컴퓨터네트워크

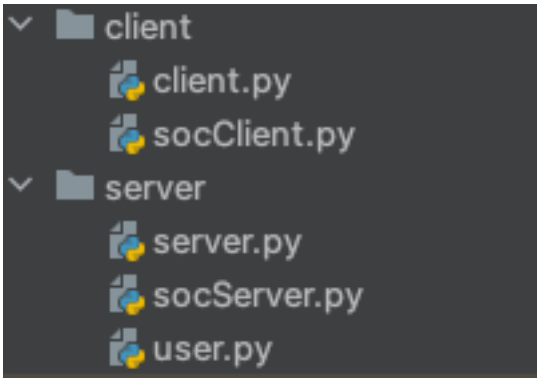
Socket Programming

2018008104 김종연

목차

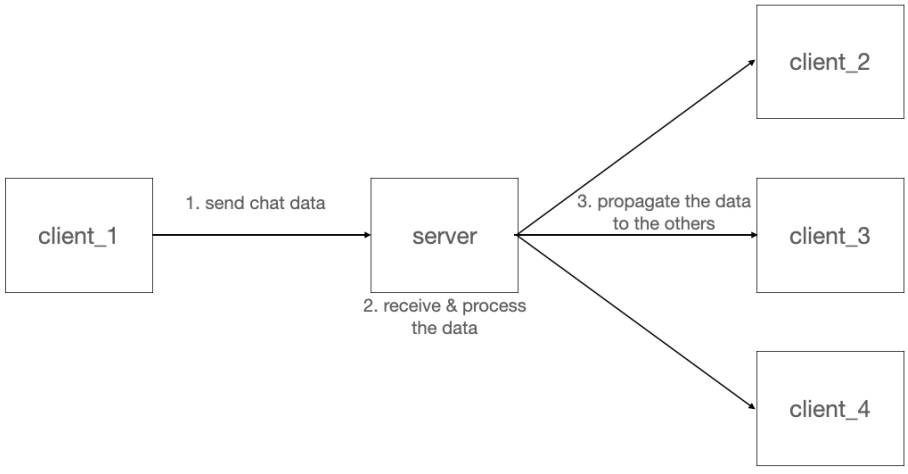
1. 프로젝트 구조

이 프로젝트는 client-server 구조를 갖고 있는 dummy chatting service 이다. client, server 두 개의 디렉토리로 이루어져 있고, 각각의 파일에 대한 설명은 다음과 같다.



- client directory 에는 entry point 인 client.py 와 client-side socket 이 구현되어 있는 socClient.py 가 포함되어 있다.
- server directory 에는 entry point 인 server.py 와 server-side socket 이 구현되어 있는 socServer.py, 그리고 client-side socket 에 nickname 과 address 를 더하여 wrapping 한 member-only class 이다.

- 본 프로젝트의 논리적인 구조는 다음과 같다.



a. 한 클라이언트에서 데이터를 생성하고 소켓을 통해 전달한다.
b. 서버는 소켓을 통해 해당 데이터를 받고 여러가지 작업을 처리한다.
c. 데이터를 전송한 클라이언트 (그림에서는 client_1)를 제외한 다른 접속된 클라이언트에게 수신된 내용을 전파한다.

2. 코드 설명

이 섹션은 서버/클라이언트 구현체인 socClient.py 와 socServer.py 에 대한 설명으로 구성되어 있다. 다른 파일들은 간단한 구조를 가지고 있어 별도의 설명이 필요하지 않다고 판단하였다.

socClient.py

socClient.py 는 class SocClient 를 포함하고 있고, 이는 socket 통신에서 client-side socket 과 관련된 feature 를 가지고 있다.

a. exec_client

exec_client 는 생성자에서 만들어진 client-side socket 과 server-side socket 사이의 연결을 개설한다. 또, server 로부터 데이터를 받아올 thread 를 별도로 정의하고 있다.

```
def exec_client(self, addr='localhost', port=8080):
    try:
        # establish a connection
        self.cliSocket.connect((addr, port))

        # As the connection is established,
        # get a nickname for the chatroom and send it to the server.
        nickname = input("Type your nickname>> ")
        self.cliSocket.send(nickname.encode())
```

```
        # open a new thread to get asynchronous messages
        # from the client-side socket
        start_new_thread(self.threaded_recv, ())

        # to avoid being interrupted by coroutine in the thread,
        # when input string is formatted, I used asyncio and prompt_toolkit
        asyncio.run(self.send_message(nickname))

        self.cliSocket.close()
        print("Closed chatroom successfully.")
    finally:
            self.cliSocket.close()
```

b. threaded_recv

위에서 언급한 수신용 thread 에서 수신 작업을 위해 호출할 method 이다. chat data 가 수신되는 동안 해당 데이터를 decode 하고 출력한다.

```
def threaded_recv(self):
    # a method called when the thread starts.
    # receive chat data and print it to client console.
    try:
        while True:
            # receive the data and decode it to normal string
            data = self.cliSocket.recv(1024)
            if not data:
                print("Server disconnected!")
                break
            dec = data.decode()

            print(dec)

        self.cliSocket.close()

    except Exception as e:
        pass
```

c. send_message

채팅을 생각하면, 입력창은 마지막 줄에 고정되고, 위로 대화 내용들이 쌓여 올라가는 구조를 갖고 있다. 그것을 구현하기 위해서, 비동기로 input 을 받도록 구현했다. 이 project 의 유일한 dependecy 인 prompt-toolkit 을 사용했다.

```
async def send_message(self, nickname):
    # the function for input interface
    # that keeps the input message from being broken
    # by newly arrived chat data.

    # create PromptSession
    session = PromptSession(message=nickname + ": ")
    with patch_stdout():
        while True:
            # get async data from the prompt
            message = await session.prompt_async()

            # when the message is "quit",
            # the user quit from the chatroom
            if message == "quit":
                break

            self.cliSocket.send(message.encode())
```

socServer.py

a. up_server

up_server 는 server-side socket 에서의 bind, listen, accept 단계를 담당한다. listen 이 호출된 이후에는, loop 을 돌면서 accept 가 일어날 때마다, 새로운 thread 를 생성해서 필요한 process 를 처리하도록 했다.

```
def up_server(self, port=8080, n_of_client=3):
    # this dummy server will only be used in localhost
    # default number of clients to listen is 3
    print(">>> Server is up on PORT:" + str(port))
    self.sevSocket.bind(('localhost', port))
    self.sevSocket.listen(n_of_client)

    try:
        while True:
            # accept client and open new thread
            # calling accept_client_threaded method
            print(">>> Waiting for clients...")
            (client, addr) = self.sevSocket.accept()
            start_new_thread(self.accept_client_threaded, (client, addr))

    finally:
        self.sevSocket.close()
        print("Socket closed in unexpected way.")
```

b. accept_client_threaded

accept 가 일어난 이후 thread 가 생성되면서 수행될 작업들이 모여있는 method 이다. 먼저 chat room 에서 쓰일 nickname 을 client 로부터 전달받고, 각 클라이언트에 맞는 입장 메시지들을 전달한다. 그 후 loop 으로 들어가 채팅 데이터를 받기 시작한다. "quit"이 입력되면 채팅을 종료한다.

```
def accept_client_threaded(self, client: socket, addr):
    # get nickname and append to connected list
    nick = client.recv(1024)
    now_user = User(nick.decode(), client, addr)
    self.connected_users.append(now_user)

    # print join message to the server console
    message = now_user.nickname + "(" + str(addr[1]) + ") has joined the chatroom!!"
    message += "₩nNow participants: " + str(len(self.connected_users))
```

```
    print(message)

    # send the welcome message to the current client
    client.send(">>> You have joined the chatroom now.₩n".encode())

    # and send the join message to the other clients
    self.propagate(now_user, ("₩n"+message+"₩n").encode())

    while True:
        try:
            # receive the chat data from the client
            # and format the chat data to "nickname(port): message"
            chat = client.recv(1024)
            temp_chat = now_user.nickname + "(" + str(now_user.addr[1]) + "): " + chat.decode()

            if not chat:
                # if the connection is not alive anymore
                # propagate the message to the other participants
                message = "User ₩"" + now_user.nickname + "₩" left chatroom."
                print(message)
                self.propagate(now_user, message.encode())
                break

            # if the data received correctly
            # print related message to the server console, and propagate the formatted chat
data.
            print("chat received from " + now_user.nickname + "(" + str(addr[0]) + ":" +
str(addr[1]) + ")")
            self.propagate(now_user, temp_chat.encode())

        except ConnectionResetError as e:
            print("Error occurred: " + e)
            print("Close connection.")
            break

    if now_user in self.connected_users:
        # when the socket finished the connection,
```

```
        # remove the corresponding User from the self.connected_users
        self.connected_users.remove(now_user)
        print("Now participants: " + str(len(self.connected_users)))

    client.close()
```

c. propagate

한 클라이언트로부터 받은 메시지를 그 클라이언트를 제외한 다른 클라이언트에게 전파할 때 사용되는 메소드이다.

```
def propagate(self, now_user, message):
    for other in self.connected_users:
        if other is not now_user:
            other.client.send(message)
    print(">>> Message successfully propagated.₩n")
```

3. 실행 예시

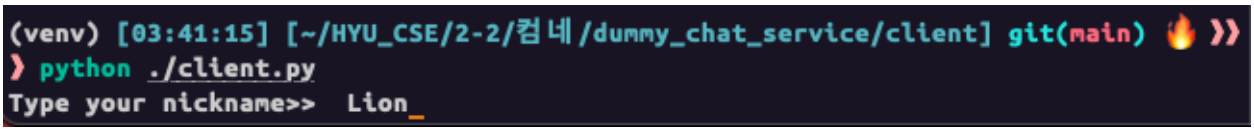
3.1 서버 시작

server.py 를 시작하면 서버가 어느 포트에서 시작했는지 알려주는 메시지와, client 를 기다리고 있다는 메시지가 뜬다.

```
(venv) [03:38:45] [~/HYU_CSE/2-2/컴네/dummy_chat_service/server] git(main) 🔥 ❯❯
❯ python ./server.py
>>> Server is up on PORT:8080
>>> Waiting for clients...
```

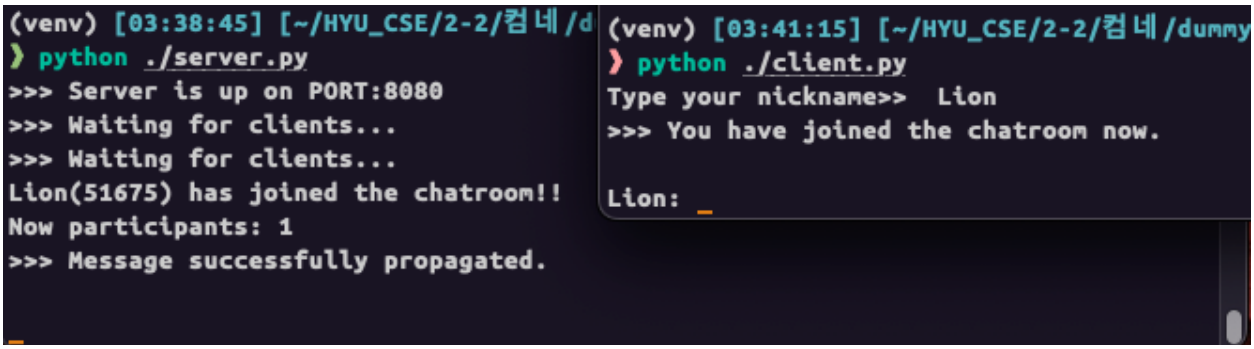
3.2 클라이언트 시작

client.py 를 시작하면 nickname 을 입력받는 창이 나온다. 닉네임을 입력해보자.



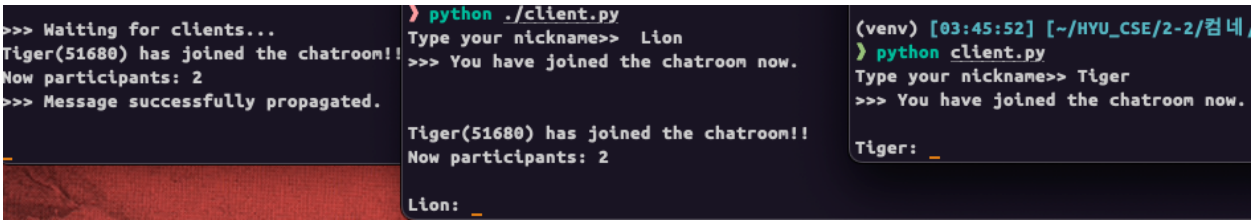
3.3 닉네임을 입력하고 난 서버와 클라이언트 콘솔

왼쪽 터미널이 서버, 오른쪽이 클라이언트다. 닉네임을 입력하고 엔터를 치니,
클라이언트에는 채팅방에 입장했다는 메시지와 함께, 입력 인터페이스가 나타난다.
서버에는 접속한 유저의 닉네임과 포트 번호, 그리고 현재 채팅에 참여하고 있는 인원 정보가 출력된다.



3.4 새로운 클라이언트가 접속했을 때

또 다른 클라이언트(제일 오른쪽 터미널) Tiger 가 채팅방에 입장했다.
기존에 접속해 있던 클라이언트 Lion 의 채팅방에는 Tiger 라는 닉네임과 포트 번호를 알려주면서 새로 접속했음을 알려준다. 서버에는 위의 3 번 상황에서와 동일한 메시지가 출력된다.



3.5 채팅 메시지를 전송했을 때

채팅을 입력하면 서버측에서는 어떤 유저로부터 채팅을 받았는지, 그리고 propagation 이 성공적으로 이루어졌는지에 대한 내용이 print 되고, client 에는 "nickname(port): message" 포맷으로 메시지가 출력된다. 자기 자신의 메시지는 포트 주소가 빠진 상태로 출력된다.

3.6 클라이언트가 연결을 종료했을 때

채팅창에 quit 을 입력하면 연결을 종료한다.

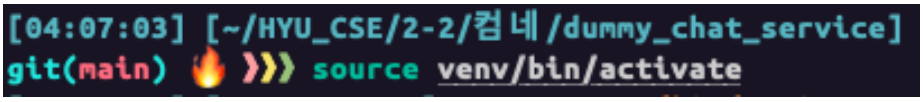
유저가 연결을 종료하면, 서버와 다른 클라이언트에는 해당 유저가 채팅방을 떠났다는 메시지가 전달된다. 연결을 종료한 유저의 콘솔창에는 성공적으로 채팅방을 종료했다는 메시지가 나타난다.

4. 실행 안내

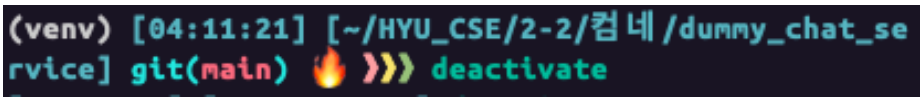
채팅 interface 구현을 위해 불가피하게 dependency 가 존재하게 되었다. 따라서 첨부하는 코드에 python virtualenv 를 함께 첨부할 예정인데, 여기서 실행할 방법이 두 가지로 나뉜다. 하나는 venv 를 이용하는 것이고, 나머지 하나는 requirements.txt 를 이용해서 해당 dependency 를 활용하는 방법이다.

4.1.1 virtualenv 활용

project root directory 에서 해당 명령어를 사용하면 된다. 이 virtualenv 는 맥에서 생성되었으므로, 맥이나 리눅스 환경에서만 실행된다.

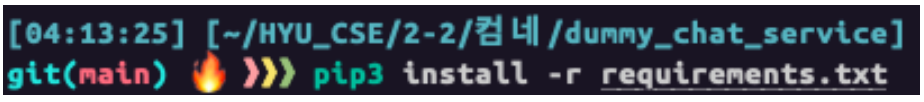


올바르게 실행되었다면, 터미널의 디렉토리 표시 앞에 (venv)라고 나타날 것이다. 여기서 deactivate 를 입력하여 venv 에서 빠져나올 수 있다.



4.1.2 requirements.txt 활용

터미널에 pip3(혹은 pip) install -r requirements.txt 를 입력하면 자동으로 의존성 설치가 시작된다.



4.2 서버 및 클라이언트 실행

root directory 하위에 있는 client, server directory 에 각각 있는 entry point (client.py, server.py)를 실행하면 서버와 클라이언트를 각각 실행할 수 있다.