

# Assignment #4 Report

2016314726 정영준

## Development Environments

OS: Ubuntu 18.04.5 LTS

Python: 3.6.9

Mininet: 2.3.0d6

## Import packages

sys, socket, threading, time, pickle

## 프로그램설명

Virtualbox에서 Ubuntu os와 mininet을 이용하여 실행하였습니다. server.py와 client.py 2개의 파일에 python 언어를 사용하여 프로그래밍하였습니다. execute\_mn.sh의 실행으로 프로그램이 시작됩니다. server shell에서 server.py, client shell에서 client.py를 python3로 실행합니다.

## server.py

클래스를 사용하여 모든 기능을 처리할 수 있도록 코딩하였습니다. serverPort를 인자로 받아 선언이 가능한 Server클래스는 \_\_init\_\_ 함수에서 앞으로 사용되는 변수들을 초기값으로 설정합니다. Server 클래스 내의 모든 함수의 기능을 설명하겠습니다.

Init\_skt 함수: socket을 시작하는 함수로 self.serverPort 변수를 사용하여 socket을 초기화하며 이를 self.skt 변수에 할당합니다.

Init\_client\_dict 함수: self.client\_dict를 비어있는 dictionary로 초기화하는 함수로 key는 client의 id, value는 client의 ip주소를 나타내는 값으로 후에 사용됩니다.

init\_time\_dict 함수: self.time\_dict를 비어있는 dictionary로 초기화하는 함수로 key는 client의 id, value는 client가 등록된 시각으로 keep alive call을 10초마다 보내어 갱신됩니다.

detect\_disconnect 함수: keep alive call이 주어진 timeout인 30초 동안 오지 않는 경우를 잡아내는 함수로 Thread를 사용하여 시작됩니다.

disconnect\_state 함수: 위 함수에서 Thread의 target으로 주어지는 함수로 while 반복문을 사용하여 disconnect가 발생할 때까지 timeout\_check 함수를 반복 실행합니다.

timeout\_check 함수: 실질적으로 timeout이 되었는지 확인하는 함수로 self.time\_dict에서 value인 최근 client의 갱신 시각이 현 시각과 30초 넘게 차이가 나면 self.client\_dict에서 해당 client를 제거하고 메시지를 출력하며 업데이트된 client 전체의 정보를 전송합니다.

start\_receive 함수: socket의 데이터를 읽는 함수로 Server 클래스 객체가 생성되고 \_\_init\_\_을 다음으로 시작되는 함수로 모든 함수를 내포하고 있는 main 함수의 역할을 합니다.

received\_data\_parsing 함수: socket을 통해 받은 정보를 ':'을 기준으로 나누어 처리하기 쉬운 형태로 바꿔주는 함수입니다.

handle\_received\_data 함수: 위의 parsing을 위한 함수를 통해 처리하기 용이하게 바뀐 전송받은 데이터를 데이터의 내용에 따라 처리하는 함수로 Exit, Update, Register, Show, Keep alive를 각각 E, U, R, S, K로 나타낸 parsing을 통해 얻은 mode라는 변수에 따라 진행 과정이 달라집니다.

## client.py

전역변수로 serverIP, serverPort, clientPort가 각각 10.0.0.3, 10080, 10081을 할당받습니다.

마찬가지로 클래스를 사용하여 모든 함수가 클래스내에 존재합니다. 위의 3개의 전역변수를 인자로 받아 각각 self.serverIP, self.serverPort, self.clientPort로 다시 할당됩니다.

init\_skt 함수: server의 함수와 동일하지만 이번엔 self.clientPort의 값을 사용하여 처리됩니다.

init\_same\_NAT 함수: 추가구현을 위해 같은 NAT내에 위치한 client 목록을 저장하기 위한 리스트를 초기화하는 함수입니다.

Init\_boolean\_var 함수: Boolean 자료형의 변수를 초기화하는 함수로 client의 상태를 나타내는 self.alive를 False로 명령어 입력 가능 상태를 나타내는 self.receiving을 True로 초기화합니다.

Init\_ipconfig 함수: 추가기능 구현을 위한 private ip를 탐색하는 함수입니다. 탐색에 실패하였을 경우 127.0.0.1로 자동 설정됩니다.

get\_id 함수: 처음 실행시 사용자에게 client의 id를 입력받는 함수입니다.

thread\_time\_check 함수: threading.Timer를 활용하여 10초마다 server로 keep alive 신호를 보내는 함수인 keep\_alive 함수를 실행합니다.

keep\_alive 함수: 'K' mode로 client의 id를 server socket으로 전송합니다.

receive 함수: 데이터를 받고 처리하는 함수로 socket을 통해 받은 데이터를 load하고 mode에 맞게 다른 과정을 수행합니다.

thread\_receiving\_start 함수: 위의 receive 함수를 thread를 사용하여 실행시키는 함수입니다.

reigister\_client 함수: 'R' mode (register)로 client의 id와 private ip를 encoding하고 server로 전송하는 함수입니다.

Send\_chat 함수: 채팅을 보내는 함수로 보내고나서 다시 command를 받기위해 self.receiving을 True로 설정합니다.

command\_process 함수: command를 처리하는 함수로 @chat @show\_list @exit 3개의 경우를 if문을 사용하여 알맞은 과정을 수행합니다.

handle\_command 함수: @chat의 경우 추가로 chat을 보낼 client의 id와 chat 내용이 필요하기 때문에 command\_process를 실행하기전 두가지 경우를 나누기 위한 함수입니다.

start\_receive 함수: client 클래스의 객체가 할당된 후 시작되는 함수로 위의 모든 함수를 사용하여 client의 기능을 모두 수행하는 main함수입니다.

## 프로그램 테스트

```
mininet@mininet-VirtualBox:~/prac_4$ sudo ./execute_mn.sh
```

위와 같이 execute\_mn.sh를 실행하여 시작합니다.

"host: pri.1.3"	"host: server"
root@mininet-VirtualBox:~/prac_4# python3 client.py Enter ID : client1 client1 10.0.0.1:58598 █	root@mininet-VirtualBox:~/prac_4# python3 server.py client1 10.0.0.1:58598 █

python3 server.py와 python3 client.py를 입력하여 각각 server와 client 파일을 실행시킵니다.

## 등록

Client의 ID를 입력하면 다음과 같이 client에게는 자신을 포함한 모든 client의 정보를 보여주고 server에는 등록된 client의 정보가 표시됩니다.

"host: pri.2.3"	"host: server"
root@mininet-VirtualBox:~/prac_4# python3 client.py Enter ID : client2 client1 10.0.0.1:58598 client2 10.0.0.2:58100 █	root@mininet-VirtualBox:~/prac_4# python3 server.py client1 10.0.0.1:58598 client2 10.0.0.2:58100 █

위와 같이 새로운 client인 client2가 새로 들어오면 현재 등록된 client인 client1과 client2의 정보가 client2에게 출력되며 server에는 새로 client2의 정보가 출력됩니다.

## SHOW\_LIST

```
"host: pri.1.3"
root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client1
client1 10.0.0.1:58598
@show_list
client1 10.0.0.1:58598
client2 10.0.0.2:58100
```

위와 같이 client2가 새로 등록하고나서 update가 이루어지고 server가 이를 모든 client에게 전송하기 때문에 client1이 @show\_list를 통해 전체 client의 정보를 확인하면 새로 등록된 client2의 정보 또한 출력됩니다.

## CHAT

```

"host: pri.1.2"
root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client1
client1 10.0.0.2:52151
client2 10.0.0.2:54563
@chat client5 nice to meet you
From client5 [I love Network]
[]

root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client5
client1 10.0.0.2:52151
client5 10.0.0.1:58523
From client1 [nice to meet you]
@chat client1 I love Network
[]
```

위와 같이 정상적으로 @chat을 사용하여 채팅 기능을 구현하였습니다.

## SAME NAT (추가구현)

```

if command == "@chat":
    SAME_NAT = True if chat_id in self.same_NAT else False
    mode = 'C'
    packed_dt = pickle.dumps([mode, str(self.client_id) + ':' + str(chat_content)])
    if SAME_NAT:
        dest_ip = self.client_dict[chat_id][0]
        dest_port = self.clientPort
    else:
        dest_ip = self.client_dict[chat_id][1]
        dest_port = self.client_dict[chat_id][2]
    self.send_chat(packed_dt, dest_ip, dest_port)
```

위와 같이 same\_NAT list에 채팅을 보내는 id가 존재할 경우 바로 chat 관련 정보의 전송이 가능하도록 코딩하였습니다. @show\_list를 할 때는 모두 public ip를 display하도록 프로그래밍하였습니다.

## EXIT

```

"host: pri.2.3"
root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client2
client1 10.0.0.1:59638
client2 10.0.0.2:54999
@show_list
client2 10.0.0.2:54999
[]

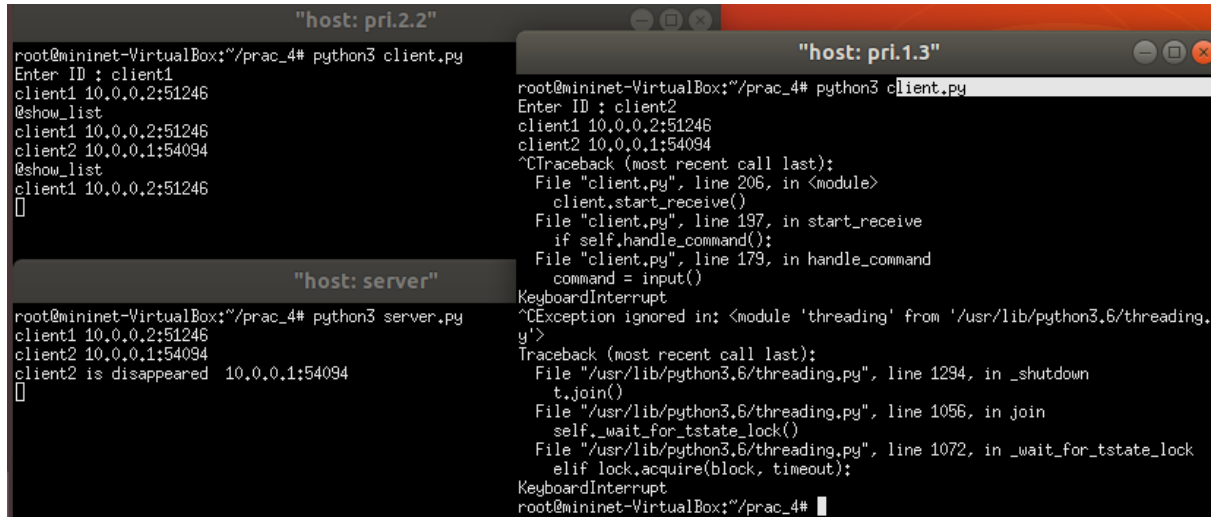
"host: server"
root@mininet-VirtualBox:~/prac_4# python3 server.py
client1 10.0.0.1:59638
client2 10.0.0.2:54999
client1 is deregistered 10.0.0.1:59638
[]

"host: pri.1.3"
root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client1
client1 10.0.0.1:59638
@exit
Successfully Terminated
root@mininet-VirtualBox:~/prac_4#
```

Client1이 @exit를 입력하여 종료를 하게되면 server에 deregistered 메시지가 출력되며

Client2가 @show\_list를 통해 전체 client의 정보를 받아오면 client1의 정보가 지워진 것을 알 수 있습니다.

## Disappear



```

"host: pri.2.2"
root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client1
client1 10.0.0.2:51246
@show_list
client1 10.0.0.2:51246
client2 10.0.0.1:54094
@show_list
client1 10.0.0.2:51246
[]

"host: pri.1.3"
root@mininet-VirtualBox:~/prac_4# python3 client.py
Enter ID : client2
client1 10.0.0.2:51246
client2 10.0.0.1:54094
^CTraceback (most recent call last):
  File "client.py", line 206, in <module>
    client.start_receive()
  File "client.py", line 197, in start_receive
    if self.handle_command():
  File "client.py", line 179, in handle_command
    command = input()
KeyboardInterrupt
^CException ignored in: <module 'threading' from '/usr/lib/python3.6/threading.py'>
Traceback (most recent call last):
  File "/usr/lib/python3.6/threading.py", line 1294, in _shutdown
    t.join()
  File "/usr/lib/python3.6/threading.py", line 1056, in join
    self._wait_for_tstate_lock()
  File "/usr/lib/python3.6/threading.py", line 1072, in _wait_for_tstate_lock
    elif lock.acquire(block, timeout):
KeyboardInterrupt
root@mininet-VirtualBox:~/prac_4#

"host: server"
root@mininet-VirtualBox:~/prac_4# python3 server.py
client1 10.0.0.2:51246
client2 10.0.0.1:54094
client2 is disappeared 10.0.0.1:54094
[]
```

위와 같이 client2가 CTRL+C를 커맨드 입력으로 잘못된 command를 입력하여 멈춘 client를 종료할 경우 30초가 지나면 server에 disappeared 메시지가 나타나며 client1이 @show\_list를 통해 정보를 확인하면 registration list에서 삭제가 된 것을 정상적으로 확인하였습니다.