컴퓨터 네트워크

HW 1 socket programming

2010-12343
환경재료과학전공
박상문

Problem statement

이번 과제는 socket API 를 이용해 작은 채팅 어플리케이션을 구현하는 것이었습니다.
저는 Python 을 이용하여 채팅 어플리케이션을 만들어보았습니다. Python socket API 는
C 의 socket API 를 wrapping 한 것으로 기본 인터페이스는 C 와 거의 흡사합니다.
Socket(), bind(), listen() 등의 method 를 사용해서 server socket 을 열고 client 에서는
Connect() method 를 통해서 서버와 연결합니다.

소켓의 비동기적 작업 처리를 위해 Select library 를 사용했습니다. IPC 나 쓰레딩이 일반적인데,
비동기적으로 쉬운 방법이라고 생각되어 select 를 사용해보았습니다. Select 는 서버 – client 를
연결하는 소켓과, client 의 Standard-In 을 바라보면서 변화를 감지하고 작동합니다.

Implementation

socket 은 serializable 한 데이터만 전달합니다. 반면 파이썬 내부 object 들은 serializable 하지 않은
경우가 많습니다. 특히 String 을 통해 message 를 전달하여야 하기 때문에 serialization 작업이
필요합니다. encode() method 를 사용해서 이를 해결하였습니다. 받는 쪽에서는 받은 데이터를
decode() method 를 사용해 python String 으로 복구하였습니다.

메세지는 다양한 정보를 담고 있습니다. String 만으로 이 정보를 표현하기 어려워, JSON 을 이용해
메세지를 표현하기로 하였습니다.

```
{
  "type": type,
  "content": content
}
```

의 양식으로 메세지의 프로토콜을 정의 합니다.
Type 으로는 "login", "logout", "message", "invitation", "leaveRoom" 을 두었습니다.
Python 에서 Json 은 dictionary 타입이므로,
sender 에서는 json.dumps()를 이용해 메세지를 String 으로 만들어 encode()하고,
receiver 에서는 decode()된 String 을 json.loads()를 이용해 다시 객체화 시켜 정보를 읽어 옵니다.

chat_client.py 와 chat_server.py 두 파일로 구성됩니다.

chat_client.py
parameter 로 host 주소와 port number 를 받습니다.
그리고 바로 server 와 socket 연결을 준비합니다.

```
    # connect to remote host
    try:
        s.connect((host, port))
    except:
        print('Unable to connect.')
        sys.exit()
```

select library 를 이용해 소켓에 접근합니다.

```
    while 1:
        socket_list = [sys.stdin, s]

        # Get the list sockets which are readable
        read_sockets, write_sockets, error_sockets = select.select(
            socket_list, [], [])
```

이 경우 소켓은 서버와의 연결소켓과 stdin 2 개가 있습니다.
서버 연결 소켓의 경우 서버에서 보내온 메세지가 있다는 뜻이고,
stdin 의 경우 client input 으로 자신이 메세지를 쓴 경우 입니다.

서버와 연결 소켓에서는 데이터의 type 을 체크해서 해당하는 작업을 진행합니다.
"Login" 은 로그인이 되었는지 판단하고, "invitation"은 나에게 날라온 초대를 받을지 말지 정합니다.
나머지는 그냥 메세지가 온 경우로 화면에 데이터를 써줍니다.

Server 와 client 모두에서 message_form 이라는 wrapper 함수로 메세지를 규정합니다.

```
def message_form(types, content):
    return json.dumps({"type": types, "content": content}).encode('utf-8')
```

서버로 메세지를 보낼 시에는 기본 메세지 전송과 예약어 사용이 있습니다.
"!invite" "!logout" "!leaveRoom" 가 예약어로 되어 있습니다.
Client 에서 메세지를 전송할 때마다 위의 예약어인지 판단하여 작업을 처리합니다.

```
# user entered a message
else:
    msg = sys.stdin.readline()
    m = re.search(r'^!invite[\s]*([a-zA-Z])', msg,)
    mm = re.search(r'^!logout', msg,)
    mmm = re.search(r'^!leaveRoom', msg,)
```

chat_server.py

서버는 연결 상태를 저장하는 몇 가지 저장소가 있습니다.

```
    # List to keep track of socket descriptors
    CONNECTION_LIST = []
    # dict to keep track chatting member
    CHAT_MEMBER_LIST = {"A": [], "B": []}
    # set to keep track invitation
    WAIT_INVITE_SET = set()
    # dict to keep track login state
    LOGIN_MAP = {}
```

CONNECTION_LIST 에서는 클라이언트와 연결된 소켓을 담당합니다.

CHAT_MEMBER_LIST 는 현재 방에 등록된 유저와 해당 유저에게 가야할 메세지 큐가 있고,

WAIT_INVITE_SET 은 해당 유저가 로그인 하면 날라갈 초대 메세지가 들어있습니다.

LOGIN_MAP 에는 현재 로그인한 유저들의 state 가 들어있습니다.

채팅방에 있는 유저 모두에게 보내야할 메세지는 broadcast_data() 함수를 사용합니다.

```
def broadcast_data(sock, message):
    # Do not send the message to master socket and the client who has
    # send us the message
    for _socket in CONNECTION_LIST:
        if (_socket != server_socket and
                _socket != sock and
                LOGIN_MAP.get(sock, None) in CHAT_MEMBER_LIST.keys() and
                LOGIN_MAP.get(_socket, None) in CHAT_MEMBER_LIST.keys()):
            try:
                # _socket.send(message.encode('utf-8'))
                _socket.send(message_form("message", message))
            except:
                # broken socket connection may be, chat client pressed
                # ctrl+c for example
                _socket.close()
                CONNECTION_LIST.remove(_socket)
    for member, queue in list(CHAT_MEMBER_LIST.items()):
        if(member not in LOGIN_MAP.values()):
            queue.append(message_form("message", message))
```

메세지를 보낸 사람과 서버 자신을 제외하고 나머지 클라이언트에게 메세지를 전달합니다.

만약 한 유저가 offline 상태라면 CHAT_MEMBER_LIST 의 해당 유저 큐에 메세지를 저장해놓습니다.

Client manual

로컬을 기준으로 설명합니다.

서버가 돌아가고 있어야 합니다. 서버의 기본 포트는 20343 으로 되어있습니다.

>> python3 chat_server.py

서버 주소와 포트번호를 넣어 클라이언트를 실행시킵니다.

>> python3 chat_client.py 127.0.0.1 20343

유저 정보는 다음과 같습니다. 꼭 맞게 입력해주세요.

```
USER_DB = [
    {"ID": "A", "PASSWORD": "123"}, {"ID": "B", "PASSWORD": "123"},
    {"ID": "C", "PASSWORD": "123"}, {"ID": "D", "PASSWORD": "123"}
]
```

기본적으로 A, B 유저는 방에 등록되어 있다.
C 나 D 를 초대하고 싶은 경우
>> !invite C
>>!invite D
를 입력하여 메세지를 보낼 수 있다.

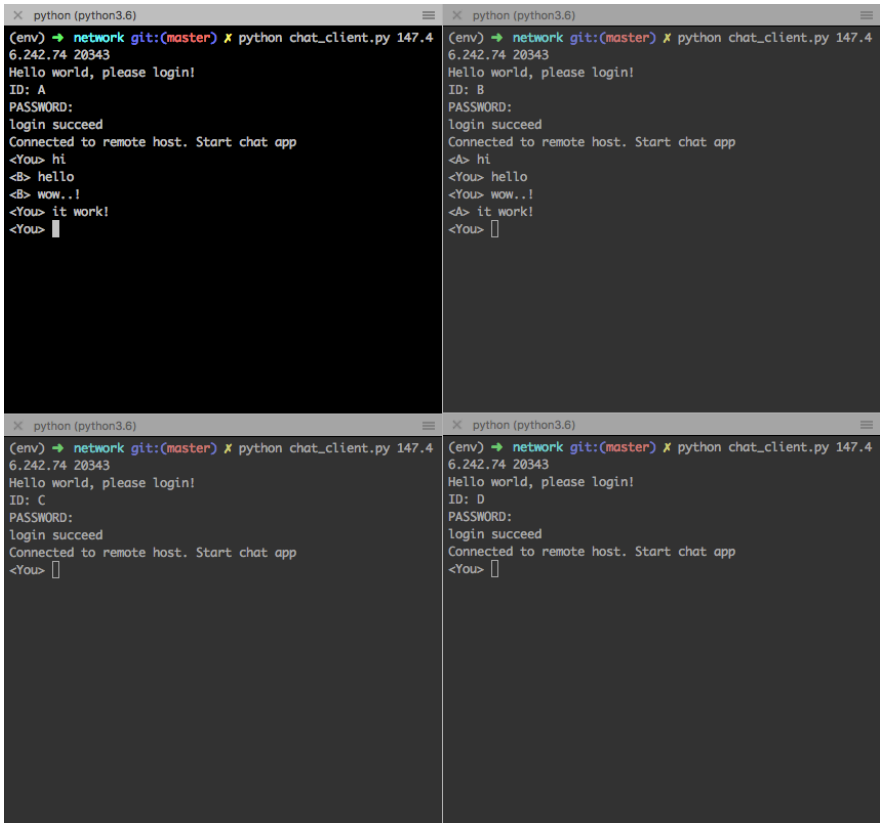
C 나 D 가 online 상태라면 바로 메세지를 확인하고 Y or N 로 승인할 수 있고,
Offline 상태라면 로그인 했을 때 해당 메세지를 보고 판단할 수 있다.

>>!logout
을 통해 다른 계정으로 들어올 수 있고,

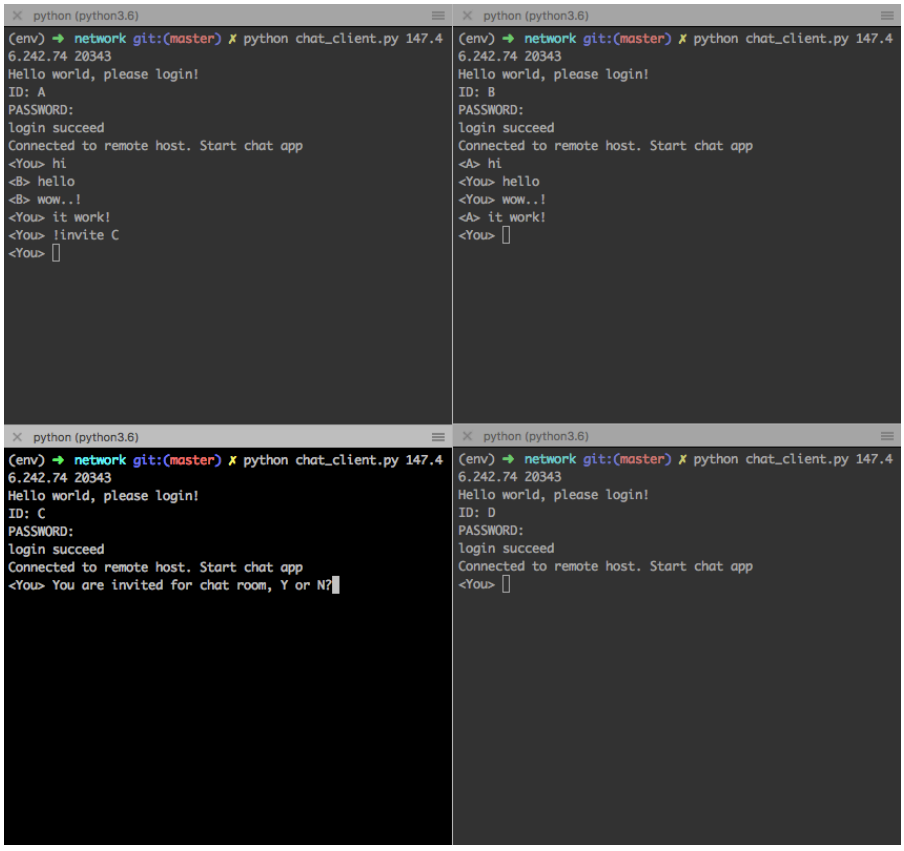
>>!leaveRoom
명령어를 통해 방에서 나갈 수 있다.
최소한 1 명의 유저는 방에 남아있어야 한다.

Result

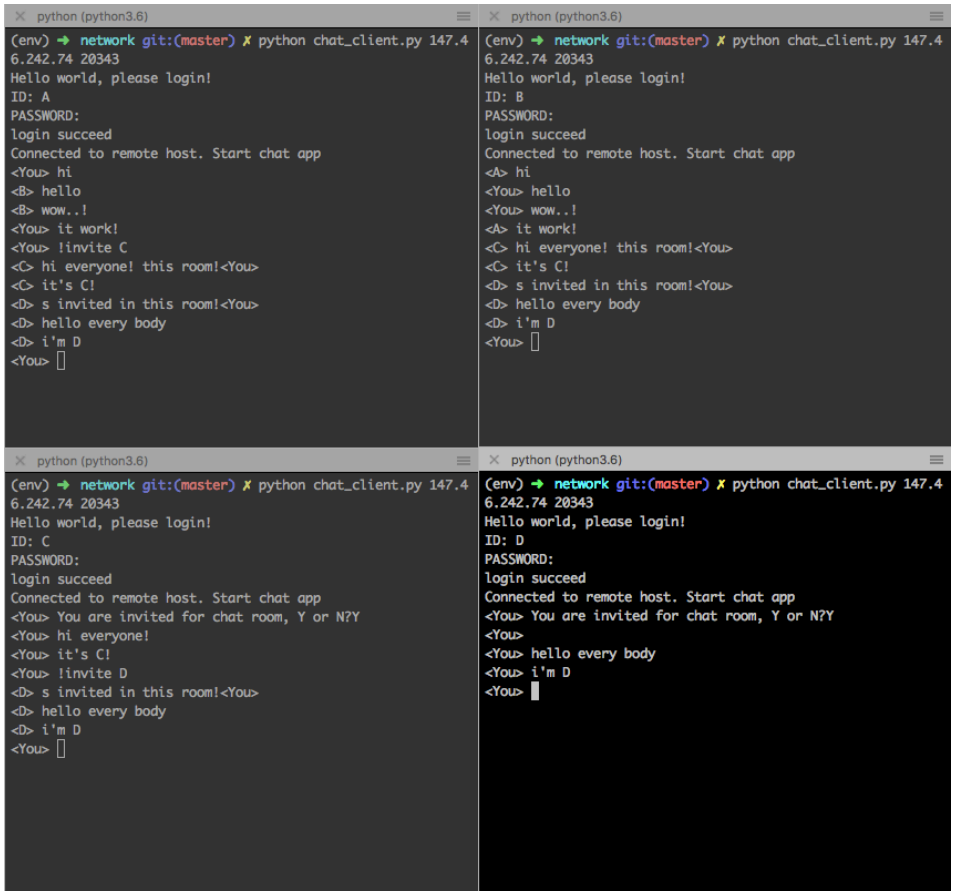
(A) (B)
(C) (D)
의 사진상 순서로 로그인 되어 있다.



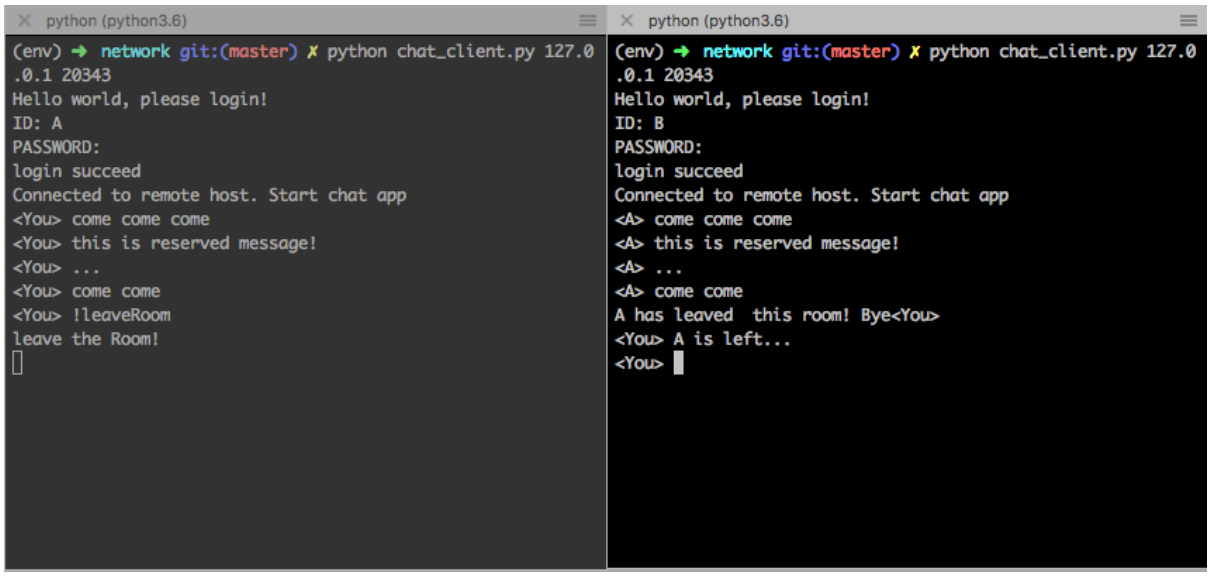
(1) A 와 B 는 방에서 잘 대화를 나누고 있다. C 와 D 는 대기방 상태이다.



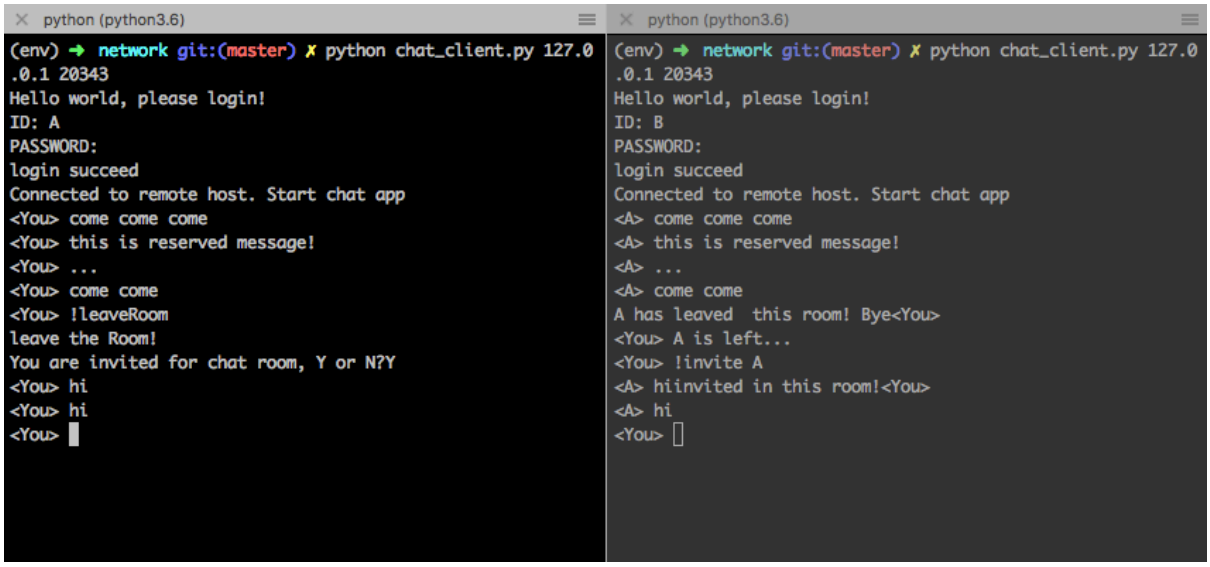
(2) A 가 C 에게 초대 메세지를 보냈다.



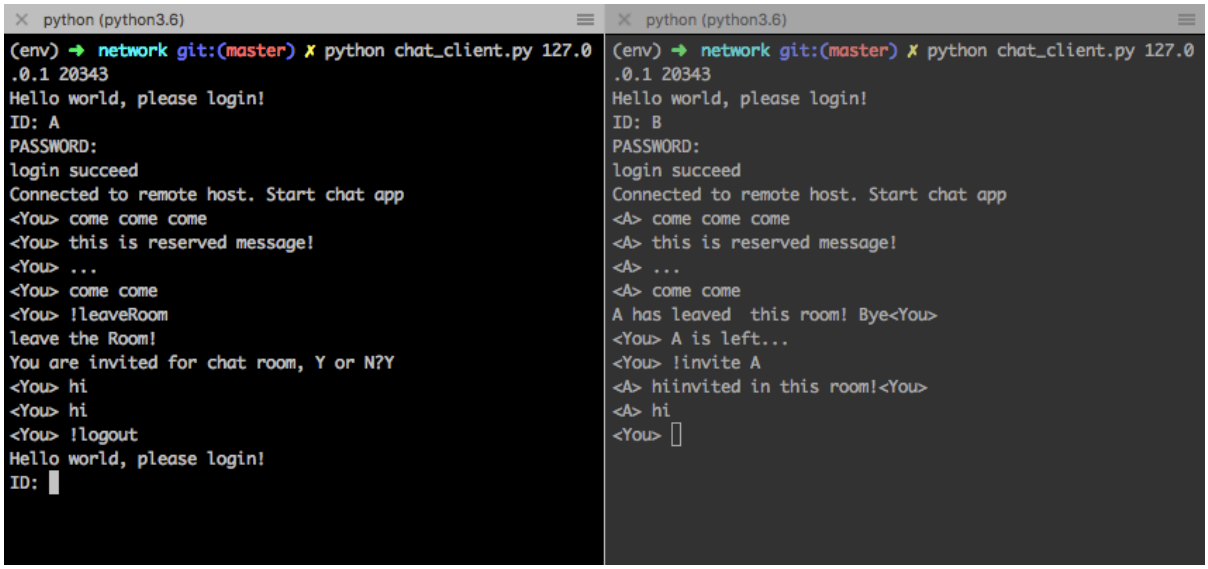
(3) C 와 D 가 모두 방에 초대된 모습. 서로 통신이 됨을 볼 수 있다.



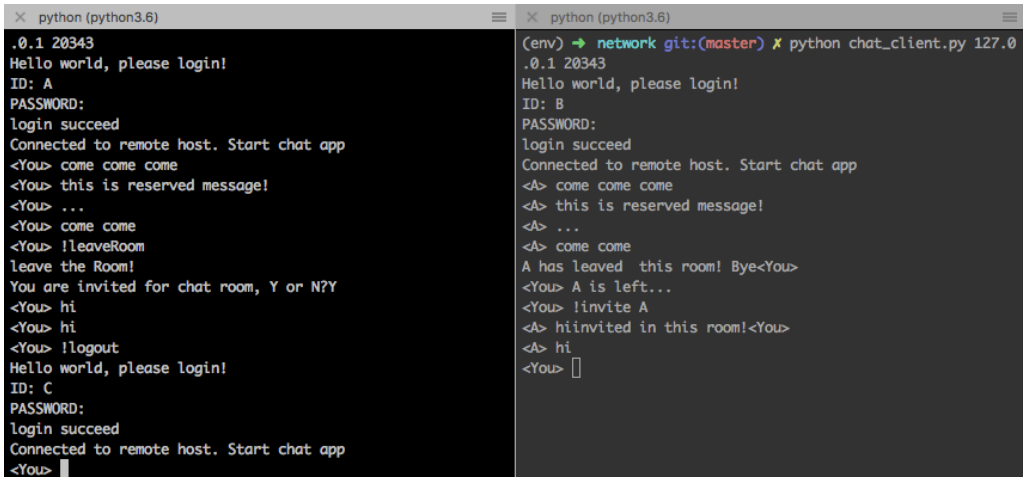
(4) A 가 방을 나갔다. B 혼자 방에 있는 상태



(5) B 가 다시 A 를 초대하여 방에 들어왔다.



(6) A 가 로그아웃 하였다.



(7) C 로 로그인 하였다. 정상적으로 들어옴을 알 수 있다.

Discussion

기본적인 스펙을 구현하였다. 다만 에러 케이스에 대한 충분한 처리가 아직 되어 있지 않다.
예를 들어 방에 있는데 초대를 보내는 경우, 방에 없는데 방 나가기를 한다 던지,
하는 경우에 대해서 처리가 되어 있지 않다. 또 ctrl + c 로 클라이언트가 종료한 경우
가끔 비정상적인 작동을 보이기도 한다. 미처 처리 못한 버그들이 있다.

서버에서 IP 와 사용자를 스태틱하게 바인드 하지 않고, 그 때 그 때 매칭 시켜야 해서 처음에
힘들었다. 이로 인해 방 정보, 로그인 정보 등을 독립적으로 운영해야 해서
CONNECTION_LIST, CHAT_MEMBER_LIST, WAIT_INVITE_SET ,LOGIN_MAP 의 여러
저장소에서 각각 관리하게 하였다.

Python 을 활용해서 스트링이나 여러 자료구조의 활용 면에서 쉽게 접근이 가능하였다.
하지만 자료가 적고, 특히 select 는 예제도 거의 없어서 힘들었다.
C 의 경우 따로 serialization 을 하지 않아도 되는데 python 은 encode()를 추가로 해주어야
한다는 것에서 막혀서 오래 걸렸다.

초대 메세지는 따로 WAIT_INVITE_SET 에 저장하게 구현하였다. 이는 CHAT_MEMBER_LIST 에는
방의 멤버에 대한 정보만 있기 때문에 넣을 수 없기 때문이다.. 하지만 결국 해당 유저에 대한
메세지라는 점 을 생각하면 방 등록 여부와 상관없이 해당 유저에 대한 메세지를 한 곳에 모아 놓는
것도 고려할 만 하다.