Network Report

2017313260 이재민

1. Development environments

OS: windows 10, VMWare Workstation 15 Player

Programming Language: python3

Version: python 3.7.4

ISO : ubuntu-20.04-live-server-amd64

2. 코드 설명

event는 register, unregister, offline으로, register은 client가 접속했을 때, unregister은 @exit로 client를 종료했을 때, offline은 client를 Ctrl-C 로 종료했을 때, timeout이 나는 경우이다.

localIP는 해당 client의 local IP 주소이고, clientAddr는 IP:Port 형태의 주소이다.

client에서 server로 보내는 메시지 형태

'client'_event_clientID_localIP:10081

server에서 client로 보내는 메시지 형태

'server'_event_clientID_clientAddr

<<client.py>>

1. getLocalIP()

구글의 공개 DNS 서버에 연결해서 localIP를 얻어낸다.

2. class serverSocket

A. class 내부 변수

clientID: client를 실행할 때 입력해준 ClientID

serverIP: client를 실행할 때 입력해준 serverIP

localSocket: 같은 NAT에 있으면 사용할 bind할 socket

globalSocket: socket

localIP: client의 local IP이다.

regClientList: 접속되어있는 client들의 주소값을 갖고 있는 dictionary이다.

online: client의 상태, thread들을 관리하기 위한 일종의 lock

B. makeMsg(event)

client에서 server로 보낼 msg를 만드는 함수. 위에서 설명한 틀을 맞춘다.

C. parseMsg(msg)

전송받은 msg를 parsing해서 event에 맞게 함수를 실행시킨다.

'server'가 msg에 있는지 확인해서 어디서 온 msg인지 확인한다.

만약 server에서 받았는데, 그 중에서

'register'을 받았다면, 서버에 연결된 것이므로 regClientList에 ID와 주소를 추가한다.

'unregister'을 받았다면, 연결을 끊은 것이므로 regClientList에서 해당 client를 제거한다. 그런데 여기서 clientID가 본인의 ID와 같으면 본인이 unregister을 요청한 것 이므로, login을 False로 바꾸고, socket.close()를 요청한다.

'offline'을 받았다면 연결이 끊긴 것이므로, regClientList에서 해당 client를 제거한다.

server에서 받은 게 아니라면 채팅 메시지 이므로 내용을 출력한다.

D. unregister()

@exit를 입력했을 때, 실행되는 함수로, unregister event를 전송해야 한다. msg에 unregister event를 담아서 server로 전송한 뒤에, online을 False로 바꾼다.

E. isSameNAT(IP_1, IP_2)

동일 NAT에 있는지 확인하는 함수이다. IP에서 맨 뒤를 떼고 비교하면 된다.

F. sendChat

채팅 메시지를 전송하는 함수이다. msg 형태는 clientID_ChatMsg이다. 누가 보냈는지 clientID로 나타내고, chatMsg에 보내고 싶은 메시지를 담는다. 같은 NAT상에 있으면 localSocket을 이용해 전송하고, 아니라면 globalSocket을 이용해 전송한다.

```
recvClientID = chatMsg[0]
msg = self.clientID + '_' + ''.join(chatMsg[1:])

recvClientAddr = self.regClientList[recvClientID].split(':')
recvClientIP = recvClientAddr[0]
recvClientPort = recvClientAddr[1]
```

이는 미리 입력해 준 '메시지를 받을 클라이언트'의 주소를 얻는 코드이다. (msg줄 제외)

G. runQuery(query)

입력해준 command에 맞춰 함수를 실행한다.

만약 @show_list라면 연결되어 있는 client들을 출력해야 하므로, regClientList에 있는 요소들을 모두 출력한다.

만약 @chat이라면 채팅 메시지를 보내야하므로 sendChat() 함수를 실행한다.

```
elif query[0] == '@chat':
    self.sendChat(query[1:])
```

여기서, sendChat에 변수를 전달할 때, @chat을 떼고 '메시지를 받을 클라이언트'의 ID와 msg만을

전달한다.

만약 @exit라면 unregister하고 프로그램을 종료해야 하므로 unregister() 함수를 실행한다.

H. keetAlive()

client가 실행되어 있다면, 10초마다 Keep Alive 메시지를 보낸다.

I. recvLocal()

online이 true인 동안, localSocket으로부터 메시지를 계속 전달받는다.

J. globalSocket()

client가 연결이 되므로 online을 true로 바꾸고, online이 true인 동안 globalSocket으로부터 메시지를 계속 전달받는다.

3. main함수

clientID와 serverIP를 입력 받고, clientSocket class를 가져온다.

thread로 각각 recvLocal, recvGlobal, keetAlive를 실행시킨다.

```
threadLocal.daemon = True
threadGlobal.daemon = True
threadAlive.daemon = True
```

이는 socket close를 요청 했지만 다른 thread들의 접근으로 프로그램이 종료되지 않는 것을 방지하기 위함이다.

online이 True인동안 command를 입력 받을 수 있게 한다.

<<server.py>>

1. getLocalIP()

구글의 공개 DNS 서버에 연결해서 localIP를 얻어낸다.

2. class serverSocket

A. class 내 변수

regClientLocalList: 각 ClientID에 해당하는 local address를 담는 dictionary

regClientGlobalList: 각 ClientID에 해당하는 global address를 담는 dictionary

regClientTime: 각 ClientID로부터 메시지를 받은 시간을 저장하는 dictinary

B. parseIP(addr)

IP:Port 형태의 주소에서 IP만 return하는 함수

C. sendToMakeAddr(addr):

IP:Port 형태의 주소에서 sendto를 실행하기 위해 tuple형태로 바꾸는 함수

D. makeMsg(event, ClientID, clientAddr)

sever에서 client로 보낼 msg를 만드는 함수. 위에서 설명한 틀을 맞춘다.

E. isSameNAT(addr_1, addr_2)

동일 NAT에 있는지 확인하는 함수이다. IP에서 맨 뒤를 떼고 비교하면 된다. server.py에서는 addr를 저장할 때, IP:Port 형태로 저장해서 client의 inSameNAT와 조금 다르다.

F. sendClientList(ClientID, addr)

새로운 client가 register했을 때, 실행되는 함수로, register된 client에게 연결되어 있는 다른 client들의 list를 줘야한다. 만약 새로 register된 client와 전달하려는 client가 같은 NAT상에 있으면 local address를 전달하고, 그렇지 않다면 global address를 전달한다.

G. recvMsg()

client로부터 전송받은 msg를 parsing해서 event에 맞게 함수를 실행시킨다.

만약 'keep Alive'를 받았다면, 해당하는 clientID의 time을 갱신시킨다. (regClientTime)

'unregister'을 받았다면, 연결을 끊은 것이므로 해당하는 clientID에 sendResponse()로 응답을 보내고, delElement()로 각 dictionary에서 해당 ID를 제거한다.

'register'을 받았다면 연결이 추가된 것이므로, addElement()로 각 dictionary에 해당 ID를추가한다. 그리고 응답과, 연결되어 있는 client들에 대한 정보를 전송한다.

H. sendResponse(ClientID, event)

event를 실행했을 때, client들의 상태를 다른 모든 client들에게 전달하는 함수.

해당 client가 register되었을 때에는 sendClientList로 해당 client에게 이미 정보를 전달했으므로, if문으로 예외처리해준다.

```
    sendAddr = self.regClientGlobalList[key]
    if self.isSameNAT(self.regClientLocalList[clientID], self.regClientLocalList[key]):
        clientAddr = self.regClientLocalList[clientID]
    else:
        clientAddr = self.regClientGlobalList[clientID]
    msg = self.makeMsg(event, clientID, clientAddr)
    self.serverSocket.sendto(msg.encode(), self.sendToMakeAddr(sendAddr))
```

그 외에는 모든 client에게 다 event를 담은 msg를 전송한다.

만약 해당 client와 전달받는 client가 같은 NAT상에 있으면 local address를 msg에 담고, 그렇지 않다면 global address를 msg에 담는다.

I. addElement(clientID, localAddr, globalAddr)

regClientTime, regClientLocalList, regClientGlobalList에 값을 추가한다.

J. delElement(clientID)

regClientTime, regClientLocalList, regClientGlobalList에서 clientID에 해당하는 값을 제거한다.

K. timeOut()

모든 client에 대해 응답이 30초동안 있었는지 확인을 한다. 현재시간에서 regClientTime에 담겨져있는 시간을 빼서 확인한다. thread가 따로 돌아가서 while문이 도는 중에 요소가 빠져서 에러가 날 수 있으므로 try-except로 처리해준다.

2. main함수

serverSocket을 가져오고, socket을 bind해준다.

그리고 serveSocket정보를 출력해준다.

그리고 timeOut, RecvMsg함수를 각각 thread로 돌려준다.

3. Test

각 환경에 client.py, server.py를 넣고 python3 client.py, python3 server.py로 실행시킨다.

server: Host windows10

client1: Host windows10

client2: VMware ubuntu server1

client3: VMware ubuntu server2

Test 순서

1. server를 킨다

2. client1, client2, client3 순서대로 연결

3. client3에서 client2로 [hi im in ubuntu] 메시지 전송

4. client2에서 client3로 [hi me too man] 메시지 전송

5. client2에서 client1로 [how about you] 메시지 전송

6. client1에서 client2로 [im in windows] 메시지 전송

7. client2에서 @exit로 종료

8. client3에서 client1로 [bye] 메시지 전송

9. client3에서 Ctrl-C로 종료

10. client1에서 @exit로 종료

server log

```
Address = ('192.168.0.75', 10080)
client1  register  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client1  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  register  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client2  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  register  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client3  local: 192.168.184.128:10081  global: 192.168.0.75:50973
```

```
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  Keep Alive  local: 192.168.184.129:10081  global: 192.168.0.75:50972
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client2  unregister  local: 192.168.184.129:10081  global: 192.168.0.75:59929
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client3  Keep Alive  local: 192.168.184.128:10081  global: 192.168.0.75:50973
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
```

```
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client3 is offline
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client1  Keep Alive  local: 192.168.0.75:10081  global: 192.168.0.75:50971
client1  unregister  local: 192.168.0.75:10081  global: 192.168.0.75:10081
```

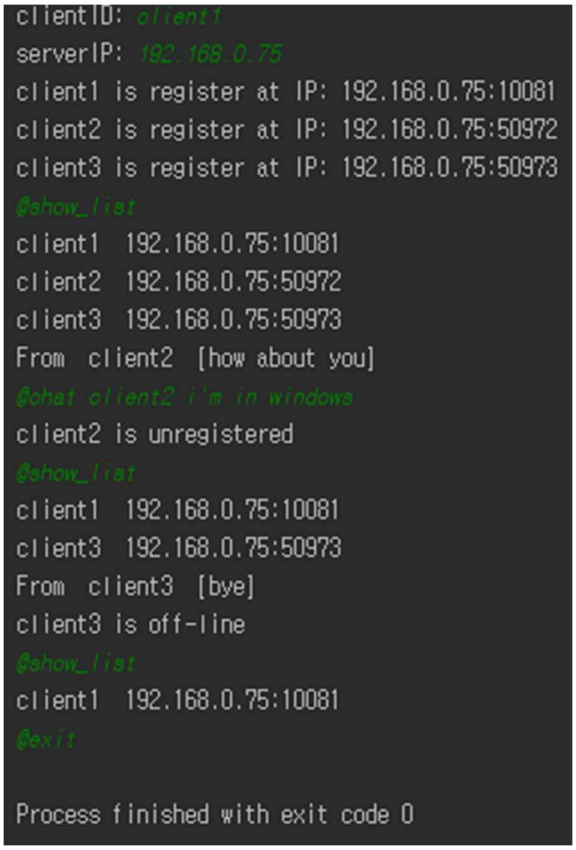
client가 등록될 때 마다 server에 표시된다.

10초마다 각 client에서 Keep Alive 메시지를 받는다.

unregister를 받을 시에 연결을 끊는다.

timeout이 나면 offline을 표시하고 다른 client들에게 이 사실을 보낸다.

client1 log:



각 client가 register될 때 마다 이 사실이 표시된다.

client2, client3이 다른 NAT상에 있어서 global address로 표시된다.

@show_list를 하면 연결된 리스트가 잘 출력된다.

client2로부터 받은 메시지가 잘 출력된다.

client2가 @exit로 꺼진 사실이 잘 알려졌다

@show_list로 연결된 리스트에서 client2가 없어진 걸 확인할 수 있다.

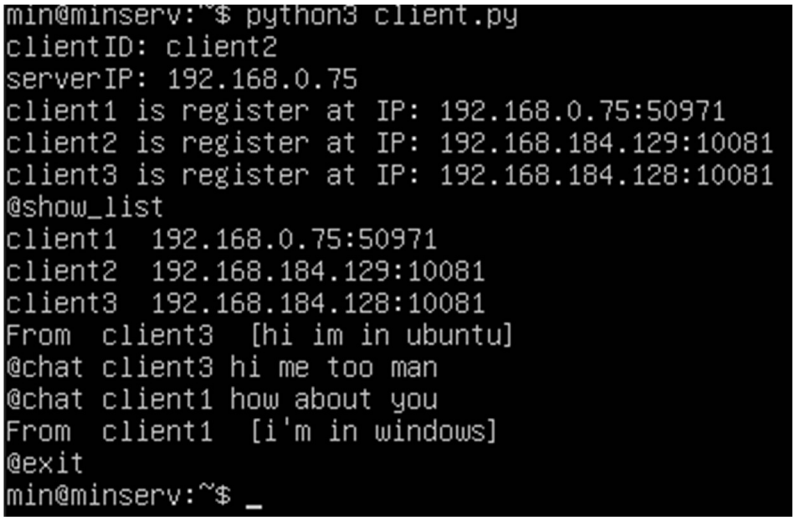
client3으로부터 받은 메시지가 잘 출력된다.

client3이 offline인 사실이 잘 알려졌다

@show_list로 연결된 리스트에서 client3이 없어진 걸 확인할 수 있다.

@exit시에 정상 종료된다.

client2 log



각 client가 register될 때 마다 이 사실이 표시된다.

client2, client3이 같은 NAT상에 있어서 local address로 표시된다.

client1은 다른 NAT상에 있어서 global address로 표시된다.

@show_list를 하면 연결된 리스트가 잘 출력된다.

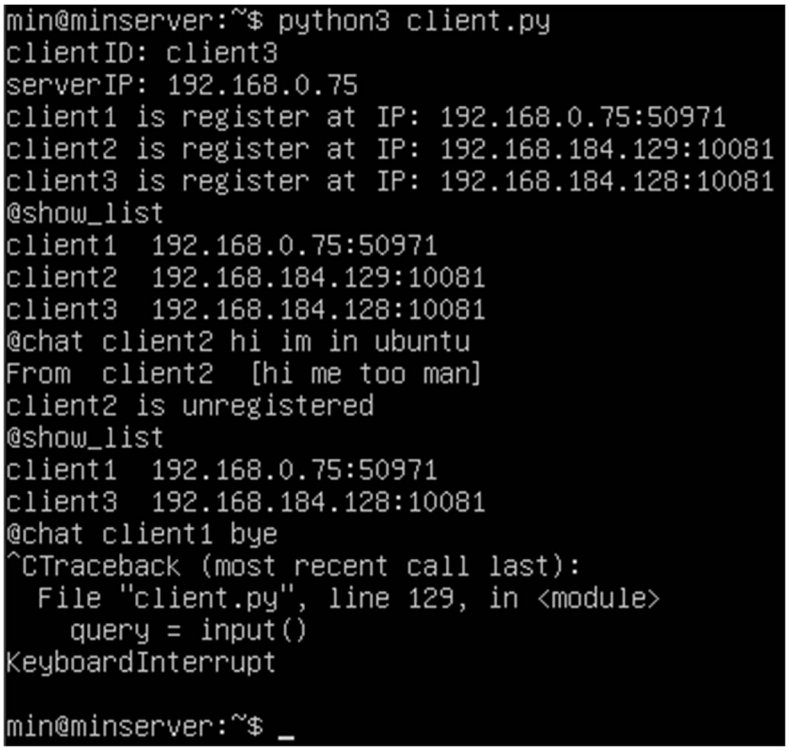
client3로부터 받은 메시지가 잘 출력된다.

client2가 @exit로 꺼진 사실이 잘 알려졌다

client1으로부터 받은 메시지가 잘 출력된다.

@exit시에 정상 종료된다.

client3 log



각 client가 register될 때 마다 이 사실이 표시된다.

client2, client3이 같은 NAT상에 있어서 local address로 표시된다.

client1은 다른 NAT상에 있어서 global address로 표시된다.

@show_list를 하면 연결된 리스트가 잘 출력된다.

client2로부터 받은 메시지가 잘 출력된다.

client2가 @exit로 꺼진 사실이 잘 알려졌다

@show_list로 연결된 리스트에서 client2가 없어진 걸 확인할 수 있다.

Ctrl-C로 종료했다.