TCP IP 기말고사 프로젝트

201944078 이대성

[1. 개요]

이번에 제가 만들고자 하는 프로그램은 기존에 있던 채팅 프로그램에 덧붙여서, IP와 Port를 직접 지정해 연결 & 연결 해제하는 기능과 서버 쪽에서 로그에 대한 내용들을 파일 출력으로 저장하는 기능을 추가한 프로그램입니다.
기존에 있던 교안과 인터넷에서 참고한 자료들을 토대로 파이참 환경에서 제작하였습니다.

[2. 설계 및 기능]

전체적인 구조는 서버 쪽과 클라이언트 쪽 모두 클래스로부터 만들어진 객체를 사용합니다. 지역 변수를 제외한 대부분의 값들을 속성 값으로 지정하였기 때문에 어느 위치에서든 접근이 가능하며, 실행 동작들은 버튼을 통한 메서드 실행이나 스레드를 통한 메서드 실행이 주요 처리 행위가 됩니다.

서버 쪽에서는 기존 프로그램과는 달리 GUI가 구현되어 있으며, 서버를 열고 닫는 조작이 가능하고, IP 주소와 Port 번호를 수동적으로 지정하여 서버를 여는 것도 가능합니다. 로그에 대한 내용은 GUI 화면에 접속, 해제, 메시지 내용 등이 실시간으로 나타나게 되며, 해당 내용들은 txt 파일에 추가되는 형식으로 저장됩니다.

클라이언트 쪽에서는 서버와 마찬가지로 IP 주소와 Port 번호를 수동적으로 지정할 수 있기 때문에 서버가 열려 있는 곳을 찾아서 들어갈 수 있습니다. 또한 접속을 하고 언제든지 버튼을 통해 나갈 수 있으며, 다시 접속을 요청하면 새로운 포트 번호로 입장을 할 수 있습니다. 닉네임 기능을 그대로 살려서 아무것도 지정하지 않았을 경우에는 Unknown으로 나타나고, 입력했을 경우에는 해당 닉네임으로 자신과 다른 상대방, 그리고 서버 쪽에까지 나타나게 됩니다. 자신의 접속/종료 알림문과 상대방의 접속/종료 알림문은 각각 자신의 채팅 화면에 나타나는 기능 또한 구현하였습니다.

자세한 설명은 3. 코드 분석 및 화면에서 다루겠습니다.

[3. 코드 분석 및 화면]

<서버>

```
if __name__ == "__main__":
    TCP_ChatServer()
    mainloop()
```

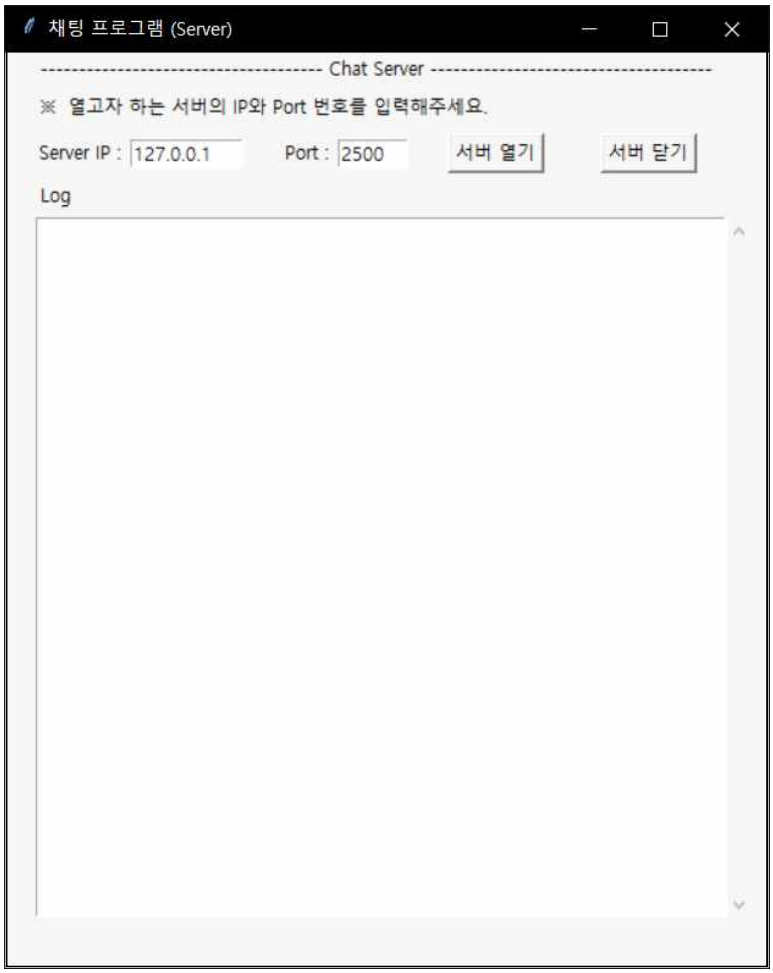
먼저 main이 되는 부분에서 클래스 생성자를 통해 객체를 만드는 것으로 서버 쪽 프로그램이 시작됩니다.

```
class TCP_ChatServer:
    list_clients = []   # 소켓을 담을 리스트
    server_socket = None

    def __init__(self):
        self.set_gui()  # GUI 생성 메서드
```

클래스의 필드로 소켓을 담을 리스트와 서버용 소켓 변수를 정의하고, 기본 생성자를 통해 GUI를 생성할 set_gui() 메서드를 실행합니다.

```
def set_gui(self):
    self.r = Tk()
    self.r.title('채팅 프로그램 (Server)')
    self.r.geometry('500x600')

    self.l_title = Label(self.r, text='------------------------------------ Chat Server ----------------------------------')
    self.l_title.place(x=20, y=0)

    self.l_explanation = Label(self.r, text='※  열고자 하는 서버의 IP와 Port 번호를 입력해주세요.')
    self.l_explanation.place(x=20, y=25)

    self.l_IP = Label(self.r, text='Server IP : ')
    self.l_IP.place(x=19, y=55)

    self.e_IP = Entry(self.r, width=10, text='127.0.0.1')
    self.e_IP.place(x=82, y=57)
    self.e_IP.insert(0, '127.0.0.1')

    self.l_Port = Label(self.r, text='Port : ')
    self.l_Port.place(x=180, y=55)

    self.e_Port = Entry(self.r, width=6, text='2500')
    self.e_Port.place(x=218, y=57)
    self.e_Port.insert(0, '2500')
```

```
self.b_Open = Button(self.r, text='서버 열기', command=self.open_server)
self.b_Open.place(x=290, y=53)

self.b_Close = Button(self.r, text='서버 닫기', command=exit)
self.b_Close.place(x=390, y=53)

self.l_Log = Label(self.r, text='Log')
self.l_Log.place(x=20, y=83)

self.t_Log_area = ScrolledText(self.r, height=35, width=64)
self.t_Log_area.place(x=20, y=108)
```

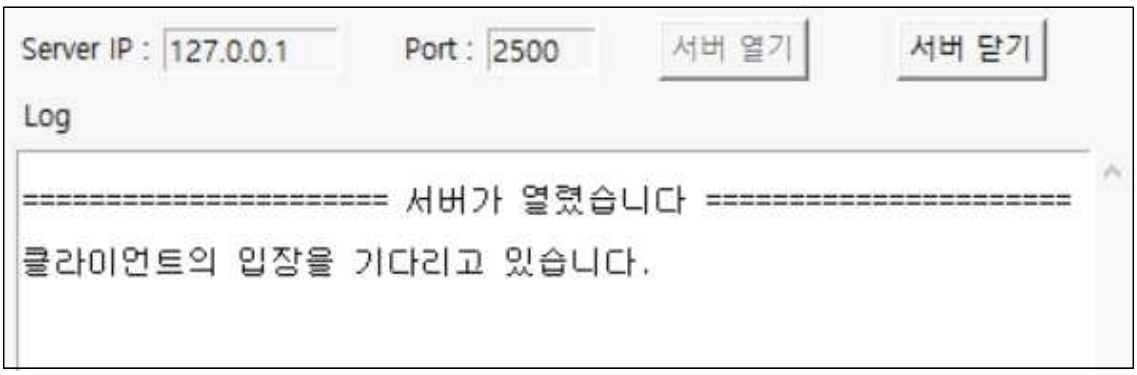


윈도우 객체를 생성하고 각각의 위젯들을 다음과 같이 배치합니다. 상단에는 제목과 설명 레이블이 존재하고, 그 다음 줄에는 열고자하는 서버의 IP와 Port 번호를 담을 Entry와 두 개의 버튼이 존재합니다. 해당 버튼은 서버를 여는 메서드와 닫는 메서드와 연결이 되어 있습니다. 로그 구역은 ScrolledText로 지정하여 스크롤바가 붙어 나오도록 지정합니다.

```python
def open_server(self):        # 서버 열기
    IP = self.e_IP.get()
    PORT = int(self.e_Port.get())
    self.b_Open['state'] = 'disabled'
    self.e_IP['state'] = 'readonly'
    self.e_Port['state'] = 'readonly'
    start_new_thread(self.make_socket, (IP, PORT))
```

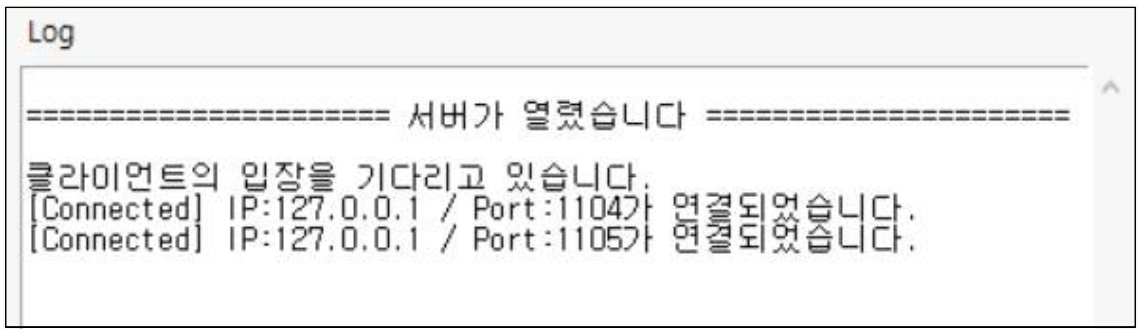
open_server() 메서드는 화면에 '서버 열기' 버튼을 누르면 동작하는 메서드로 서버를 열기 위한 준비를 하는 단계입니다. IP와 Port 번호를 입력했던 엔트리 상자에서 값을 가져오고 '서버 열기' 버튼은 비활성화 시킵니다. 그리고 엔트리 상자는 읽기만 가능하게 지정해주며 소켓을 생성하기 위해 스레드를 만들고 아까 받았던 IP와 Port번호를 넘겨줍니다.

```python
def make_socket(self, IP, PORT):        # 소켓 생성, 바인드, 리스닝 및 수락
    self.server_socket = socket(AF_INET, SOCK_STREAM)
    self.server_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)  # 소켓 옵션 설정
    self.server_socket.bind((IP, PORT))
    self.server_socket.listen()
    fp = open('Server Log.txt', 'a')
    self.t_Log_area.insert("end", '\n====================== 서버가 열렸습니다 ======================\n\n')
    fp.write('\n====================== 서버가 열렸습니다 ======================\n\n')
    self.t_Log_area.insert("end", '클라이언트의 입장을 기다리고 있습니다.\n')
    fp.write('클라이언트의 입장을 기다리고 있습니다.\n')
```



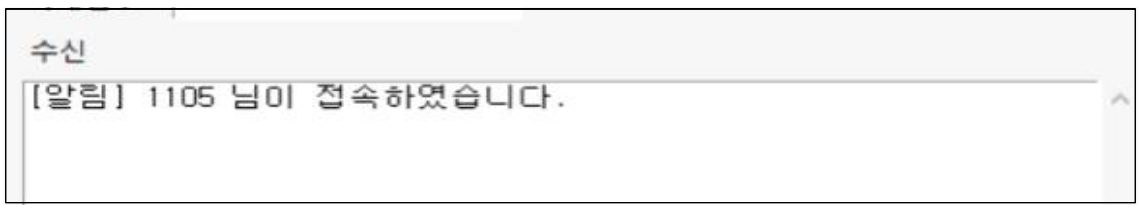
make_socket() 메서드에서는 소켓을 생성하고 바인드하며, 리스닝 및 수락을 하는 단계입니다. 소켓을 TCP 형식으로 생성하고 C언어에 memset과 비슷한 동작을 하는 setsockopt를 통해 WinError 10048 에러를 방지하고자 옵션 설정을 합니다.
그리고 매개변수로 받은 IP와 Port 번호로 바인드를 시키고 리스닝 과정을 거칩니다.
정상적으로 수행되면 로그 화면에 문구가 출력되며 해당 내용은 파일 출력을 통해 txt파일에 저장됩니다.

```
while True:
    fp = open('Server Log.txt', 'a')
    client_socket, addr = self.server_socket.accept()  # accept()을 통해 연결 수락
    if client_socket not in self.list_clients:      # 해당 소켓을 리스트에 삽입
        self.list_clients.append(client_socket)

    self.t_Log_area.insert("end", '[Connected] IP:{0} / Port:{1}가 연결되었습니다.\n'.format(addr[0], str(addr[1])))
    fp.write('[Connected] IP:{0} / Port:{1}가 연결되었습니다.\n'.format(addr[0], str(addr[1])))
    self.t_Log_area.yview(END)
    Thread(target=self.send_recv, args=(client_socket, addr)).start()  # 스레드 생성
```



계속해서 반복문을 통해 클라이언트 쪽에서 들어오는 요청을 accept()로 수락하게 되면 클라이언트와 통신할 소켓과 주소 정보를 가져옵니다. 중복되지 않는 소켓을 리스트에 담고 로그 화면에 IP와 Port 번호로 연결 문구를 띄웁니다. 마찬가지로 파일 출력까지 마치면 메시지 송신과 수신 처리를 담당할 스레드를 생성합니다.

```
def send_recv(self, client_socket, addr):   # 메시지 전송과 수신
    for client in self.list_clients:
        client.sendall('[알림] {0} 님이 접속하였습니다.'.format(str(addr[1])).encode())    # 알림문 전송
        self.t_Log_area.yview(END)
```



send_recv() 메서드에서는 받은 메시지를 다시 클라이언트 쪽에 넘겨주는 일을 하거나 접속 알림문을 전송하는 역할을 합니다. 클라이언트에서 서버로 접속을 하게 되면 해당 알림문을 먼저 클라이언트 쪽에 전송하여 접속 안내를 알립니다.

```
while True:
    try:
        fp = open('Server Log.txt', 'a')
        message = client_socket.recv(1024)
        self.t_Log_area.insert("end",'[Message Log] ({0}) - {1}  \n'.format(str(addr[1]), str(message.decode())))
        fp.write('[Message Log] ({0}) - {1}  \n'.format(str(addr[1]), str(message.decode())))
        self.t_Log_area.yview(END)
        for client in self.list_clients:
            client.sendall(message.decode().encode())   # 메시지 전송
```

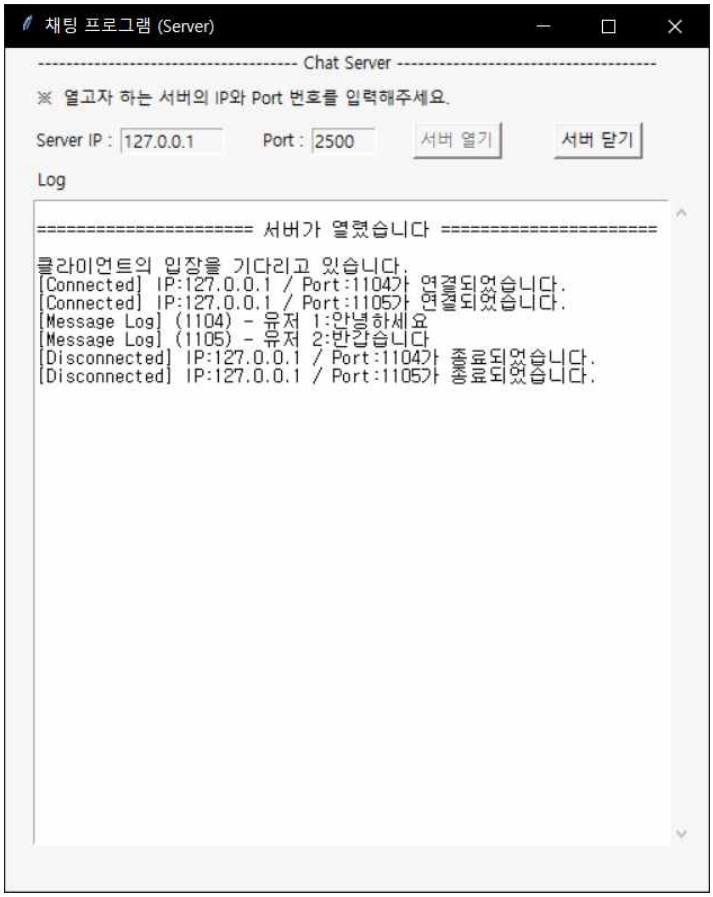
```
Log

====================== 서버가 열렸습니다 ======================

클라이언트의 입장을 기다리고 있습니다.
[Connected] IP:127.0.0.1 / Port:1104가 연결되었습니다.
[Connected] IP:127.0.0.1 / Port:1105가 연결되었습니다.
[Message Log] (1104) - 유저 1:안녕하세요
[Message Log] (1105) - 유저 2:반갑습니다
```

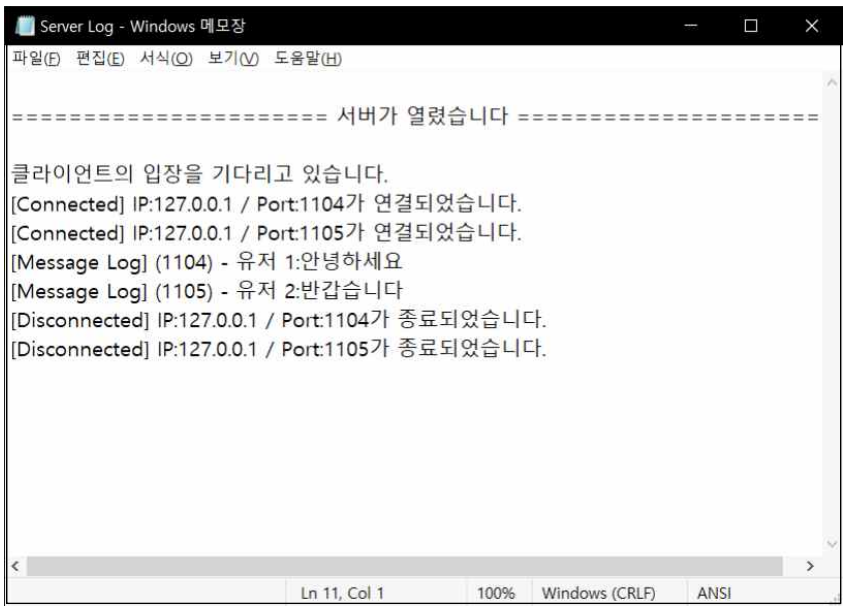
그리고 반복문을 통해 메시지를 만약 받게 된다면 해당 내용을 서버 로그 화면에 출력하고 다시 클라이언트 쪽으로 전해줍니다.

```
    except ConnectionResetError as e:
        fp = open('Server Log.txt', 'a')
        self.list_clients.remove(client_socket)     # 소켓 삭제
        for client in self.list_clients:
            client.sendall('[알림] {0} 님이 나갔습니다.'.format(str(addr[1])).encode())  # 알림문 전송
            self.t_Log_area.yview(END)
        self.t_Log_area.insert("end", '[Disconnected] IP:{0} / Port:{1}가 종료되었습니다.\n'.format(addr[0], str(addr[1])))
        fp.write('[Disconnected] IP:{0} / Port:{1}가 종료되었습니다.\n'.format(addr[0], str(addr[1])))
        self.t_Log_area.yview(END)
        break
client_socket.close()
```

```
Log

====================== 서버가 열렸습니다 ======================

클라이언트의 입장을 기다리고 있습니다.
[Connected] IP:127.0.0.1 / Port:1104가 연결되었습니다.
[Connected] IP:127.0.0.1 / Port:1105가 연결되었습니다.
[Message Log] (1104) - 유저 1:안녕하세요
[Message Log] (1105) - 유저 2:반갑습니다
[Disconnected] IP:127.0.0.1 / Port:1104가 종료되었습니다.
[Disconnected] IP:127.0.0.1 / Port:1105가 종료되었습니다.
```

만약 이 과정에서 클라이언트에서 접속을 종료하게 되면 리스트에 담았던 해당 소켓을 삭제하고 방을 나갔다는 안내 문구를 남은 클라이언트 쪽에 전송하게 됩니다.





프로그램이 종료되면 서버 로그 화면에 출력되었던 모든 내용들은 Server Log.txt 파일에 저장되어 남게 됩니다.

<클라이언트>

```
if __name__ == "__main__":
    TCP_ChatClient()
    mainloop()
```

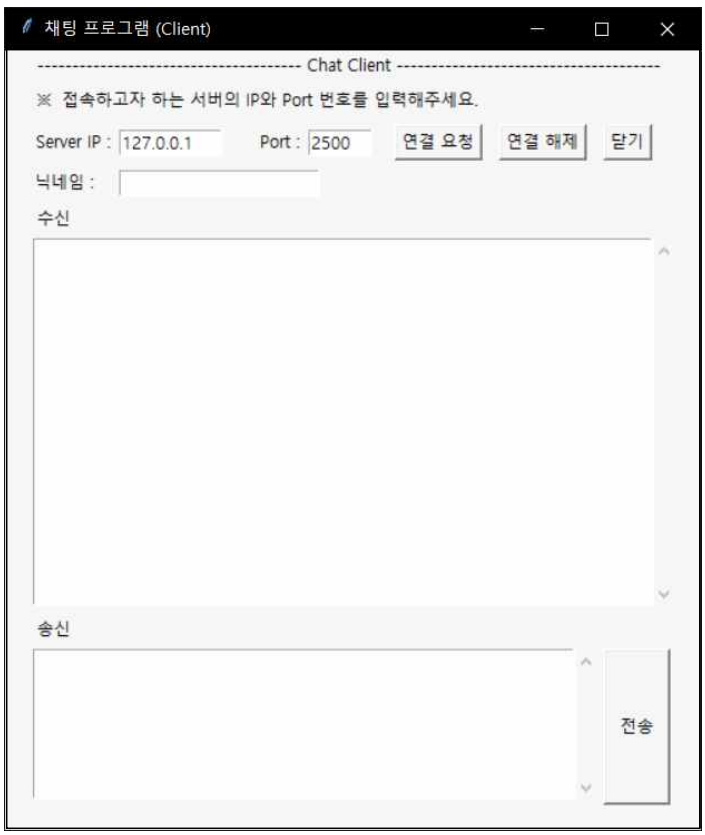
클라이언트 프로그램도 서버와 마찬가지로 클래스 객체를 만드는 것으로 시작됩니다.

```
class TCP_ChatClient:
    client_socket = None
    T_F_Discon = False
    T_F_send = False

    def __init__(self):
        self.set_gui()  # GUI 생성 메서드
```

클라이언트 소켓을 담을 변수와 메시지 전송을 하였는지, 접속 종료를 하였는지를 체크하는 변수를 정의하고 GUI 생성 메서드를 실행합니다.

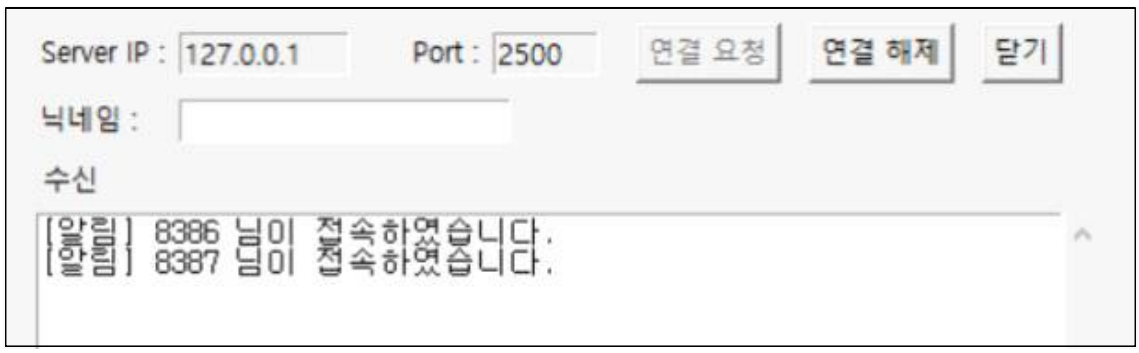
```
def set_gui(self):
    self.r = Tk()        # Tk 객체 생성
    self.r.title('채팅 프로그램 (Client)')
    self.r.geometry('500x560')

    self.l_title = Label(self.r, text='------------------------------------ Chat Client ----------------------')
    self.l_title.place(x=20, y=0)

    self.l_explanation = Label(self.r, text='※  접속하고자 하는 서버의 IP와 Port 번호를 입력해주세요.')
    self.l_explanation.place(x=20, y=25)

    self.l_IP = Label(self.r, text='Server IP : ')
    self.l_IP.place(x=19, y=55)

    self.e_IP = Entry(self.r, width=10, text='127.0.0.1')
    self.e_IP.place(x=82, y=57)
    self.e_IP.insert(0, '127.0.0.1')

    self.l_Port = Label(self.r, text='Port : ')
    self.l_Port.place(x=180, y=55)

    self.e_Port = Entry(self.r, width=6, text='2500')
    self.e_Port.place(x=218, y=57)
    self.e_Port.insert(0, '2500')
```

```
self.b_Connect = Button(self.r, text='연결 요청', command=self.connect_server)
self.b_Connect.place(x=280, y=53)

self.b_Disconnect = Button(self.r, text='연결 해제', command=self.release_server)
self.b_Disconnect.place(x=355, y=53)

self.b_Close = Button(self.r, text='닫기', command=exit)
self.b_Close.place(x=430, y=53)

self.l_Nickname = Label(self.r, text='닉네임 : ')
self.l_Nickname.place(x=18, y=84)

self.e_Nickname = Entry(self.r)
self.e_Nickname.place(x=82, y=86)

self.l_Reception = Label(self.r, text='수신')
self.l_Reception.place(x=20, y=110)

self.t_Reception_area = ScrolledText(self.r, height=20, width=63)
self.t_Reception_area.place(x=20, y=135)

self.l_Send = Label(self.r, text='송신')
self.l_Send.place(x=20, y=405)

self.t_Send_area = ScrolledText(self.r, height=8, width=55)
self.t_Send_area.place(x=20, y=430)

self.b_Send = Button(self.r, text='전송', command=self.send_switch)
self.b_Send.place(x=430, y=430, relheight='0.2', relwidth='0.095')

self.t_Send_area.bind('<Return>', self.enter_press)  # 엔터 키와 바인드
```



화면 구성은 서버와 비슷하지만 다른 점은 연결 요청과 연결 해제 버튼, 그리고 송신할 메시지를 입력할 텍스트 영역 부분입니다.
연결 요청 버튼과 연결 해제 버튼은 각각 서버로의 접속을 처리할 connect_server() 메서드와 release_server() 메서드와 연결되어 있습니다.
전송 버튼은 엔터키로도 동작하도록 바인드 되어 있으며, 해당 버튼은 메시지를 전송하기 위한 처리작업을 도울 send_switch() 메서드와 연결되어 있습니다.

```
def connect_server(self):   # 서버 접속 처리 메서드
    self.e_IP['state'] = 'readonly'
    self.e_Port['state'] = 'readonly'
    self.b_Connect['state'] = 'disabled'
    self.b_Disconnect['state'] = 'active'
    self.T_F_Discon = False

    IP = self.e_IP.get()
    PORT = int(self.e_Port.get())

    self.client_socket = socket(AF_INET, SOCK_STREAM)
    self.client_socket.connect((IP, PORT))

    Thread(target=self.message_send, args=(self.client_socket, )).start()
    Thread(target=self.message_recv, args=(self.client_socket, )).start()
```



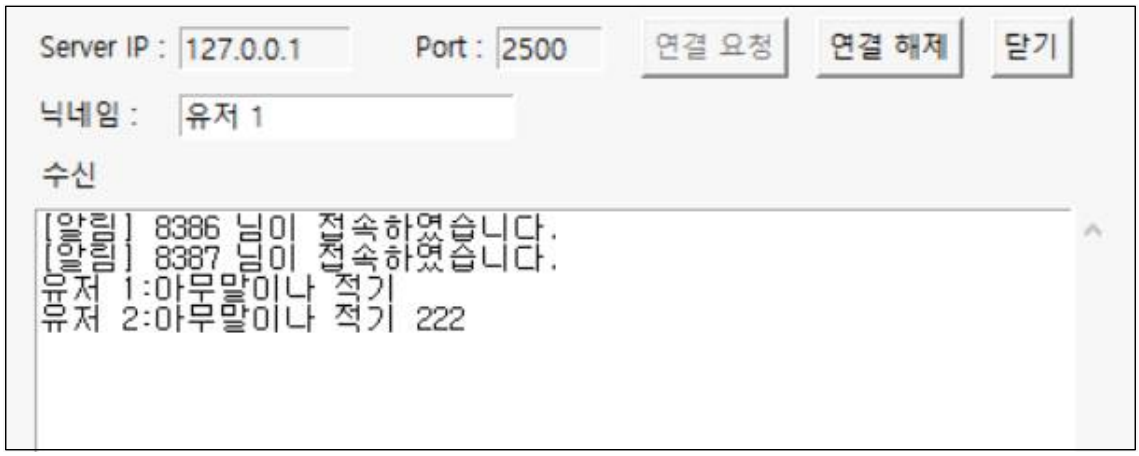
connect_server() 메서드가 실행되면 IP와 Port 번호를 입력할 엔트리 상자는 읽기 전용으로 변하고 연결 요청 버튼은 비활성화, 연결해제 버튼은 활성화되고 접속 종료에 대한 체크 변수를 False로 지정합니다. IP와 Port 번호를 가져와서 소켓을 TCP 형식으로 생성하고 서버 쪽으로 connect 요청을 하게 됩니다.
정상적으로 접속되면 수신 화면에 접속 문구를 띄우는데 이 때, 다른 클라이언트에서도 접속을 하면 자신의 화면에 접속 문구가 나타나게 됩니다.
접속이 완료되면 메시지를 송수신할 각각의 스레드를 생성합니다.

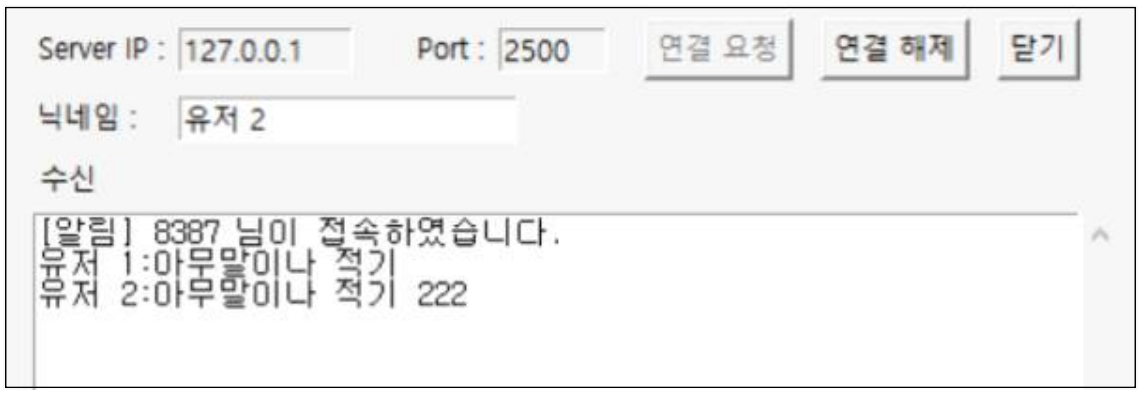
```
def release_server(self):   # 서버 연결 해제 처리 메서드
    self.e_IP['state'] = 'normal'
    self.e_Port['state'] = 'normal'
    self.b_Connect['state'] = 'active'
    self.b_Disconnect['state'] = 'disabled'
    self.T_F_Discon = True
```

release_server() 메서드는 연결 해제 버튼을 눌렀을 때 실행되며 IP와 Port번호 엔트리 상자를 입력 가능하게 설정하고 '연결 요청' 버튼은 활성화, '연결 해제' 버튼은 비활성화를 시킵니다. 그리고 연결이 해제되었기 때문에 이를 알려주는 T_F_Discon 값을 True로 지정해 현재 유저가 방에서 나갔음을 의미해줍니다.

```
def message_send(self, client_socket):  # 메시지 전송 메서드
    while True:
        if self.T_F_send:
            nickname = self.e_Nickname.get().strip() + ':'
            if nickname == ':':
                nickname = 'Unknown:'
            message = self.t_Send_area.get(1.0, "end").strip()
            data = nickname + message
            client_socket.send(data.encode())
            self.t_Send_area.delete(1.0, "end")
            self.T_F_send = False
        else:
            if self.T_F_Discon:
                client_socket.close()
                exit()
            sleep(0.1)
```

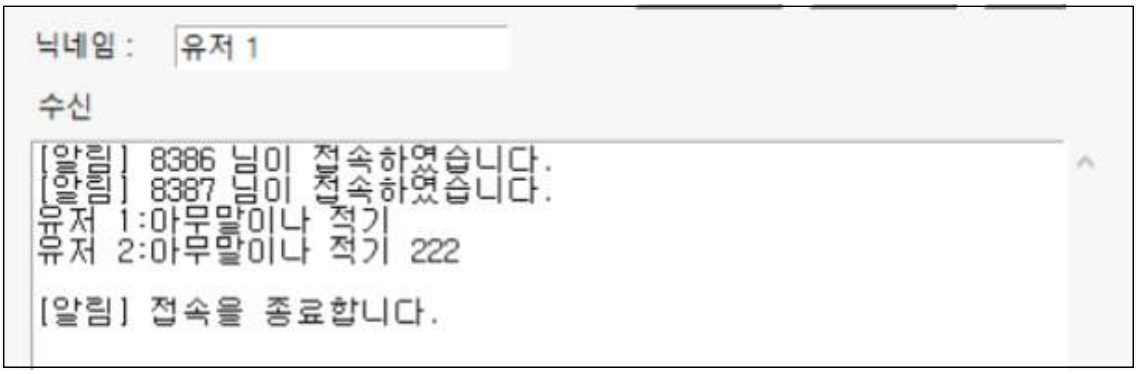
```
def send_switch(self):  # 메시지 전송 전 처리 작업 메서드
    self.T_F_send = True
```

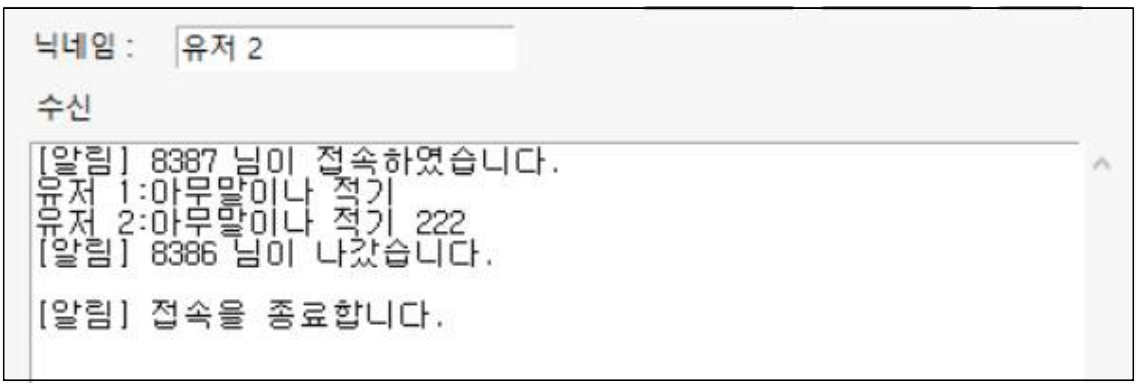




message_send() 메서드에서는 조건문을 통해 메시지를 전송할지 말지를 판단합니다. 클라이언트 화면에서 만약 사용자가 메시지 전송 버튼을 누르게 되면 T_F_send 변수가 True 값으로 변하는데 이는 message_send() 메서드에서 전송할 메시지가 있다는 것을 의미합니다. True 값이라면 닉네임 엔트리 상자에서 닉네임을 가져오고 메시지 내용과 결합해 서버 쪽으로 전송합니다. 닉네임이 없다면 자동으로 Unknown으로 설정됩니다. 만약 False 값이라면 보낼 메시지가 없다는 것을 의미하고 거기에 종료 되었는지에 대한 값을 나타내는 T_F_Discon이 True 값이라면 프로그램을 종료하게 됩니다.

```
def message_recv(self, client_socket):  # 메시지 수신 메서드
    while True:
        try:
            data = client_socket.recv(1024)
            self.t_Reception_area.insert("end", str(data.decode()) + '\n')
            self.t_Reception_area.yview(END)
            self.t_Reception_area.see('end')
        except ConnectionAbortedError as e:
            self.t_Reception_area.insert("end", '\n[알림] 접속을 종료합니다.\n')
            self.t_Reception_area.yview(END)
            exit()
```

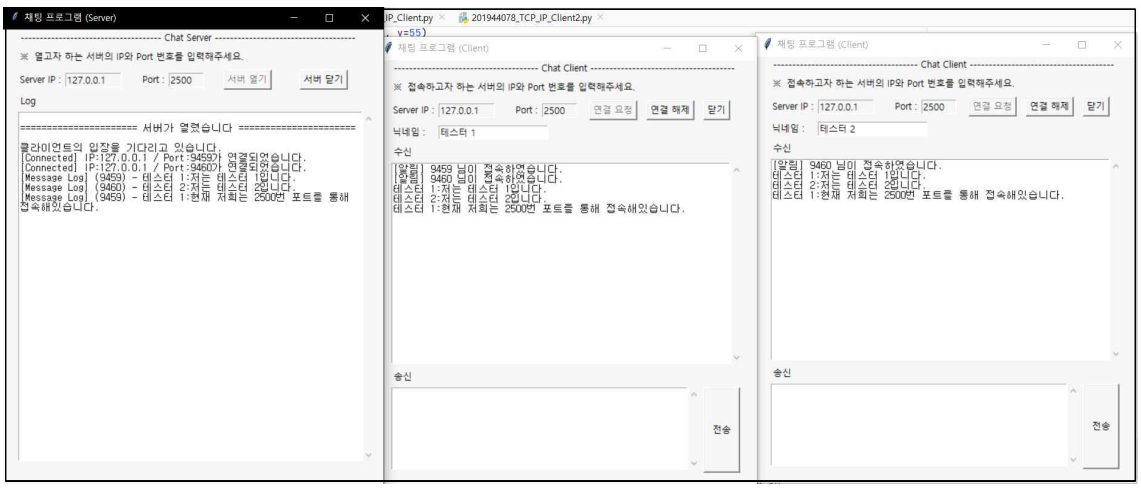




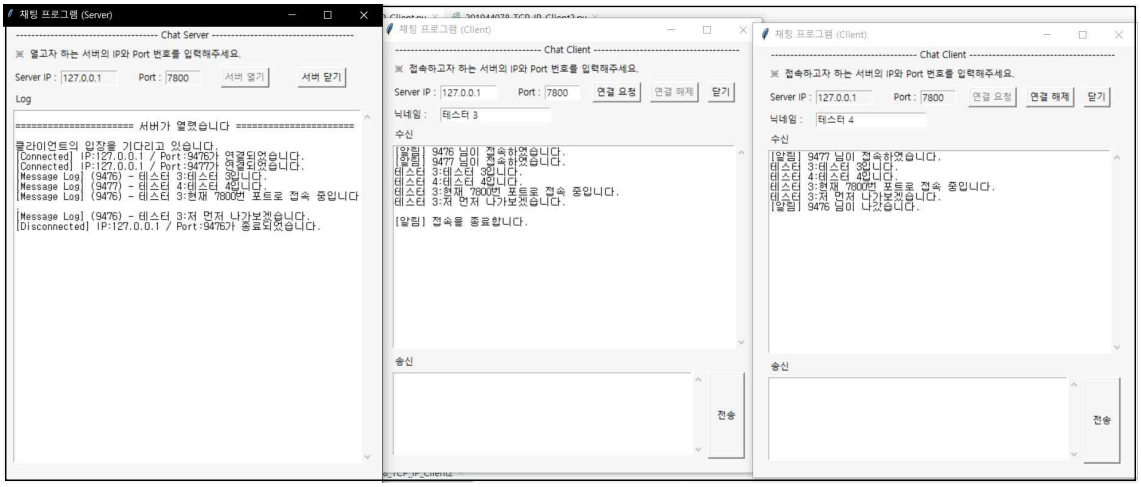
message_recv() 메서드에서는 서버로부터 전송받은 메시지를 변수에 넣어 수신 화면에 나타냅니다. 이 때 클라이언트가 나가게 되면 본인 화면에서는 접속을 종료했다는 문구가 나타나고 상대방 화면에서는 xxxx님이 나갔다는 문구가 나타나게 됩니다.

[4. 결과 화면]

<2500번 포트로 접속해 있는 경우>



<7800번 포트로 접속해 있는 경우>



<Log 기록>