은 data 전송

    self.tcp_socket.send(data)

    pass

def udp_send(self,data:bytes)->None:
    # UDP socket(udp_socket)를 통해 상대방의 udp 주소(target_udp_addr)로 입력받은
data 전송

    self.udp_socket.sendto(data,self.target_udp_addr)

    pass

def tcp_rcv(self)->bytes:
    # TCP socket(tcp_socket)으로 들어오는 packet의 data 반환

    data =self.tcp_socket.recv(1024)

    return data

def udp_rcv(self)->bytes:
    # UDP socket(udp_socket)으로 들어오는 packet의 data 반환

    data ,addr =self.udp_socket.recvfrom(1024)

    return data

```



```

def close(self)->None:
    try:
        self.tcp_socket.close()
    except socket.error as msg:
        print(f"Unexpected {msg}, {type(msg)}")

    try:
        self.udp_socket.close()

    except socket.error as msg:
        print(f"Unexpected {msg}, {type(msg)}")

def server_open_func(self,host:str="",tcp_port:int =DEFAULT_TCP_PORT,
                    udp_port:int =DEFAULT_UDP_PORT)->int:
    try:
        server_socket =self.tcp_server_socket(host,tcp_port)
        self.tcp_socket,self.target_tcp_addr =self.tcp_server_connect(server_socket)
        self.udp_socket =self.udp_server_socket(host,udp_port)
        self.tcp_send("ack".encode(ENCODING))
        self.target_udp_addr =self.udp_server_connect(self.udp_socket)
        return 0

    except socket.error as msg:
        print(f"Unexpected {msg}, {type(msg)}")
        return -1

def client_connect_func(self,host:str="",tcp_port:int =DEFAULT_TCP_PORT,
                    udp_port:int =DEFAULT_UDP_PORT)->int:
    try:
        self.target_tcp_addr =(host,tcp_port)
        self.target_udp_addr =(host,udp_port)
        self.tcp_socket =self.tcp_client_socket(host,tcp_port)
        _=self.tcp_recv() # recv ack
        self.udp_socket =self.udp_client_socket(*self.target_udp_addr)
        return 0

    except socket.error as msg:
        print(f"Unexpected {msg}, {type(msg)}")
        return -1

```

## b. 세부 설명

## (1) 함수 tcp\_server\_socket

함수 tcp_server_socket
<pre>@staticmethod def tcp_server_socket(host:str,port:int)-&gt;socket.socket:     sock =socket.socket(socket.AF_INET,socket.SOCK_STREAM)     sock.bind((host,port))     sock.listen(10)     return sock</pre>
설명
<p>요약 : TCP 소켓을 생성하는 함수 tcp_server_socket 정의.</p> <p>sock =socket.socket(socket.AF_INET,socket.SOCK_STREAM)  AF_INET : 현재 네트워크상 대부분의 IP주소는 IPv4이므로, AF_INET으로 설정  SOCK_STREAM : TCP를 사용할 것을 의미</p> <p>sock.bind((host,port))  소켓을 주어진 'host'와 'port'에 바인딩</p> <p>sock.listen(10)  동시에 최대 10개의 연결 요청을 대기할 수 있도록 설정,</p> <p>return sock  생성하고 설정한 'sock' 객체 반환</p>

## (2) 함수 tcp\_server\_connect

함수 tcp_server_connect
<pre>@staticmethod def tcp_server_connect(server_socket:socket.socket)-&gt;Tuple[socket.socket,any]:     conn_sock,addr =server_socket.accept()     return conn_sock,addr</pre>
설명
<p>요약 : TCP 서버 소켓을 사용하여 클라이언트와의 연결을 수립하고, 연결된 소켓과 클라이언트의 주소를 반환하는 함수 tcp_server_connect 정의</p> <p>conn_sock,addr =server_socket.accept()  클라이언트의 연결 요청을 대기하다가 연결 요청이 들어오면 그 연결을 수락</p> <p>return conn_sock,addr  연결된 소켓 'conn_sock'과 클라이언트 주소 'addr' 반환</p>

## (3) 함수 tcp\_client\_socket

함수 tcp_client_socket
<pre>@staticmethod def tcp_client_socket(host:str,port:int)-&gt;socket.socket:     sock =socket.socket(socket.AF_INET,socket.SOCK_STREAM)     sock.connect((host,port))     return sock</pre>
설명
<p>요약 : TCP 클라이언트 소켓을 생성하고, 주어진 호스트와 포트에 연결하는 함수 tcp_client_socket 함수 정의</p> <p>sock =socket.socket(socket.AF_INET,socket.SOCK_STREAM)  AF_INET : 현재 네트워크상 대부분의 IP주소는 IPv4이므로, AF_INET으로 설정  SOCK_STREAM : TCP를 사용할 것을 의미</p> <p>sock.connect((host,port))  주어진 'host'와 'port'에 연결 시도. 해당 주소 서버와 TCP 연결 수립</p> <p>return sock  연결된 'sock' 객체 반환</p>

## (4) 함수 udp\_server\_socket

함수 udp_server_socket
<pre>@staticmethod def udp_server_socket(host:str,port:int)-&gt;socket.socket:     sock =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)     sock.bind((host,port))     return sock</pre>
설명
<p>요약 : UDP 소켓을 생성하는 함수 <code>udp_server_socket</code> 정의.</p> <p>sock =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)  AF_INET : 현재 네트워크상 대부분의 IP주소는 IPv4이므로, AF_INET으로 설정  SOCK_DGRAM : UDP를 사용할 것을 의미</p> <p>sock.bind((host,port))  소켓을 주어진 'host'와 'port'에 바인딩</p> <p>return sock  생성하고 설정한 'sock' 객체 반환</p> <p>*TCP와는 달리 listen이 없음을 유의 (비연결형 프로토콜이기 때문)</p>

## (5) 함수 udp\_server\_connect

함수 udp_server_connect
<pre>@staticmethod def udp_server_connect(udp_server_socket:socket.socket):     data,addr =udp_server_socket.recvfrom(1024)     return addr</pre>
설명
<p>요약 : UDP 서버 소켓을 사용하여 클라이언트로부터 데이터를 수신하고, 해당 데이터를 보낸 클라이언트의 주소를 반환하는 함수 <code>udp_server_connect</code> 정의</p> <p>data,addr =udp_server_socket.recvfrom(1024)  클라이언트로부터 최대 1024 바이트의 데이터를 수신  수신된 데이터는 'data'에 저장  데이터를 보낸 클라이언트의 주소는 'addr'에 저장</p> <p>return addr  데이터를 보낸 클라이언트의 주소 'addr' 반환</p>

## (6) 함수 udp\_client\_socket

함수 udp_client_socket
<pre>@staticmethod def udp_client_socket(host:str,port:int)-&gt;socket.socket:     sock =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)     sock.sendto(b'HELO',(host,port))     return sock</pre>
설명
<p>요약 : UDP 클라이언트 소켓을 생성하고, 서버에 데이터 패킷을 전송하는 함수 <code>udp_client_socket</code> 정의</p> <p>sock =socket.socket(socket.AF_INET,socket.SOCK_DGRAM)  AF_INET : 현재 네트워크상 대부분의 IP주소는 IPv4이므로, AF_INET으로 설정  SOCK_DGRAM : UDP를 사용할 것을 의미</p> <p>sock.sendto(b'HELO',(host,port))  "HELO"라는 바이트 문자열 데이터를 주어진 host와 port에 있는 서버에 전송</p> <p>return sock  생성된 'sock' 객체 반환</p>

## (7) 함수 tcp\_send

함수 tcp_send
<pre>def tcp_send(self,data:bytes)-&gt;None:     self.tcp_socket.send(data)     pass</pre>
설명
<p>요약 : TCP 소켓을 사용하여 데이터를 전송하는 함수 <code>tcp_send</code> 정의</p> <pre>self.tcp_socket.send(data)</pre> <p>TCP 소켓 'self.tcp_socket'을 통해 입력받은 'data' 전송</p> <p><b>pass</b></p> <p>함수 정의가 끝났음을 나타내기 위해 사용. 의미 없음.</p>

## (8) 함수 udp\_send

함수 udp_send
<pre>def udp_send(self,data:bytes)-&gt;None:     self.udp_socket.sendto(data,self.target_udp_addr)     pass</pre>
설명
<p>요약 : UDP 소켓을 사용하여 특정 주소로 데이터를 전송하는 함수 <code>udp_send</code> 정의</p> <pre>self.udp_socket.sendto(data,self.target_udp_addr)</pre> <p>UDP 소켓 'self.udp_socket'을 통해 'self.target_udp_addr' 주소로 입력받은 'data' 전송</p> <p><b>pass</b></p> <p>함수 정의가 끝났음을 나타내기 위해 사용. 의미 없음.</p>

## (9) 함수 tcp\_recv

함수 tcp_recv
<pre>def tcp_recv(self)-&gt;bytes:     data =self.tcp_socket.recv(1024)     return data</pre>
설명
<p>요약 : TCP 소켓을 사용하여 들어오는 데이터 패킷을 수신하는 함수 <code>tcp_recv</code> 정의</p> <pre>data =self.tcp_socket.recv(1024)</pre> <p>TCP 소켓 'self.tcp_socket'으로부터 최대 1024 바이트의 데이터 수신 수신된 데이터는 'data'에 저장</p> <p><b>return data</b></p> <p>수신된 'data' 반환</p>

## (10) 함수 udp\_recv

함수 udp_recv
<pre>def udp_recv(self)-&gt;bytes:     data ,addr =self.udp_socket.recvfrom(1024)     return data</pre>
설명
<p>요약 : UDP 소켓을 사용하여 들어오는 데이터 패킷을 수신하는 함수 <code>udp_recv</code> 정의</p> <pre>data ,addr =self.udp_socket.recvfrom(1024)</pre> <p>UDP 소켓 'self.udp_socket'으로부터 최대 1024 바이트의 데이터 수신 수신된 데이터는 'data'에 저장. 데이터를 보낸 주소는 'addr'에 저장.</p> <p><b>return data</b></p> <p>수신된 'data' 반환</p>

4. 동작 화면 : Server, Client 양쪽에서 서로 10회 이상 다른 메시지를 주고받을 수 있는지 확인했다.

Server

Client

## V 고찰

1. 테스트를 수행한 결과 TCP, UDP, 둘 다 송수신이 설계한대로 잘 작동함을 확인할 수 있었다.
2. TCP는 UDP와 달리 신뢰성이 높지만 속도가 느리다는 특징을 가지며, UDP는 TCP와 달리 속도가 빠르지만 신뢰성이 낮다는 특징을 갖는다.  
서버와 클라이언트간 데이터를 송신하고 수신하기까지 걸리는 시간을 측정해보거나, 두 프로토콜 간 오차 발생 빈도를 기록하는 기능을 추가해본다면  
TCP와 UDP의 차이를 실제 관찰되는 값으로서 비교해볼 수 있을 것으로 생각된다.
3. 앞서 언급한 TCP와 UDP의 차이에서 유추할 수 있듯, 향후 구현될 파일 전송 기능은 실시간으로 전송되어야 할 필요가 아니라면 TCP를 통해 이루어지는 것이 적합할 것으로 판단된다.

## VI 참고문헌

1. Kurose, J., & Ross, K. (2018, October 23). Computer Networking: A Top-Down Approach, Global Edition. Pearson Higher Ed.
2. 컴퓨터네트워크 강의자료 Chapter 1. Introduction
3. 컴퓨터네트워크 강의자료 Chapter 2. Application Layer (2.1-2.3)
4. 컴퓨터네트워크 강의자료 Chapter 3. Transport Layer (3.1-3.4)
5. 컴퓨터네트워크 강의자료 Chapter 3. Transport Layer (3.5-3.7)
6. 컴퓨터네트워크 강의자료 231011\_Socket\_Programming\_Project1