<컴퓨터 네트워크 Project #2>

과 목	컴퓨터네트워크
담당 교수	한승재 교수님
제출 일자	2020.10.31
전공	산업공학
학번	2016147008
이름	조성현

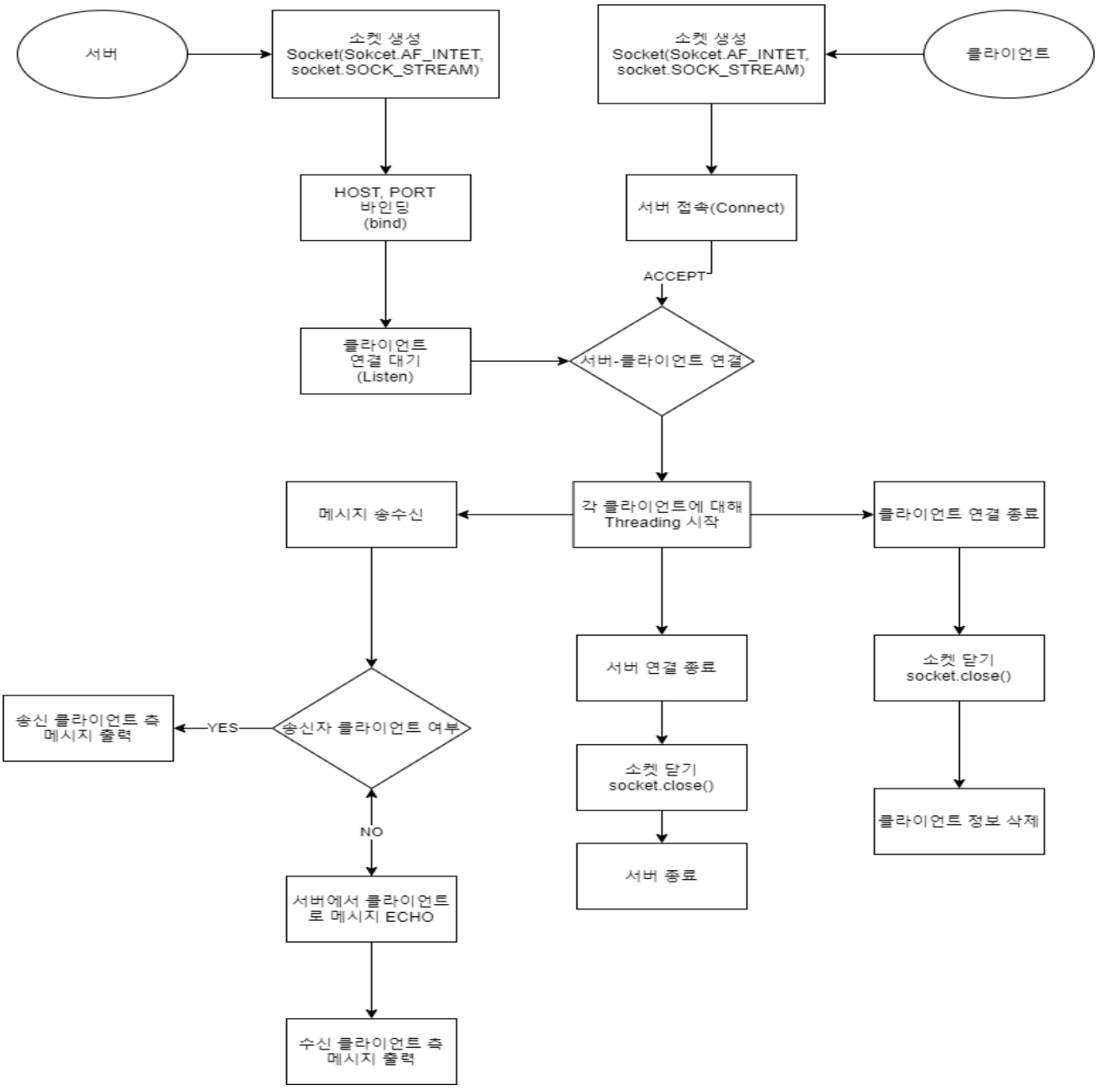
1. Introduction/Reference

- 본 프로젝트에서는 Python(Version: 3.8) 언어를 사용하였다.

- 본 프로젝트의 실행환경은 Ubuntu(Version: 18.04)를 기반으로 한 Linux OS이다.

- 프로젝트에서는 Socket, threading, signal, sys 표준 라이브러리를 사용하였다.

2. Flowchart



3. Explanations Of the 8 functions

1) socket(socket.AF_INET, socket.SOCK_STREAM) : 이 함수를 이용해서 소켓 객체를 생성할 수 있다. 서버든 클라이언트 등 동일하게 소켓을 이용한 네트워킹을 하기 위해서는 소켓을 먼저 생성할 필요가 있다. 이 함수는 두 가지 인자를 받는데, 하나는 패밀리이고 다른 하나는 타입이다. Socket.AF_INET은 IPv4 주소 체계를 나타내기 위함이며, socket_SOCK_STREAM의 경우 TCP를 이용한 소켓을 생성한다는 것을 의미한다.

2) setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) : 소켓은 소켓옵션을 조작해서 세부사항을 조절할 수 있는데, 이를 위해 setsockopt() 함수를 사용할 수 있다. 소켓을 이용한 서버프로그램을 운용하다 보면 강제 종료되거나 비정상 종료되는 경우가 발생한다. 테스트를 목적으로 할 경우에는 특히 강제종료 시켜야 하는 경우가 자주 발생하는데, 강제종료 시키고 프로그램을 다시 실행시킬 경우 기존 프로그램이 종료되었지만, 비정상 종료된 상태로 아직 커널이 bind정보를 유지하고 있다. 이는 흔히 TCP 연결에서 timeout과 연관될 수도 있고, 보통 1분 후에 사라지긴 하지만, 그 시간동안 연결 지연을 방지하기 위해 삽입한다.

3). bind(("0.0.0.0", 7777)): 일반적으로 서버 소켓은 고정된 포트 번호를 사용한다. 그리고 그 포트 번호로 클라이언트의 연결 요청을 받아들이게 된다. 그래서 운영체제가 특정 포트 번호를 서버 소켓이 사용하도록 만들기 위해 소켓과 포트 번호를 결합(bind)해야 하는데, 이 때 사용하는 API가 바로 bind()이다. 현재 “0.0.0.0”는 로컬 호스트의 IP 주소가 0.0.0.0임을 의미하며, 7777은 포트 번호를 의미한다.

4) •listen(5): bind()를 마치게 되면, 클라이언트 측의 연결을 기다리는 상태로 만들기 위한 함수이다. listen()함수는 연결을 요청받을 클라이언트의 수를 Parameter로 입력받는다. 이 경우 5명의 클라이언트에 대한 연결을 허용하겠다는 의미이다.

5) connect((host, port)): 클라이언트는 서버에 연결하기 위해 sock.connect()를 사용하며 이 때 사용하는 인자는 bind()와 동일하다. 이 메소드는 연결이 수립되면 종료되고, 리턴하는 값은 존재하지 않는다. 클라이언트는 서버에 연결되어야 하므로, 로컬 호스트 주소를 sys를 통해 입력받고 이전에 서버 측에서 생성한 port번호를 동일하게 입력하도록 한다.

6) accept(): 이 함수는 연결지향 소켓에 사용된다. 이것은 아직 처리되지 않은 연결들이 대기하고 있는 큐에서 제일 처음 연결된 연결을 가져와서 새로운 연결된 소켓을 만든다. 그리고 소켓을 가르키는 파일 지정자를 할당하고 이것을 리턴한다. 이 때는 소켓의 정보와 IP주소, Port 번호를 리턴한다.

7) close(): 더 이상 데이터 송수신이 필요 없게 되면, 소켓을 닫기 위해 사용한다.

8) Thread(): Thread를 사용하는 이유는 우리가 서버와 클라이언트에서 메시지를 보낼 때 순서에 상관없이 보내기 위함이다. Thread를 이용하면 하나의 프로그램에서 한번에 하나의 일만 처리하는 것이 아니라, 동시에 여러가지 일을 처리할 수 있다. Thread() 함수는 여러가지 인자를 받게 되는데, TCP 소켓 통신에서 사용하는 인자는 target과 args이다. Target은 쓰레딩을 통해 실행할 함수가 입력되는 부분이며, 그 함수에 필요한 파라미터를 args에 전달하도록 한다. 그런데 이 때, args에서 인자가 하나일 경우 ','을 넣어서 튜플로 인식하도록 해야 한다. 이렇게 생성된 Thread는 start()함수를 통해 작동을 시작해야 한다. 본 프로젝트에서는 while True안에 Thread를 삽입함으로써 지속적인 쓰레딩이 일어나도록 하였고, 프로세스가 종료될 때 쓰레딩이 같이 종료되도록 구현하였다.

4. difference between using multi-thread and the function select()

Socket을 이용한 통신에는 blocking이라는 문제가 있다. blocking이란 어떤 일이 일어나기를 기다리는 상태인데, 서버와 클라이언트가 서로 데이터를 넘겨주는 상황을 기다리는 상태이다. 특히 TCP 연결에서는 timeout을 통해 blocking 상태를 벗어나도록 한다. Select()함수의 경우 rlist, wlist, xlist, timeout을 파라미터로 받는다. Rlist는 읽을 준비가 될 때까지 기다리며, wlist는 쓰기 준비가 될 때까지 기다린다. xlist에는 예외 조건을 기다린다. Timeout은 선택적인 입력 파라미터로 시간제한을 지정할 수 있다. 즉, 소켓 set에 저장된 소켓들에 변화가 생길 때까지 기다리고 있다가 소켓이 어떤 동작을 하게 되면 동작한 소켓을 제외한 나머지 소켓을 모두 제거하고 해당되는 소켓에 대해 작업을 진행하게 된다. 본 프로젝트에서 Select()를 사용하게 된다면, 다중 클라이언트에 대하여 서버가 작업을 하고 있기 때문에, 자신의 메시지에 대한 송수신을 하기 위해서는 자신의 순서가 될 때까지 무조건 기다려야 하는 문제점이 있다. 물론 timeout을 통해 그 시간을 줄일수는 있지만, threading 방식에 비해 효율적이지 못하다. 반면 Thread는 하나의 프로그램에서 하나의 소켓에 대한 일만 처리하는 것이 아니라, 동시에 여러가지 일을 처리하게 된다.

Thread는 하나의 프로세스를 다수의 실행 단위로 구분하여 자원을 공유하고 자원의 생성과 관리의 중복성을 최소화하여 수행 능력을 향상시킨다. 하나의 프로그램에서 동시에 여러 개의 일을 수행할 수 있도록 하여 Select()보다 작업의 효율성을 높이게 해준다. 또한 특정한 Thread에서 작업을 하고 있는 동안 다른 Thread에서 작업이 가능하여 사용자가 받는 응답이 증가한다. 또한 프로세스 내 자원들과 메모리를 공유하고 이기 때문에 자원의 소모가 줄어든다. 그러나, 공유하는 자원에 대한 경계가 분명하지 않은 상황에서 스레드는 다른 스레드에서 사용 중인 변수나, 변경된 값에 잘못 접근하게 되는 문제가 있다. 또한 Thread에서 lock을 적절히 acquire, release를 못하게 되면 특정 스레드에서 병목현상을 겪게 될 수도 있다. Select()는 작업이 동시에 진행되지 못하는 단점은 있지만 Thread가 가지고 있는 자원의 공유 문제를 해소할 수 있다.

3.1 Explanation – 서버 측

```
# 매개변수로 host, port 정보를 입력받습니다.
host = sys.argv[1]
port = int(sys.argv[2])
print("Chat Server started on port " + str(port)+".")

# 소켓 객체 생성
# 주소 체계(address family)로 IPv4, 소켓 타입: TCP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# WinError 10048 해결하기 위한 코드
# Address already in use Error solution
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)


# bind 함수는 소켓을 특정 네트워크 인터페이스와 포트 번호에 연결하는데 사용
# HOST는 hostname, ip address, 빈 문자열 ""이 될 수 있음
# 빈 문자열이면 모든 네트워크 인터페이스로부터의 접속을 허용
# port는 1-65536 사이의 숫자를 사용 가능
server_socket.bind((host, port))

# accept 함수에서 대기하다가 클라이언트가 접속하면
# 클라이언트의 소켓과 클라이언트의 주소를 반환해서 저장
server_socket.listen()
```

sys 모듈을 이용하여 HOST, PORT에 대한 정보를 입력 받는다. 그리고 채팅서버가 시작했다는 메시지를 출력한다. 이후, 서버 측의 소켓을 생성하였다. Socket.AF_INET은 IPv4 주소 체계를 나타내기 위함이며socket_SOCK_STREAM의 경우 TCP를 이용한 소켓을 생성한다는 것을 의미한다.

다음으로 server_socket.bind((host,port))를 이용하여 생성한 소켓을 bind시켜준다. 생성된 소켓의 번호와 실제 IPv4를 연결하는 과정이다. bind 함수에서는 소켓과 AF_INET을 연결하는 과정으로써 두 가지 Parameter(IP, PORT)를 받게 된다.

bind()를 마치게 되면, 클라이언트 측의 연결을 기다리는 상태로 변한다. 이는 listen()함수를 통해 알 수 있으며, listen()함수 안에 Parameter를 입력하지 않음으로써 연결을 요청하는 클라이언트의 수를 제한하지 않도록 하였다. 이 과정까지 진행되면 서버 소켓은 클라이언트 측의 접속을 대기하는 상태가 된다.

```
# 메시지를 전송합니다
# 무한루프를 돌면서
while True:
    interrupted = False
    signal.signal(signal.SIGINT, signal_handler)

    # 서버가 클라이언트의 접속을 허용하도록 함
    client_socket, addr = server_socket.accept()

    userNumber += 1
    userid = id(client_socket)
    adduser(userid, client_socket)

    t = threading.Thread(target=threading_func, args=(client_socket, addr))
    t.daemon = True
    t.start()
```

위 코드에서 interrupted와 signal 부분은 이 후에 Ctrl+C를 입력하여 종료하기 위해 만들어진 함수이다. 이는 마지막 부분에서 사용하기 위해 생성해두었다.

앞서, 서버 소켓은 클라이언트의 접속을 대기하는 상태로 존재했다. 이 때, accept()함수를 이용해 클라이언트와 연결하게 된다. 클라이언트 측 소켓이 연결을 시도하면 accept()가 실행되고, return 값으로 클라이언트 측의 소켓과 주소를 전달하게 된다.

클라이언트가 연결되게 되면 현재 서버에 접속한 userNumber를 증가시키고, userId는 고유한 값을 할당하기 위해 Client_socket의 id를 입력받아 adduser()함수에 전달하도록 하였다. adduser()함수는 밑에서 다시 설명하도록 한다.

이 후에, 서버에 접속한 다중 클라이언트들과 메시지를 주고받기 위해 멀티쓰레딩(Multi-threading)을 시작한다. 대부분의 서버는 하나의 서버에 하나의 클라이언트만 접속하는 것이 아니라 수 많은 클라이언트가 서버에 접속한다. 서버 입장에서 다중 클라이언트가 접속한 상황에서 여러 클라이언트들이 동시에 여러 작업을 하는 경우 작업을 하나씩 순서대로 처리하는 것이 아니라, 하나의 클라이언트가 접속하면 Thread를 추가로 하나 생성해서 각 클라이언트들이 수행하는 작업을 각각의 Thread 안에서 처리하게 해야 한다.

```python
def threading_func(client_socket, addr):
    login_notice(client_socket, addr)
    while True:
        try:
            message = client_socket.recv(1024).decode()
            if not message:
                break
            print('[{}:{}] {}'.format(addr[0], addr[1], message))

            for clients in users.values():
                sender_ip = addr[0]
                sender_port = addr[1]
                if clients != client_socket:
                    # clients.send(message.encode())
                    clients.sendall(
                        ("["+str(sender_ip)+":"+str(sender_port)+"] "+message).encode())
        except ConnectionResetError:
            break

    userleaves(client_socket, addr)
    client_socket.close()
```

서버에 접속한 각 클라이언트에 대해 위와 같은 Threading Function이 작동하게 된다. 서버에 클라이언트가 접속하면, login_notice()함수를 호출하여 클라이언트의 정보를 담고, 서버와 클라이언트 측 채팅창에 접속에 대한 알림을 띄우도록 한다.

Thread 상태에서는 클라이언트 측의 메시지를 수신하고 있다. 클라이언트 측에서 메시지가 송신되면 서버는 그 메시지를 수신하여 decode한 뒤, 서버 상에 출력하게 된다. 한편, 서버와 클라이언트가 연결될 때마다 추가되는 유저 정보를 확인하여 메시지를 송신한 클라이언트를 제외한 다른 모든 클라이언트들에게는 서버가 받은 메시지를 다시 재전송(Echo)하도록 한다. 이 때, 메시지의 내용은 다시 Encode()를 하여 전달하도록 하며, 메시지를 전달할 때 방금 메시지를 송신한 클라이언트의 IP주소와 Port번호를 같이 전송하도록 하여 다른 클라이언트에서 이 메시지를 확인했을 때, 그 출처를 파악하고 출력할 수 있도록 구현하였다. 한편, 각 클라이언트에 대한 Threading은 클라이언트가 접속을 종료하면 ConnectionResetError 예외가 발생하여 종료되게 되고, userleaves()라는 함수를 호출하여 유저 정보에 담긴 클라이언트의 정보를 삭제한 후, 연결되었던 클라이언트 측의 소켓을 종료하도록 한다.

```python
userNumber = 0
users = {}
lock = threading.Lock()

# 유저가 들어오면 User 목록에 추가합니다.
def adduser(userid, clients):
    lock.acquire()
    users[userid] = clients
    lock.release()

# 유저가 나가면 userNumber를 줄이고 User목록에서 제거합니다.
def removeuser(userid):
    lock.acquire()
    global userNumber
    userNumber -= 1
    del users[userid]
    lock.release()
```

adduser와 removeuser() 함수는 각각 이전에 설명한대로, 클라이언트와 서버가 연결될 때 클라이언트의 정보와 현재 연결된 클라이언트 수의 정보 등을 갱신하도록 한다. 유저 정보는 Dictionary 객체로 저장하는데, Key값을 클라이언트가 연결될 때 생성한 id값으로 설정하고, value값으로 client를 설정하도록 한다. 따라서 유저가 들어오게 되면 Dictionary에 정보를 추가하고, 유저가 나가게 되면 userNumber를 감소시킨 후에, 저장된 유저 정보에서 그 값을 제거하도록 한다. 이 때 lock.acquire(), lcok.release()를 사용하여, 동시에 접속하거나 로그아웃하는 클라이언트에 대한 문제를 해결하도록 하였다. Threading.lock()을 사용하여 자원을 동시에 사용하지 않고 순차적으로 처리하도록 하였다.

```python
# 유저가 로그인하면 알림을 띄웁니다.
def login_notice(client_socket, addr):
    if userNumber == 1:
        print('> New user {}:{} entered ({} user online)'.format(
            addr[0], addr[1], userNumber))
        client_socket.sendall(
            ("> Connected to the chat server " + "("+str(userNumber)+" user online)").encode())
        for clients in users.values():
            sender_ip = addr[0]
            sender_port = addr[1]
            if clients != client_socket:
                clients.sendall(
                    ("> New user "+str(sender_ip)+":"+str(sender_port)+" entered (" + str(userNumber)+" user online)").encode())

    else:
        print('> New user {}:{} entered ({} users online)'.format(
            addr[0], addr[1], userNumber))
        client_socket.sendall(
            ("> Connected to the chat server " + "("+str(userNumber)+" users online)").encode())
        for clients in users.values():
            sender_ip = addr[0]
            sender_port = addr[1]
            if clients != client_socket:
                clients.sendall(
                    ("> New user "+str(sender_ip)+":"+str(sender_port)+" entered (" + str(userNumber)+" users online)").encode())
```

위는 유저가 로그인했을 때 서버 측에서 작동하는 login_notice()함수이다. 우선 기본적으로 userNumber가 1인지, 1이 아닌지를 확인하여 각 서버와 클라이언트 측에 출력되는 내용이 다르게 하도록 설정하였고(=user or users), 서버에서 로그인한 클라이언트의 IP주소와 Port번호를 출력하도록 한다. 이 후에 현재 유저 정보에 담긴 클라이언트들을 확인하여 방금 연결이 된 클라이언트를 제외한 기존 클라이언트들에게 새로운 클라이언트가 접속했다는 메시지를 출력하도록 한다. 이 때에도 마찬가지로 새로 접속한 클라이언트의 IP주소와 Port 번호를 출력하도록 같은 양식을 이용한다.

```
# 유적 나가면 removeuser()함수를 호출한 뒤, 알림을 띄웁니다.
def userleaves(client_socket, addr):
    global userNumber
    removeuser(id(client_socket))
    if userNumber <= 1:
        print("< The user {}:{} left ({} user online)".format(
            addr[0], addr[1], userNumber))
        for clients in users.values():
            leave_ip = addr[0]
            leave_port = addr[1]
            if clients != client_socket:
                clients.sendall(("< The user " + str(leave_ip) + ":"+str(leave_port) +
                                 " left ("+str(userNumber)+" " + "user online)").encode())

    else:
        print("< The user {}:{} left ({} users online)".format(
            addr[0], addr[1], userNumber))
        for clients in users.values():
            leave_ip = addr[0]
            leave_port = addr[1]
            if clients != client_socket:
                clients.sendall(("< The user " + str(leave_ip) + ":"+str(leave_port) +
                                 " left ("+str(userNumber)+" " + "users online)").encode())
```

유저가 연결을 해제했을 때 서버 측에서 작동하는 userleaves()함수이다. 우선 기본적으로 removeuser()함수를 호출하여 현재 저장된 users 목록에서 해당 클라이언트에 대한 내용을 삭제한 뒤, 남아있는 userNumber 수를 갱신하도록 한다. 이후에 userNumber가 1보다 작은지, 큰지를 확인하여 각 서버와 클라이언트 측에 출력되는 내용이 다르게 하도록 설정하였고(=user or users), 서버에서 연결을 해제한 클라이언트의 IP주소와 Port번호를 출력하도록 한다. 이 후에 현재 유저 정보에 담긴 클라이언트들을 확인하여 방금 연결이 종료된 클라이언트를 제외한 기존 클라이언트들에게 새로운 클라이언트가 연결을 해제했다는 메시지를 출력하도록 한다. 이 때에도 마찬가지로 연결을 해제한 클라이언트의 IP주소와 Port 번호를 출력하도록 같은 양식을 이용한다.

```
# ctrl+c 입력을 처리하는 함수
# ctrl+c 가 입력되면 stop 변수를 True로 변환해줍니다.
def signal_handler(sign, frame):
    print("\nexit")
    server_socket.close()
    sys.exit()
```

마지막으로 서버를 종료할 때 이용하는 함수이다. 서버에서 Ctrl+C를 입력하면 KeyboardInterruptionError를 처리하는 대신에, Signal로 입력받는다. 이 때 Ctrl+C는 SIGNUM이라는 신호를 보내게 되는데, 이를 전달받으면 signal_handler()함수를 호출하여 예외를 일으키는 대신에 함수의 내용을 실행하도록 한다. 즉, Ctrl+C 가 입력되면 프로젝트의 요구사항대로 "exit"를 출력한 뒤, 서버의 소켓을 종료하고 프로그램을 종료하도록 하였다.

3.2 Explanation – 클라이언트 측

```
# 서버의 주소입니다. hostname 또는 ip address를 사용할 수 있습니다.
host = sys.argv[1]
# 서버에서 지정해 놓은 포트 번호입니다.
port = int(sys.argv[2])
# 소켓 객체를 생성합니다.
# 주소 체계(address family)로 IPv4, 소켓 타입으로 TCP 사용합니다.
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 지정한 HOST와 PORT를 사용하여 서버에 접속합니다.
client_socket.connect((host, port))

# 쓰레딩을 시작합니다.
t = threading.Thread(target=receivemsg, args=(client_socket,))
t.daemon = True
t.start()
```

클라이언트 소켓에서는 bind, listen, accept가 없으며 connect함수가 존재한다. Connect()함수를 통해 현재 생성되어 있는 서버 측의 소켓과 연결하게 된다. Connect()함수에는 호스트 주소와 포트번호를 입력 받아야 한다. 이때 마찬가지로 sys 모듈을 이용하여 host, port정보를 입력받게 하였다. 서버와 연결된 후에는 프로젝트의 양식처럼 알맞은 문구를 출력하도록 한다.

이후, 클라이언트에서 메시지를 수신하는 과정을 멀티쓰레딩을 이용하여 구현했다. 본 프로젝트에서는 서버 측에서 어떤 클라이언트의 메시지를 수신한 뒤, 그 메시지를 다른 모든 클라이언트들에게 재전송하도록 만들었다.

```python
def receivemsg(client_socket):
    while True:
        try:
            recvMessage = client_socket.recv(1024).decode()
            if not recvMessage:
                break
            print(recvMessage)

        except KeyboardInterrupt:
            pass
```

클라이언트에서는 recv()함수를 이용하여 서버 측에서 재전송(Echo)받은 내용을 전달받는다. 이후에 수신된 메시지를 decode()하도록 하고, 올바른 메시지라면 그 메시지를 출력하도록 한다. 한편, 서버 측에서 메시지를 재전송할 때, 송신한 클라이언트의 정보도 함께 전송하였기 때문에, 이 메시지를 그대로 decode()하여 출력하면, 메시지를 최초로 송신한 클라이언트의 IP, PORT 정보도 출력할 수 있다.

```python
# ctrl+c 입력을 처리하는 함수
# ctrl+c가 입력되면 종료합니다.
def signal_handler(sign, frame):
    print("\nexit")
    client_socket.close()
    sys.exit()


# 메시지를 전송합니다.
while True:
    stop = False
    signal.signal(signal.SIGINT, signal_handler)
    message = input()
    client_socket.sendall(message.encode('utf-8', "ignore"))
    print("[You]", message)
```

한편 클라이언트는 기본적으로 서버에 메시지를 전송하는 역할을 하고 있다. 서버 측에서 구현했던 것과 마찬가지로 signal()함수를 이용하여 Ctrl+C를 입력받을 수 있도록 구현했다. Ctrl+C가 입력되면 signal_handler()함수가 호출되어 "exit"문구를 출력하고 클라이언트 소켓이 종료되도록 구현하였다.

메시지의 전송의 경우 input()을 통해 내용을 받은 뒤, 그 내용을 encode()하여 전송하도록 한다. Encoding은 utf-8을 사용하였고, "ignore" 인자를 추가하여 완전한 글자가 아닌 에러에 대한 문제를 처리하였다. 전송을 한 후에는 자신의 채널에서도 [You], message라는 내용으로 다시 출력하도록 구현하였다.

4. Snapshots of Result

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 srv.py 0.0.0.0 3333
Chat Server started on port 3333.
> New user 127.0.0.1:45384 entered (1 user online)
> New user 127.0.0.1:45386 entered (2 users online)
```

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 cli.py 127.0.0.1 3333
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45386 entered (2 users online)
```

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 cli.py 127.0.0.1 3333
> Connected to the chat server (2 users online)
```

위 결과물은 각각 서버, 클라이언트1, 클라이언트2가 연결되는 상태를 순서대로 나타낸 것이다.

최초로 서버가 Chat Server를 시작하고, 클라이언트1이 접속한 뒤 클라이언트2가 접속한 상황이다.

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 srv.py 0.0.0.0 3333
Chat Server started on port 3333.
> New user 127.0.0.1:45390 entered (1 user online)
> New user 127.0.0.1:45392 entered (2 users online)
[127.0.0.1:45390] 안녕하세요?
[127.0.0.1:45392] 네 안녕하세요ㅋㅋ
```

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 cli.py 127.0.0.1 3333
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45392 entered (2 users online)
안녕하세요?
[You] 안녕하세요?
[127.0.0.1:45392] 네 안녕하세요ㅋㅋ
```

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 cli.py 127.0.0.1 3333
> Connected to the chat server (2 users online)
[127.0.0.1:45390] 안녕하세요?
네 안녕하세요ㅋㅋ
[You] 네 안녕하세요ㅋㅋ
```

위 결과물은 서버가 있고, 클라이언트 1, 2가 차례때로 접속한 상태에서 클라이언트2가 "안녕하세요?"라고 입력한 후, 클라이언트1이 "네 안녕하세요ㅋㅋ"라고 답변한 결과물이다.

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 srv.py 0.0.0.0 3333
Chat Server started on port 3333.
> New user 127.0.0.1:45390 entered (1 user online)
> New user 127.0.0.1:45392 entered (2 users online)
[127.0.0.1:45390] 안녕하세요?
[127.0.0.1:45392] 네 안녕하세요ㅋㅋ
< The user 127.0.0.1:45392 left (1 user online)
< The user 127.0.0.1:45390 left (0 user online)
^C
exit
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$
```

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 cli.py 127.0.0.1 3333
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45392 entered (2 users online)
안녕하세요?
[You] 안녕하세요?
[127.0.0.1:45392] 네 안녕하세요ㅋㅋ
< The user 127.0.0.1:45392 left (1 user online)
^C
exit
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$
```

```
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$ python3 cli.py 127.0.0.1 3333
> Connected to the chat server (2 users online)
[127.0.0.1:45390] 안녕하세요?
네 안녕하세요ㅋㅋ
[You] 네 안녕하세요ㅋㅋ
^C
exit
josunghyeon@DESKTOP-8TJH6BQ:/mnt/c/Users/josunghyeon/Desktop/network_project2$
```

위 그림은 클라이언트2가 접속을 종료하고, 클라이언트1이 접속을 종료한 뒤 최종적으로 서버에서 종료를 한 결과물이다. "Ctrl+C"를 입력했을 때 각 소켓은 종료되었으며, 시작과 종료 시 프로젝트 결과물에서 요구한 포맷으로 출력되도록 하였다.