

Project2

2016147556 김성하

1. Introduction/reference

Operating system: Ubuntu 20.04.1 LTS

Programming language: python3.7.9

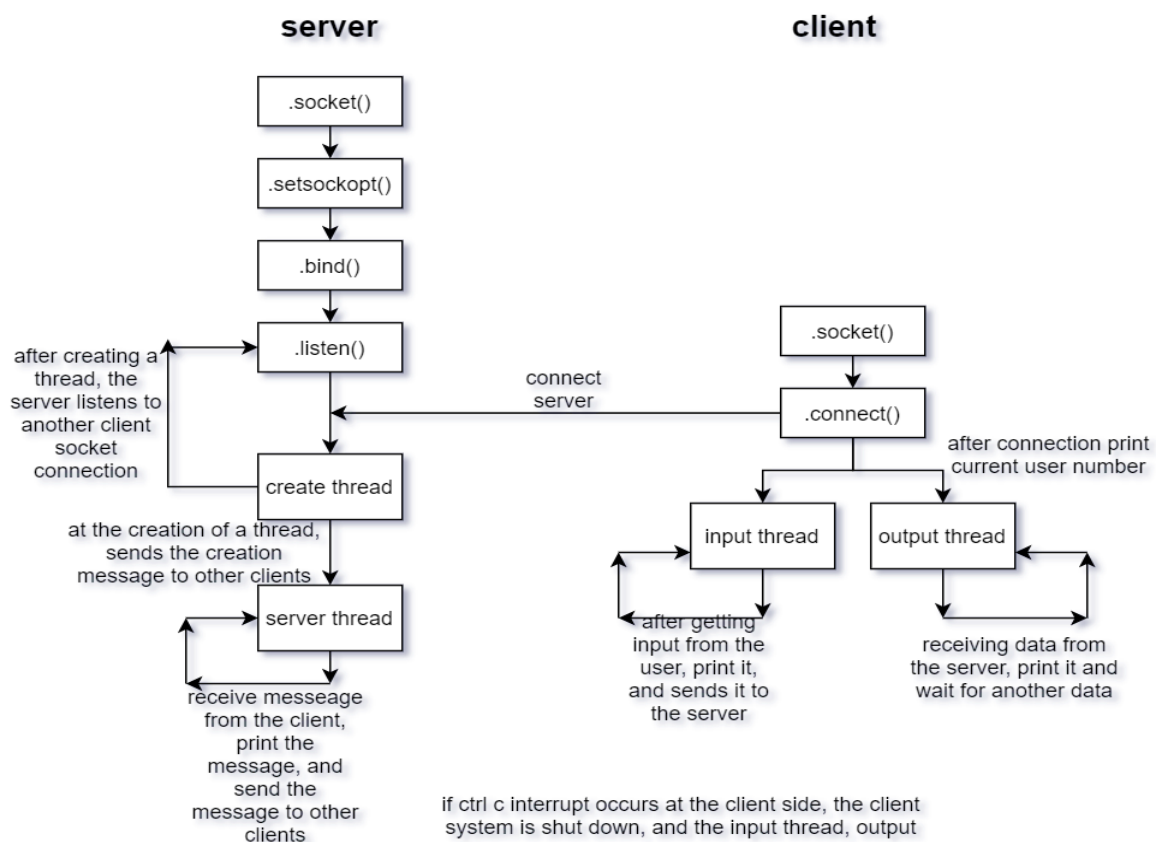
Reference:

Socket-programming-<https://realpython.com/python-sockets/>

Python multithreading시 주의점- <https://coding-groot.tistory.com/103>

Socket functions- <https://docs.python.org/ko/3/howto/sockets.html>

2. Flowchart of a program



if ctrl c interrupt occurs at the client side, the client system is shut down, and the input thread, output thread is also shut down, the client_socket is closed. Then, the server thread corresponding to it detects the client socket close, and sends the leaving message to other clients. then the server thread is also shut down.

Server 소켓에 대한 설정을 하고 client 의 접속을 기다립니다. Client 에서 소켓을 설정하고 server 쪽에 접속을 성공한 순간, server 쪽에서는 sever thread 하나, 그리고 클라이언트 쪽에서는 output thread, input thread 두개를 생성해서 실행합니다.

이때 server thread 가 실행되는 순간, 하나의 유저가 접속에 성공한 것이기 때문에 이미 접속되어 있는 client들에게 생성 메시지를 보냅니다. 그리고 해당 client 쪽에서 오는 데이터를 기다립니다. 만약 데이터가 도착하면, 출력하고, 이를 현재 접속하고 있는 다른 .client 들에게 보냅니다.

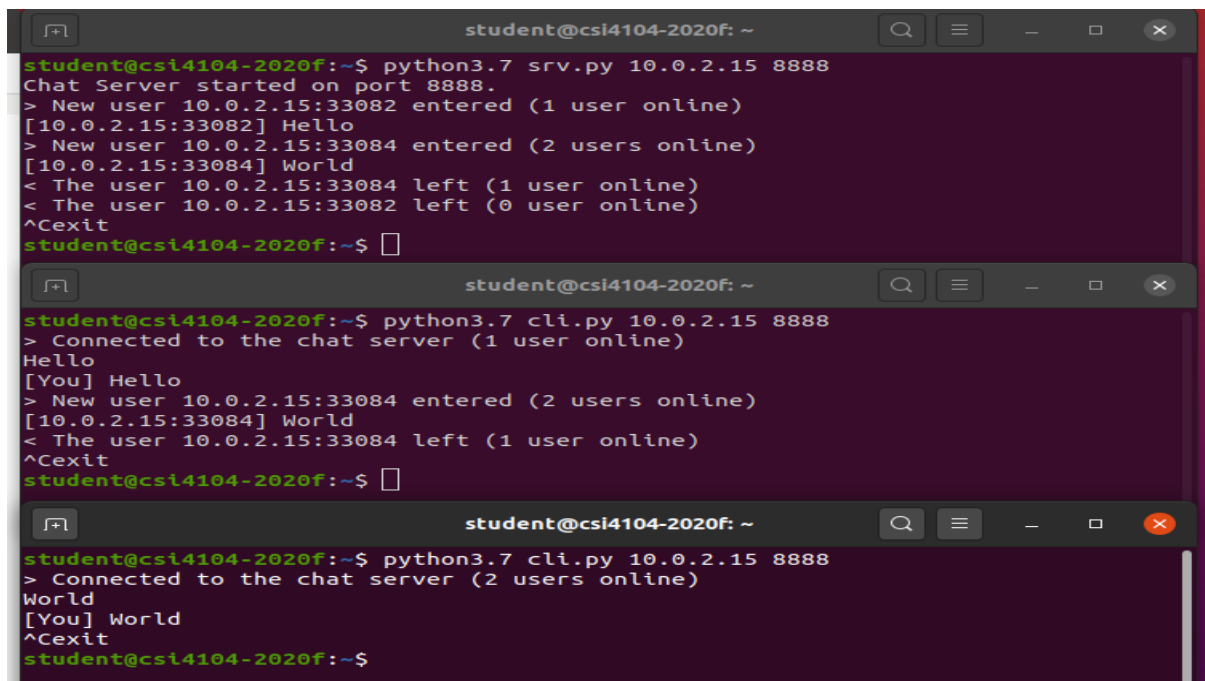
client 사이드의 input thread에서는 유저로부터 입력을 받고 이를 서버에 보냅니다.

Client 사이드의 output thread에서는 server 로부터 다른 유저의 접속정보, 다른 유저의 메시지, 그리고 다른 유저의 연결 해제 상태 메시지를 입력 받아 출력합니다.

3개의 thread 가 무한루프를 도는데, 만약 client 쪽에서 ctrl c 로 접속을 해제하는 순간, client 쪽 두개의 thread, 그리고 client system 이 종료되며, 이에 대응하는 server thread 에서는 이를 감지해 다른 client 들에게 연결 해제 상태 메시지를 보내고, 종료합니다.

추가적으로 client쪽에서 연결이 된 순간 서버로부터 현재 접속 인원 정보를 받아 출력합니다.

3. snap shots of code execution



The image displays three terminal windows from a user named 'student' on a machine 'csi4104-2020f'. The first window shows the server being started with 'python3.7 srv.py 10.0.2.15 8888'. It logs two users connecting (10.0.2.15:33082 and 10.0.2.15:33084), their messages ('Hello' and 'World'), and their disconnections. The second window shows a client running 'python3.7 cli.py 10.0.2.15 8888', which receives the server's messages and logs the user count. The third window shows another client instance where a user sends 'World' and the server responds with the current user count (2 users online).

```
student@csi4104-2020f: ~  
student@csi4104-2020f:~$ python3.7 srv.py 10.0.2.15 8888  
Chat Server started on port 8888.  
> New user 10.0.2.15:33082 entered (1 user online)  
[10.0.2.15:33082] Hello  
> New user 10.0.2.15:33084 entered (2 users online)  
[10.0.2.15:33084] World  
< The user 10.0.2.15:33084 left (1 user online)  
< The user 10.0.2.15:33082 left (0 user online)  
^Cexit  
student@csi4104-2020f:~$  
  
student@csi4104-2020f:~$ python3.7 cli.py 10.0.2.15 8888  
> Connected to the chat server (1 user online)  
Hello  
[You] Hello  
> New user 10.0.2.15:33084 entered (2 users online)  
[10.0.2.15:33084] World  
< The user 10.0.2.15:33084 left (1 user online)  
^Cexit  
student@csi4104-2020f:~$  
  
student@csi4104-2020f:~$ python3.7 cli.py 10.0.2.15 8888  
> Connected to the chat server (2 users online)  
World  
[You] World  
^Cexit  
student@csi4104-2020f:~$
```

Example in the project 2 pdf (user1 enter)->(user1 'Hello')->(user2 enter)->(user2 World)->(user2 disconnect)->(user1 disconnect)->(server close)

The image displays four terminal windows, each titled 'student@csi4104-2020f: ~', showing the execution of a chat server and its clients. The first window shows the server starting on port 8888 and handling three users (10.0.2.15:33088, 10.0.2.15:33090, 10.0.2.15:33092) who exchange messages and then disconnect. The second window shows a client (cli.py) connecting to the server and sending the same messages. The third window shows another client (cli.py) connecting and sending messages. The fourth window shows a third client (cli.py) connecting and sending messages. The server logs show the sequence of users entering and leaving the chat.

```
student@csi4104-2020f:~$ python3.7 srv.py 10.0.2.15 8888
Chat Server started on port 8888.
> New user 10.0.2.15:33088 entered (1 user online)
[10.0.2.15:33088] Hi
[10.0.2.15:33088] my
[10.0.2.15:33088] name
[10.0.2.15:33088] is
[10.0.2.15:33088] sungha
> New user 10.0.2.15:33090 entered (2 users online)
> New user 10.0.2.15:33092 entered (3 users online)
[10.0.2.15:33092] im user3
[10.0.2.15:33090] im user2
< The user 10.0.2.15:33090 left (2 users online)
< The user 10.0.2.15:33088 left (1 user online)
< The user 10.0.2.15:33092 left (0 user online)
^Cexit

student@csi4104-2020f:~$ python3.7 cli.py 10.0.2.15 8888
> Connected to the chat server (1 user online)
Hi
[You] Hi
my
[You] my
name
[You] name
is
[You] is
sungha
[You] sungha
> New user 10.0.2.15:33090 entered (2 users online)
> New user 10.0.2.15:33092 entered (3 users online)
[10.0.2.15:33092] im user3
[10.0.2.15:33090] im user2
< The user 10.0.2.15:33090 left (2 users online)
^Cexit

student@csi4104-2020f:~$ python3.7 cli.py 10.0.2.15 8888
> Connected to the chat server (2 users online)
> New user 10.0.2.15:33092 entered (3 users online)
[10.0.2.15:33092] im user3
im user2
[You] im user2
^Cexit

student@csi4104-2020f:~$ python3.7 cli.py 10.0.2.15 8888
> Connected to the chat server (3 users online)
im user3
[You] im user3
[10.0.2.15:33090] im user2
< The user 10.0.2.15:33090 left (2 users online)
< The user 10.0.2.15:33088 left (1 user online)
^Cexit
student@csi4104-2020f:~$
```

User1 enter-> user1 type message ->user2 enter->user3 enter->user3 type message->user2 type message->user2 disconnect->user1 disconnect->user3 disconnect

The image shows three terminal windows stacked vertically, all with the title 'student@csi4104-2020f: ~'.
The top window shows the execution of 'python3.7 srv.py 10.0.2.15 1234'. The output indicates the chat server started on port 1234, received two users (10.0.2.15:49204 and 10.0.2.15:49206), processed their messages ('Hello' and 'World'), and then both users exited.
The middle window shows the execution of 'python3.7 cli.py 10.0.2.15 1234'. The output shows the client connected to the chat server (1 user online), sent 'Hello', and then exited.
The bottom window shows the execution of 'python3.7 cli.py 10.0.2.15 1234'. The output shows the client connected to the chat server (2 users online), sent 'World', and then exited.

첫번째 예시와 똑 같은 상황에서 다른 포트를 사용해봤습니다.

4. logical explanation

Srv.py

```
1 import socket
2 import threading
3 import sys
4 import signal
5
6 #setting for socket
7 host=sys.argv[1]
8 port=int(sys.argv[2])
9 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
11 server_socket.bind((host,port))
12 server_socket.listen()
13 client_num=0
14 client_socket_list=[]
15 exist_list=[]
16
17 #handling control c interruption
18 def control_c(sig,frame):
19     print('exit')
20     server_socket.close()
21     sys.exit(0)
22
23 signal.signal(signal.SIGINT,control_c)
```

소켓에 대한 설정을 하고, control c interruption 을 위한 control_c 함수를 정의합니다. control_c 함수는 control c interruption 이 일어났을 때의 동작을 명시합니다. Exist_list 에는 index 에 따른

client접속이 끊겼는지 아닌지를 저장합니다. Client_socket_list 에는 현재 접속하고 있는 client 의 소켓이 저장됩니다. Client_num은 client 가 접속할때마다 1이 증가되는 변수로, 일종의 index 역할을 해줍니다.

```
#returns activating ser_threading
def current_alive():
    global exist_list
    num=0
    for idx in range(len(exist_list)):
        if exist_list[idx]!=0:
            num=num+1

    if num>1: return str(str(num)+' users')
    else: return str(str(num)+' user')
```

Current_alive 함수는 현재 접속하고 있는 client의 수를 반환합니다. 이때 2명이상이면 users, 1명이거나 없으면 user 를 뒤에 붙여서 반환합니다.

```
#thread to deal with each client_socket
def server_threading(client_socket,addr,index):
    #prints the current client, send current client connected, and send connection message to other clients
    print('> New user %s:%d entered (%s online)'%(addr[0],addr[1],current_alive()))
    start_msg='> Connected to the chat server ('+str(current_alive())+' online)'
    client_socket.send(start_msg.encode())

    for idx in range(len(client_socket_list)):
        if exist_list[idx]!=0 and idx!=index:
            data1='> New user '+addr[0]+'-'+str(addr[1])+' entered ('+str(current_alive())+' online)'
            client_socket_list[idx].send(data1.encode())

    #waits for data from the socket
    while True:
        try:
            data=client_socket.recv(1024)
            #if client_socket is closed, it sends empty data
            if not data:
                break
            print('%s:%d'%(addr[0],addr[1]),data.decode())
            for idx in range(len(client_socket_list)):
                if exist_list[idx]!=0 and idx!=index:
                    msg_to_client='['+addr[0]+'-'+str(addr[1])+' left ('+str(current_alive())+' online)'+
                    client_socket_list[idx].send(msg_to_client.encode())
        except:
            break
    #if the connection is closed, send the disconnection message to other clients
    exist_list[index]=0
    print('< The user %s:%d left (%s online)'%(addr[0],addr[1],current_alive()))
    for idx in range(len(client_socket_list)):
        if exist_list[idx]!=0:
            data2='< The user '+addr[0]+'-'+str(addr[1])+' left ('+str(current_alive())+' online)'
            client_socket_list[idx].send(data2.encode())
    client_socket.close()
```

서버 쪽에서 client의 접속이 이뤄질 때마다 할당되는 thread입니다. 처음에 생성 메시지를 출력하고, client 에게 현재 접속 인원수를 보내줍니다. 그리고 다른 client 들에게도 생성메시지를 보냅니다. 그후에, client 로부터 무한루프로 입력된 메시지를 받아서 출력합니다. 이때 만약 client 쪽에서 control c interruption 이 일어나 종료되면 무한루프가 끝나게 되며, 이때 접속이 끊겼다는 정보를 출력하고, 이 상태 메시지를 다른 client들에게 보내줍니다.

```

print('Chat Server started on port %d'%(port))
while True:
    #for every client connection, run th server_threading
    client_socket,addr=server_socket.accept()
    new_client=threading.Thread(target=server_threading,args=(client_socket,addr,client_num))
    new_client.daemon=True
    client_num=client_num+1
    client_socket_list.append(client_socket)
    exist_list.append(1)
    new_client.start()
#for interruption
signal.pause()

```

무한루프로 client 의 접속이 있을 때마다 thread를 할당해서 실행하고, 그 정보를 저장합니다.

Client.py

```

import socket
import threading
import sys
import signal

#setting socket
host=sys.argv[1]
port=int(sys.argv[2])
client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
client_socket.connect((host,port))

#signal to deal with control c
def control_c(sig,frame):
    print('exit')
    client_socket.close()
    sys.exit(0)

signal.signal(signal.SIGINT,control_c)

```

Client 쪽에서의 소켓 설정과 control_c 가 입력됐을때의 동작을 정의해줍니다.

```

#input thread
def inp():
    while True:
        message=input()
        print('[You]',message)
        client_socket.send(message.encode())

#output thread
def out():
    while True:
        in_data=client_socket.recv(1024)
        print(in_data.decode())

```

Client 쪽에서 실행되는 input, output thread로 입력을 받으면 출력 후 서버에 전송 을하고, 서버에서 데이터를 받으면 출력을 해줍니다.

```
#run input thread and output thread and set it as daemons to terminate when the main thread is
shut down
inp_thr=threading.Thread(target=inp,args=())
out_thr=threading.Thread(target=out,args=())
inp_thr.daemon=True
out_thr.daemon=True
inp_thr.start()
out_thr.start()
inp_thr.join()
out_thr.join()
signal.pause()
```

두개의 thread 를 실행시키는 부분으로, daemon 을 설정해 시스템이 끝나면 두개의 thread도 종료되게 설정해줍니다.

5. Explanation of the functions

- socket(socket.AF_INET, socket.SOCK_STREAM): 소켓 객체를 만드는 함수로, 첫번째 인자는 주소체계로는 IPv4를 사용한다는 것을 나타내고, 두번째 인자는 소켓의 타입으로 Tcp 소켓임을 나타냅니다.
- setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1): 소켓의 옵션 값을 설정하는 함수로, 주소가 이미 사용되고 있는 에러를 피할 수 있습니다.
- bind(("0.0.0.0", 7777)): 소켓을 특정 ip 주소와 port 번호에 연결하는 함수입니다.
- listen(5): 클라이언트의 접속을 허용하는 함수로, 외부 연결을 거부하기 전에 5개 의 연결 요청을 큐에 넣는다는 것을 뜻합니다
- connect((host, port)): 서버에 접속을 시도하는 함수입니다
- accept(): 대기하다가 클라이언트가 접속하면 새로운 소켓을 리턴하는 함수입니다.
- close(): 소켓을 닫는 함수입니다
- Thread(): multithreading 을 할 때, thread 객체를 생성하는 함수입니다

6. Difference between using multithread and the function select()

Multithreading 은 하나의 프로세스에서 병렬적으로 진행되는 다수의 thread를 통해서 수행 능력을 향상시키는 것을 의미합니다. Select 함수는 파일 디스크립터의 변화를 확인하는 함수로, multiplexing 서버를 구현하기 위한 방법입니다. Select 함수가 더 빠르게 작동하지만, blocking method 처리 등을 위한 추가적인 코드를 작성해야 되기 때문에 multithreading 기법을 사용할 때 보다 코드를 작성할 때 더 까다롭습니다. Multithreading 도 thread 간의 소통 등을 위한 코드를 작성해야 하지만, select 함수보다는 덜 까다롭습니다.