http 프로토콜을 사용한 tcp 통신 프로그램

20213083 정찬하

사용한 언어: python
사용한 모듈: socket, sys

테스트 실행 환경:

운영체제: windows 11
프로세서: AMD Ryzen 5 PRO 4650U with Radeon Graphics 2.10 GHz

실행 가능 환경:
운영체제: windows, linux, macOS

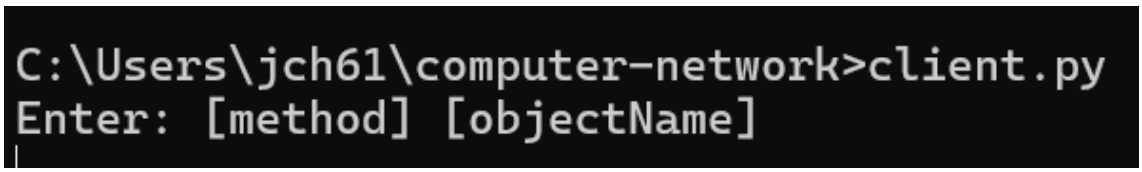
프로젝트 소개:
HTTP의 GET/HEAD/POST/PUT method를 구현한 간단한 tcp 통신 프로그램.

사용하는 방법:

1. 우선 client.py 파일과 server.py 파일을 다운 받고 각 server의 주소가 될 IP와 port 주소를 각 파일의 ip와 port 변수에 설정한다.

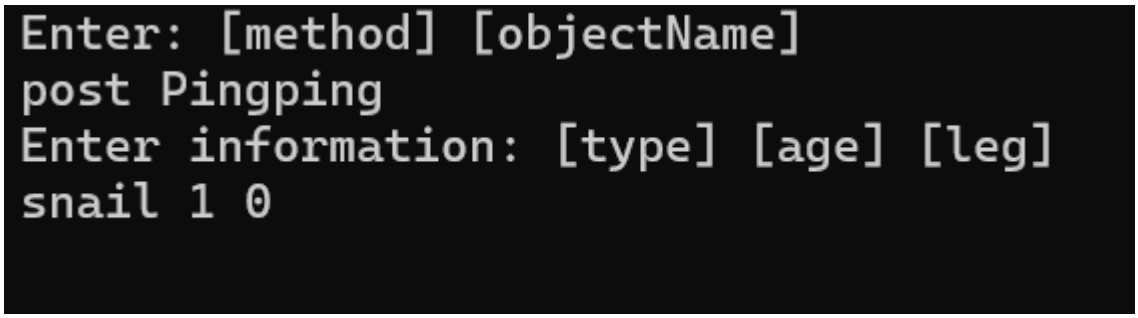
2. server.py을 실행해서 server를 시작한 후에, client로 서버에 접속한다.

3. 다음 사진과 같이 사용자의 입력을 요청하는 문구가 나온다.



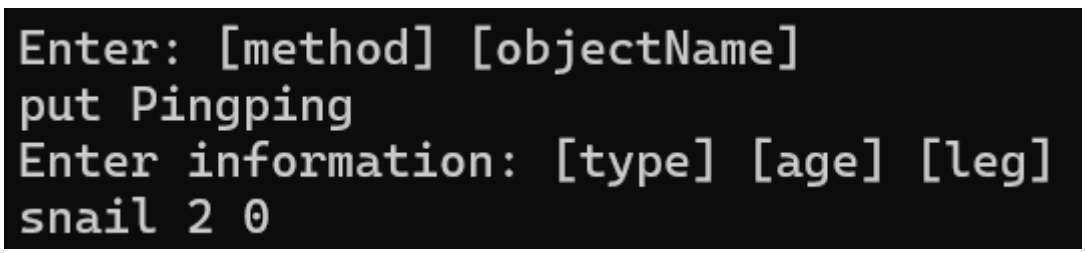
명령은 [method] [object name]의 형태로 적어야 한다.
사용할 수 있는 method:

1. get: server의 data table에서 원하는 data 정보를 가져온다. get all 명령어를 사용해서 server의 모든 데이터를 읽을 수 있다.
2. head: server의 data table에서 원하는 data 정보의 헤더를 가져온다. response message에 response body가 포함되어 있지 않고 header 정보만 가져온다는 것이 get과의 차이점이다.
3. post: server의 data table에 새로운 객체를 추가한다.



post [object name]을 입력하면 다음과 같이 추가 입력을 요청한다. [object name]의 field들을 설정하기 위한 정보이며 [type] [age] [leg] 순으로 입력한다.

4. put: server의 data table에 있는 data를 수정한다.



post와 마찬가지로 추가 입력을 요청한다.

사용 예시:

```
Enter: [method] [objectName]
get all
```

get all 명령어로 서버의 data table의 모든 객체 정보를 요청한다.

```
----------REQUEST----------

GET / HTTP/1.1
Content-Type: application/json
Content-Length: 0



------------------------------
```

server.py에는 client가 server에 도착한 request message가 출력된다.

```
----------RESPONSE----------

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 143

{'Doldol': {'type': 'chicken', 'age': 1, 'leg': 2}, 'Baduk': {
'type': 'dog', 'age': 3, 'leg': 4}, 'Yaong': {'type': 'cat', '
age': 2, 'leg': 4}}
------------------------------
```

client.py에서는 server가 client에 전달한 response message가 출력되고 response message의 body를 통해 요청한 정보를 확인할 수 있다.

객체 추가하기:

```
Enter: [method] [objectName]
post Pingping
Enter information: [type] [age] [leg]
snail 1 0
```

```
----------REQUEST-----------

POST /Pingping HTTP/1.1
Content-Type: application/json
Content-Length: 55

{'Pingping': {'type': 'snail', 'age': '1', 'leg': '0'}}
----------------------------
```

```
----------RESPONSE-----------

HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 55

{'Pingping': {'type': 'snail', 'age': '1', 'leg': '0'}}
----------------------------
```

response message를 통해 객체가 생성됐음을(Created) 알 수 있다.

```
get Pingping
----------RESPONSE-----------

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 55

{'Pingping': {'type': 'snail', 'age': '1', 'leg': '0'}}
----------------------------
```

다시 get 명령을 통해 server의 data table에 정보가 잘 저장되었음을 확인해 볼 수도 있다.

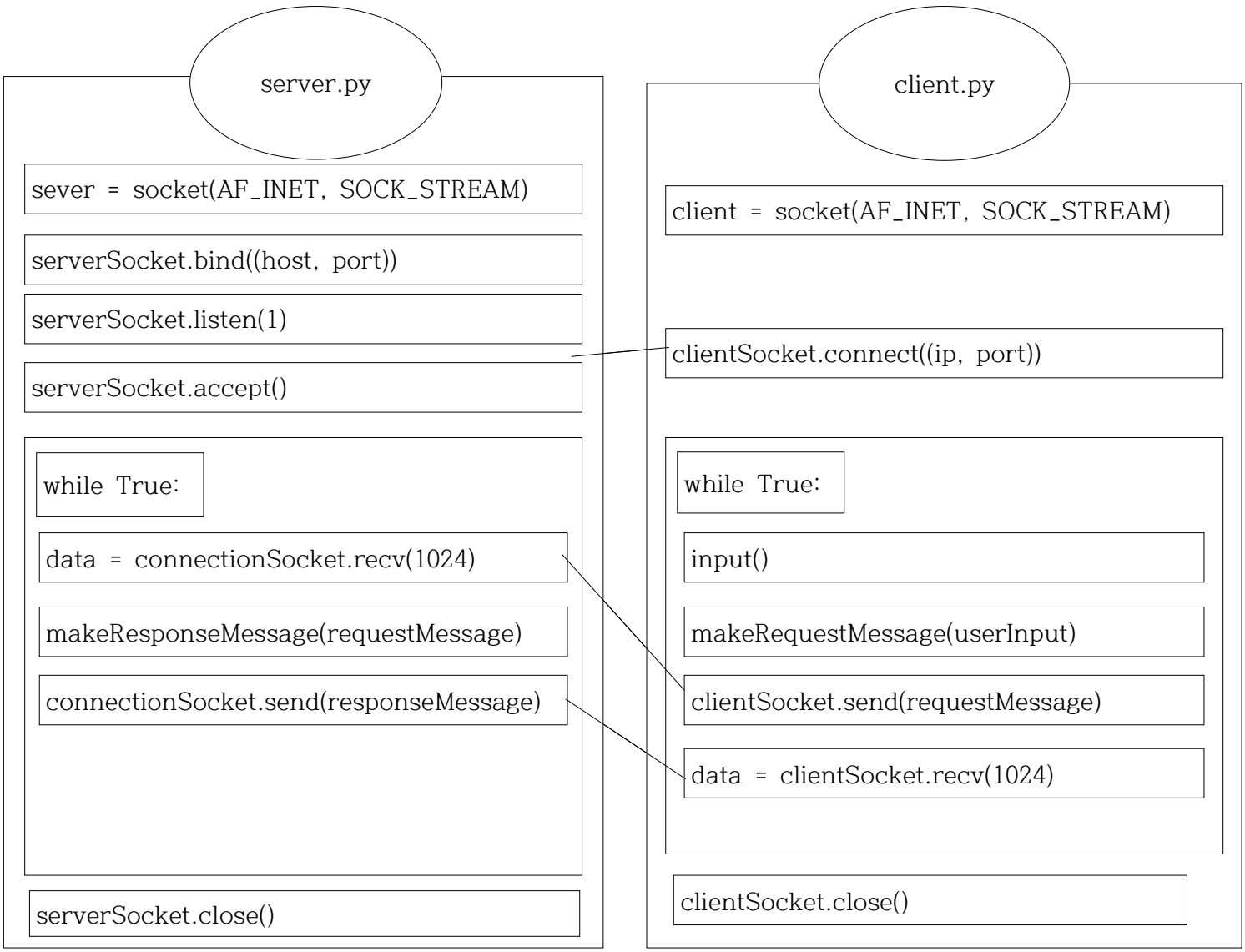
```
head Pingping
----------RESPONSE-----------

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 0


--------------------------------
```

head 명령으로는 reponse message의 헤더 정보만 가져올 수 있다.

-프로그램 구조

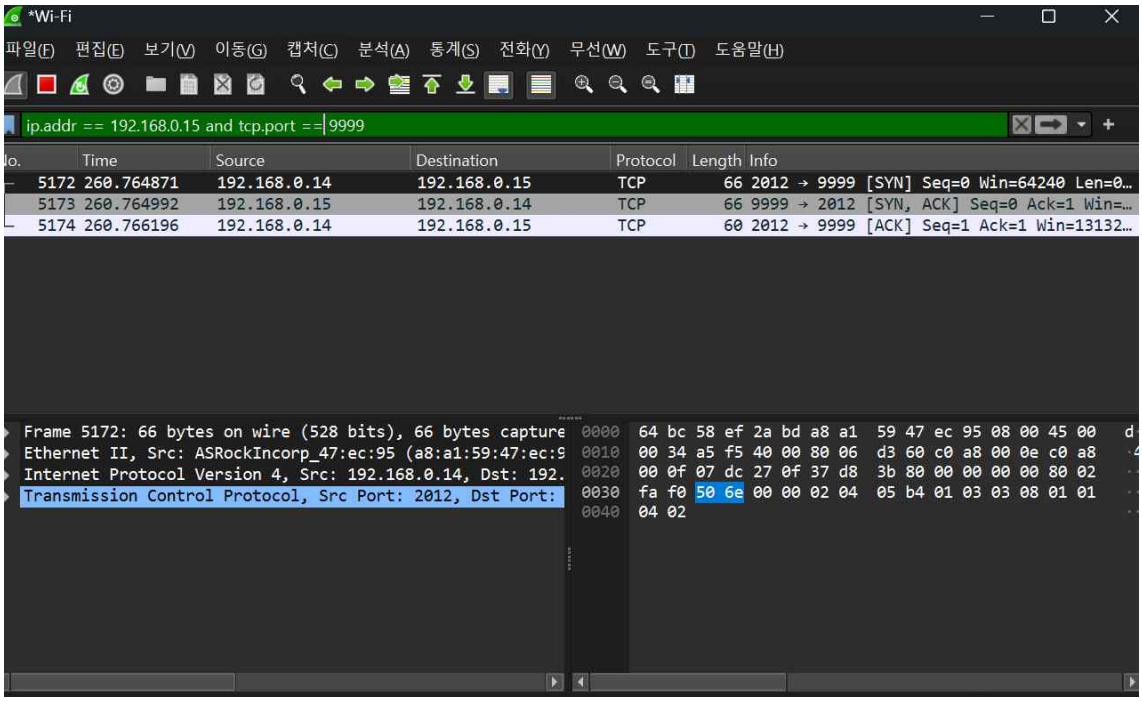


1. server와 client에서 각각 socket 객체를 생성한다.
2. server socket은 자신의 ip, port 정보를 통해 bind한다.
3. server는 listen 함수로 client의 connect를 기다린다.
4. client가 connect를 하고 server에서 accept한다.
5. 각각의 while문 안에서 우선 client에서 유저의 input을 받고 makeRequestMessage 함수를 통해 request message를 생성한 뒤 server에 전송한다.
6. server는 request를 receive하고 makeResponseMessage 함수를 통해 response message를 생성하여 client에 응답한다. client에서는 이를 받고 앞의 과정을 반복한다.

7. 통신이 끝나면 server와 client 둘 다 close() 함수를 호출하여 종료한다.

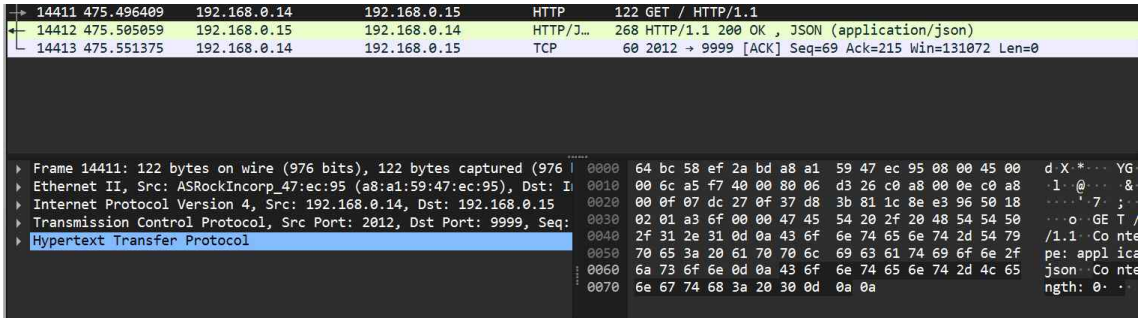
-Wireshark로 본 tcp 통신

Server ip: 192.168.0.15
Client ip: 192.168.0.14



1. server.py와 client.py를 실행한 직후의 캡처. 3-way shake를 진행했다는 것을 확인할 수 있다. 먼저 client에서 server 측으로 SYN bit가 1인 request를 보내고 server는 SYN bit와 ACK bit를 1로 설정하여 응답한다. 다시 client에서 ACK bit를 1로 설정하여 server에 message를 보낸다. server와 client 간의 connection이 형성되었다.

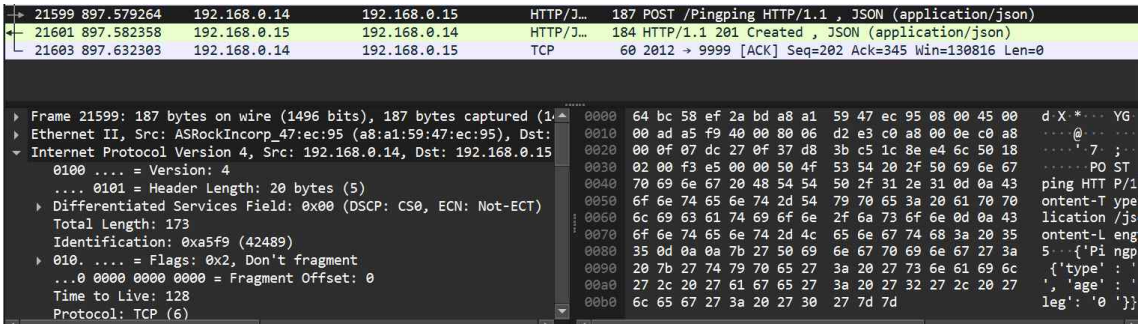
2. get all 명령어 이후의 캡처

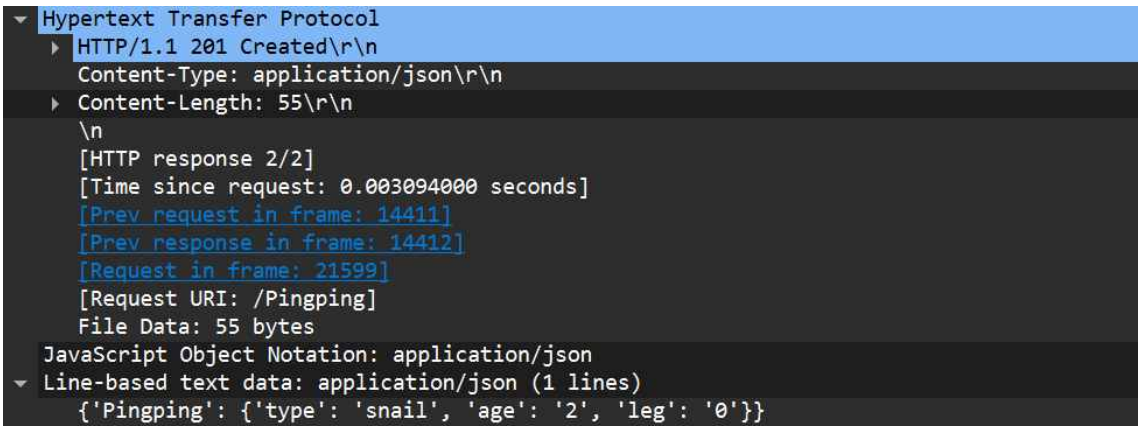


client에서 request, server에서 response를 주고 받는 것을 확인할 수 있으며 client가 server의 response를 받은 후에는 message를 잘 받았다고 ACK를 보내는 것을 확인할 수 있다.

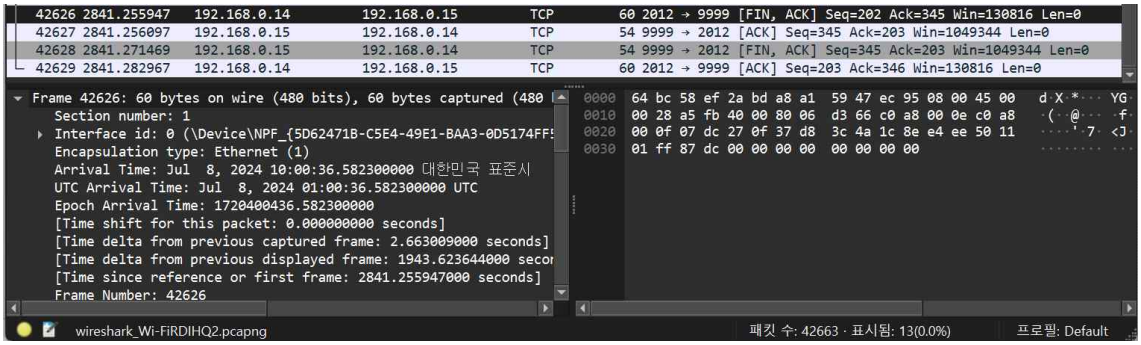
3. 명령어 post Pingping

snail 2 0를 입력했을 때 캡처





wireshark에서 response message를 분석하여 보여준다. 의도한 대로 message가 잘 전송되고 있다는 것을 확인 할 수 있다.



4. q를 눌러 종료할 때의 캡처

4-way handshake을 진행하는 것을 확인할 수 있다. 처음 client가 종료를 요청하면서 FIN과 ACK bit를 보내고 server에서는 ACK bit로 응답한다. 다시 server에서 종료할 준비가 되었다고 FIN bit를 보내면 client에서는 ACK bit로 응답한 후에 connection이 종료된다.

-소스코드 설명

server.py

```
# IPv4 주소를 지정하고, TCP를 사용한다
serverSocket = socket(AF_INET, SOCK_STREAM)
# server를 설정한 ip와 port binding 한다
serverSocket.bind((host, port))
# clinet의 요청을 대기한다
serverSocket.listen(1)
print("Server: listening...")

# client가 접속됐을 경우 새로운 socket과 client의 주소를 return값으로 받는다.
connectionSocket, clientAddress = serverSocket.accept()
```

server에서는 우선 socket 객체를 만드는 것에서 시작한다. 이때 매개변수에 전달한 AF_INET, SOCKSTREAM은 각각 IPv4를 사용하겠다는 것과 TCP 통신을 하겠다는 것을 의미한다. server는 자신의 ip 주소, port 주소에 binding 한다. 그 후 listen 함수를 통해 client의 접속을 기다린다.

client.py

```
# IPv4 주소를 사용하고 TCP 통신을 한다.
clientSocket = socket(AF_INET, SOCK_STREAM)
# server의 ip, port 주소로 접속한다
clientSocket.connect((ip, port))
```

client에서도 마찬가지로 socket 객체를 우선 생성한다. connect 함수를 통해 입력한 ip와 port가 가리키는 server에 접근을 시도한다.

server.py while문

```
# request-response를 주고 받는 while문
while True:

    # client로부터 request를 받는다
    data = connectionSocket.recv(1024)
    requestMessage = data.decode("utf-8")

    print("-----------REQUEST-----------\n")
    print(requestMessage)
    print("-----------------------------\n")

    # request 메시지를 해석하고 response를 보낸다
    try:
        connectionSocket.send(makeResponseMessage(requestMessage).encode("utf-8"))
    except:
        break


# server socket을 종료한다
serverSocket.close()
```

connection이 되면 본격적으로 client와 message를 주고 받는 while 문이 실행된다. server에서는 우선 client가 보내는 request를 recv 함수로 받고 이를 출력한다. 또 makeResponseMessage(request) 함수를 통해 받은 request에 대한 response message를 생성한 후 client에 send 한다. while 문을 벗어나게 되면 close() 함수를 호출하여 server를 종료한다.

client.py while문

```
while True:

    # 사용자로부터 input을 받는다
    print("Enter: [method] [objectName]")
    userInput = sys.stdin.readline().strip()

    # q를 입력할 경우 while문 실행 종료
    if (userInput == "q"):
        break

    # request message를 생성한다
    requestMessage = makeRequestMessage(userInput)
    # 잘못된 input 처리
    if (requestMessage == -1):
        print("Wrong Input: try again")
        continue

    # request message를 server에 전송한다
    clientSocket.send(requestMessage.encode("utf-8"))

    # server로부터 response message를 받는다
    data = clientSocket.recv(1024)
    responseMessage = data.decode("utf-8")
```

client의 while문에선 사용자의 입력을 바탕으로 request message를 생성해야 하기 때문에 우선 사용자에게 입력을 받는다. 입력이 잘못 되었을 경우 적절히 처리하고 만약 사용자가 q를 입력하였을 경우 socket 통신을 종료한다는 뜻으로 이해하고 while문을 종료한다. 사용자에게 받은 입력은 makeRequestMessage(input) 함수에 전달되어 request message이 생성된다. 그 후에는 생성된 request를 server에 전송한 후 server로부터 response가 올 때까지 대기한다.

```
    print("----------RESPONSE-----------\n")
    print(responseMessage)
    print("-----------------------------\n")


# clinet를 종료한다
clientSocket.close()
```

response가 오면 메시지를 출력한 후 이 과정을 반복한다. while문이 종료될

경우 close() 함수를 통해 client를 종료한다.