컴퓨터 네트워크

- 04. Chatting -

제 출 일	2018.10.04
과제번호	04
학 과	컴퓨터공학과
학 번	201402391
이 름	이 병 만

※ 과제 해결 방법

※ Thread를 이용한 group chat 구현

- chat_client.py -

```
import socket
import sys
import threading
```

➔ 구현에 필요한 socket, sys, threading 모듈을 import한다.

```
def recv_message(message_socket):
    while True:
        message = message_socket.recv(1024)
        print(message.decode())
        if message[7:11].lower() == 'quit':
            break

    print('Host Disconnected')
```

➔ 받은 메시지를 뿌려주는 함수이다. recv(1024)로 데이터를 받아서 message에 저장한다. 저장한 message를 출력해준다.

```
def main():
    if len(sys.argv) != 4:
        print('python groupchatclient.py [IPADDRESS] [PORTNUMBER] [Client ID]')
        sys.exit()

    ip_address = sys.argv[1]
    port_number = int(sys.argv[2])
    client_id = sys.argv[3]

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((ip_address, port_number))

    print('Host connected')

    client_socket.send(client_id.encode())

    threading.Thread(target = recv_message, args = (client_socket,)).start()

    while True:
        message = input('')
        send_msg = '[' + client_id + ']' +  message
        client_socket.send(send_msg.encode())

        if message[0:4].lower() == 'quit':
            break

    print('Host Disconnected')
    client_socket.close()

if __name__=="__main__":
    main()
```

부분

① 인자를 IPADDR, PORTNUM, ID 순으로 입력한다.
② 받은 인자를 변수로 저장한다.
③ socket을 생성하고 ip_주소와 port_번호로 연결한다.
④ 연결을 한 후 'Host connected'를 출력한다.
⑤ client_id를 인코딩 후 전송한다.
⑥ Thread를 생성한다.

부분

① 채팅창을 입력하기 위해서 반복문을 돌려준다.
② 메시지를 입력한다.
③ 입력한 메시지를 보낸다.
④ 반복문이 실행이 안되면 'Host disconnected'를 출력한다.
⑤ socket을 닫는다.

- chat_server.py -

```
import socket
import threading
import sys

clnt_list = []
```

➔ 구현에 필요한 socket, sys, threading 모듈을 import하고, 연결된 client를 저장할 리스트를 선언한다.

```
def recv_msg(client_socket):
    global clnt_list
    while True:
        message = client_socket.recv(1024).decode()
        print(message) # 서버에 각 사용자들의 채팅을 출력한다.

        # 현재 접속해있는 사용자들에게 메시지를 뿌려준다.
        for i in range(len(clnt_list)):
            if clnt_list[i] is not client_socket:
                clnt_list[i].send(message.encode())

        if message[9:13].lower() == 'quit':
            break
    print('Client Disconnected')
```

➔ 받은 메시지를 뿌려주는 함수이다. recv(1024)로 데이터를 받아서 message에 저장한다. 저장한 message를 출력해준다.

```python
def main():
    if len(sys.argv) != 2:
        print('python chat server.py [PORTNUMBER]')
        sys.exit()

    port_number = int(sys.argv[1])

    chat_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    chat_server_socket.bind(('', port_number))
    chat_server_socket.listen(5) # 5명을 받아주기 위해 인자 값을 5로지정

    while True:
        (client_socket, addr) = chat_server_socket.accept()
        clnt_list.append(client_socket)

        client_id = client_socket.recv(1024).decode()
        print(client_id + ' Client connect')

        for i in range(len(clnt_list)):
            if clnt_list[i] is not client_socket:
                clnt_list[i].send(('client' + client_id + '- enter').encode())

        threading.Thread(target = recv_msg, args = (client_socket,)).start()

    print('Client Disconnected')
    chat_server_socket.close()

if __name__ == "__main__":
    main()
```

부분

① 인자로 PORTNUM을 입력해준다. 만약 입력을 안 했으면 그에 대한 에러 메시지를 출력한다.
② 받은 인자를 port_number라고 저장한다.
③ socket을 생성하고 저장한 port_number로 연결한다.
④ client를 5명을 받아주기 위해 listen의 인자값을 5로 지정해준다.

부분

① 서버는 계혹 다른 소켓이 접속해있는지 항상 확인하여야 하므로 while을 통해 대기한다.
② client가 추가될 때마다 List에 client를 추가한다.
③ client로부터 받은 id를 저장하고 해당 client정보를 출력한
⑤ 접속해있는 사용자들에게 client가 입장했다는 것을 뿌려준다.
⑥ Thread를 생성해준다.

※ Python select를 이용한 group chat 구현

- select_client.py -

```
import socket
import select
import sys
```

➔ 구현에 필요한 socket, select, sys 모듈을 import한다. 여기서는 Thread대신에 select를 사용했다.

```
if len(sys.argv) < 3:
    print("Usage : python {0} hostname port".format(sys.argv[0]))
    sys.exit()

HOST = sys.argv[1]
PORT = int(sys.argv[2])

MASTER_SOCK = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
MASTER_SOCK.settimeout(200)
```

① 인자를 IPADDR, PORTNUM 순으로 입력한다.
② 받은 인자를 HOST, PORT라고 저장한다.
③ socket을 생성하고 타임아웃 지정해준다. 해당 시간이 지나면 다시 실행한다.

```
while True:
    # SOCKET_LIST에 입력과 자신의 SOCKET 리스트를 생성한다.
    SOCKET_LIST = [sys.stdin, MASTER_SOCK]
    # thread 대신 select를 이용하여 read,write,error등 준비된 socket을 처리한다.
    READ_SOCKETS, WRITE_SOCKETS, ERROR_SOCKETS = select.select(SOCKET_LIST, [], [])

    # READ_SOCKET으로 통신을 한다.
    for sock in READ_SOCKETS:
        if sock == MASTER_SOCK:
            data = sock.recv(4096) # 받은 데이터를 저장한다.
            if not data: # 데이터가 아니라면
                print('\nDisconnected from chat server')
                sys.exit() # 연결을 끊는다.
            else:  # 데이터이면 출력한다.
                print(data.decode(), end="")
        else:
            msg = sys.stdin.readline() # 줄바꿈을 문자를 제거하기 위해서 end파라미터를 설정한다.
            print("\x1b[1A" + "\x1b[2K", end="")
            MASTER_SOCK.sendall(msg.encode()) # 모든 데이터를 송신한다.
```

① SOCKET_LIST를 생성해서 입력과 자신의 socket을 넣어준다.
② 이번 과제에서는 select를 사용하는 것이기 때문에 select함수를 사용해서 준비된 socket을 처리한다.
③ READ_SOCKET으로 통신을 하는데 MASTER_SOCK이면 받은 데이터를 저장한다. 받은 값이 데이터가 아니면 에러메시지를 출력해주고 연결을 끊는다. 데이터가 맞으면 받은 데이터를 출력해준다. 여기서 end 파라미터의 역할은 줄바꿈 문자를 제거하기 위해서 사용하는 것으로 알고 있다.
④ MASTER_SOCK이 아닌 들어온 유저가 메시지를 입력하면 입력한 값의 한 줄을 읽어 msg의 값을 출력해준다.

- select_server.py -

```
import socket
import select
```

➔ 구현에 필요한 socket, select 모듈을 import한다. 여기서는 Thread대신에 select를 사용했다.

```
def broadcast_data(message):
    """ Sends a message to all sockets in the connection list. """
    # 서버를 제외한 모든 client에게 메세지를 뿌려준다.
    for sock in CONNECTION_LIST:
        if sock != SERVER_SOCKET:
            try:
                sock.sendall(message)  # 모든 데이터를 송신한다.
            except Exception as msg:  # 에러 처리
                print(type(msg).__name__)
                sock.close()
                try:
                    CONNECTION_LIST.remove(sock)
                except ValueError as msg:
                    print("{}:{}".format(type(msg).__name__, msg))
```

➔ 받은 메시지를 뿌려주는 함수이다. 연결된 client들에 대해서 메시지를 뿌려줄 것이다. 만약 서버 소켓이 아니라면 try-except를 실행해준다. 받은 모든 데이터를 송신하고 만약 에러가 생긴다면 처리해준다. 그 후 리스트에서 소켓을 제거해준다.

```
CONNECTION_LIST = [] # 연결된 client를 저장할 리스트를 선언한다.
RECV_BUFFER = 4096
PORT = 1234 # 포트를 지정해준다.

SERVER_SOCKET = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# SO_REUSEADDR 옵션으로 기존의 바인딩된 주소를 다시 사용할 수 있게 한다.
SERVER_SOCKET.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
SERVER_SOCKET.bind(("", PORT))  #

input_listen = input('연결 가능한 Client 개수 : ')
print("Listening...")
SERVER_SOCKET.listen(int(input_listen))  # xx개의 socket을 기다린다.

CONNECTION_LIST.append(SERVER_SOCKET)
print("Server started!")
```

① 연결된 Client를 저장할 리스트를 선언한다.

② 입력가능한 버퍼와 포트를 지정해준다.

③ socket을 생성하고 보통 소켓을 사용하는 프로그램은 강제 종료되었지만, 커널단에서 해당 소켓을 바인딩해서 사용하고 있기 때문에 발생하는 에러가 있는데 SO_REUSEADDR의 옵션을 사용해서 기존의 바인딩된 주소를 다시 사용할 수 있게 한다.

④ 연결 가능한 client 개수를 입력받는다.

⑤ 연결된 client를 리스트에 추가한다.

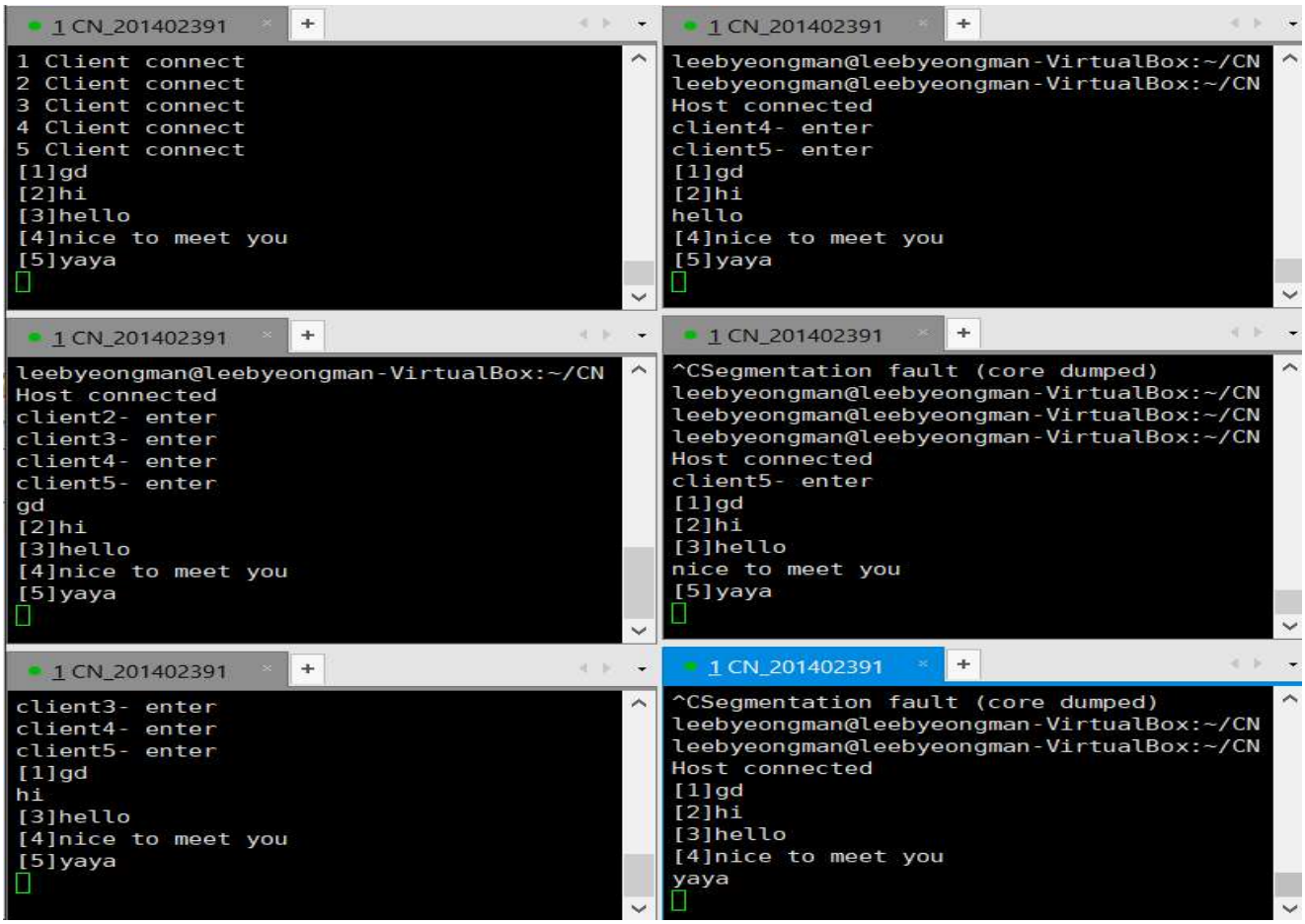
```
while True:
    # Select 함수를 사용하는 부분이다. Read_Socket으로 COnnection_List를 사용하였고
    # 나머지 Write_socket, Error_Socket은 빈 리스트로 놓았다.
    READ_SOCKETS, WRITE_SOCKETS, ERROR_SOCKETS = select.select(CONNECTION_LIST, [], [])

    for SOCK in READ_SOCKETS:  # 새로운 접속이 생길 경우
        # SERVER_SOCKET은 listen받는 소켓이다.
        if SOCK == SERVER_SOCKET:
            SOCKFD, ADDR = SERVER_SOCKET.accept()
            CONNECTION_LIST.append(SOCKFD)  # 연결된 소켓을 CONNECTION_LIST에 추가한다.
            # 사용자가 들어왔다는 메시지 출력과 broadcast를 통해 send한다.
            print("\rClient ({0}, {1}) connected".format(ADDR[0], ADDR[1]))
            broadcast_data("Client ({0}:{1}) entered room\n"
                           .format(ADDR[0], ADDR[1]).encode())
        else:  # SERVER_SOCKET이 아닌 입력이 들어온 경우
            try:
                DATA = SOCK.recv(RECV_BUFFER)  # client로 부터 데이터를 받는다.
                if DATA:
                    ADDR = SOCK.getpeername()  # socket의 client 사용자 정보를 얻는다.
                    message = "\r[{}:{}]: {}".format(
                        ADDR[0], ADDR[1], DATA.decode())
                    print(message, end="")
                    broadcast_data(message.encode())
            except Exception as msg: # client가 disconnect일 경우
                print(type(msg).__name__, msg) # 메시지를 출력해준다.
                print("\rClient ({0}, {1}) disconnected.".format(
                    ADDR[0], ADDR[1]))
                broadcast_data("\rClient ({0}, {1}) is offline\n"
                               .format(ADDR[0], ADDR[1]).encode())
                SOCK.close()
                try:
                    CONNECTION_LIST.remove(SOCK) # CONNECTION_LIST에서 삭제한다.
                except ValueError as msg:
                    print("{}:{}.".format(type(msg).__name__, msg))
                continue

SERVER_SOCKET.close()
```

➔ 다음 반복문은 select함수를 사용하는 부분이다. SERVER_SOCKET인지 아닌지의 여부에 따라 처리를 해준다. 새로운 접속이 생긴다면 Thread에서 했던 방식과 비슷하게 accept를 해서 연결을 하고 리스트에 접속한 socket을 추가해준다. SERVER_SOCKET이 아닌 입력이 들어온다면 client로부터 받은 데이터를 출력해준다.
연결이 되지 않은 경우에는 해당 예외 메시지를 출력해주고 broadcast_data함수를 사용해서 client가 offline이라는 메시지를 전달해주고 socket을 닫는다.

※ Must add reference link

https://github.com/dnutiu/python-networking/tree/master/chat

※ Thread를 이용한 group chat 구현 (실행 결과)



※ Python select를 이용한 group chat 구현 (실행 결과)