Computer Network Project 2

2015147531_서기원

1. Introduction (Software Environment)

1-1. Linux Version

```
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ cat /proc/version
Linux version 5.4.0-48-generic (buildd@lcy01-amd64-010) (gcc version 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
) #52-Ubuntu SMP Thu Sep 10 10:58:49 UTC 2020
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ uname -a
Linux giwon-virtual-machine 5.4.0-48-generic #52-Ubuntu SMP Thu Sep 10 10:58:49 UTC 2020 x86_64 x86
_64 x86_64 GNU/Linux
```

1-2. Ubuntu Version

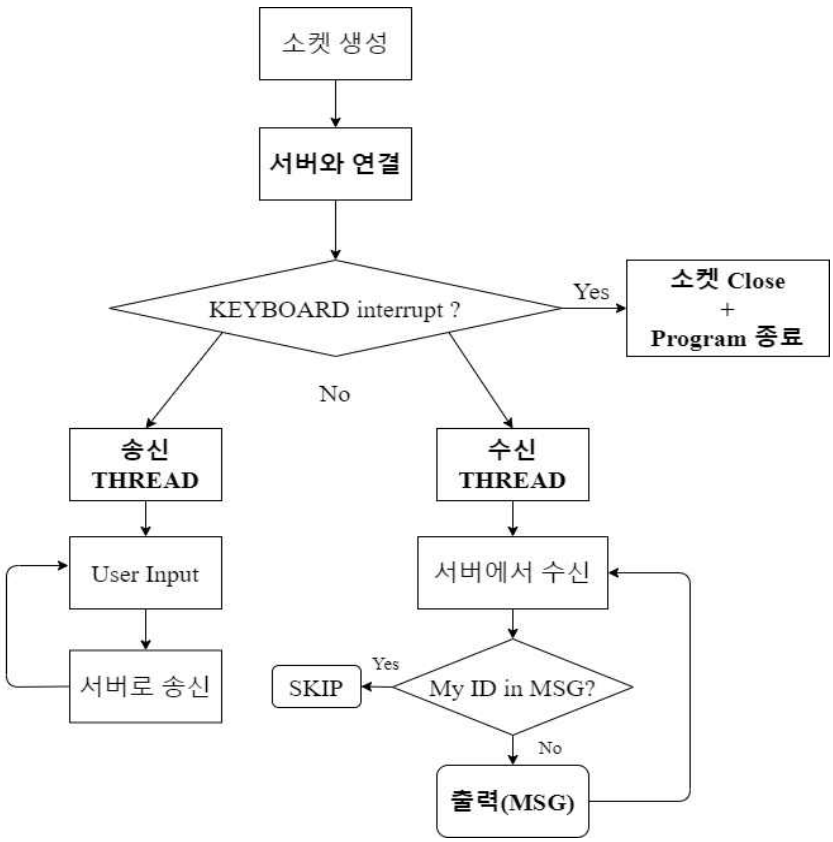
```
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.1 LTS"
```

1-3. Programming Language : PYTHON 3.8.5

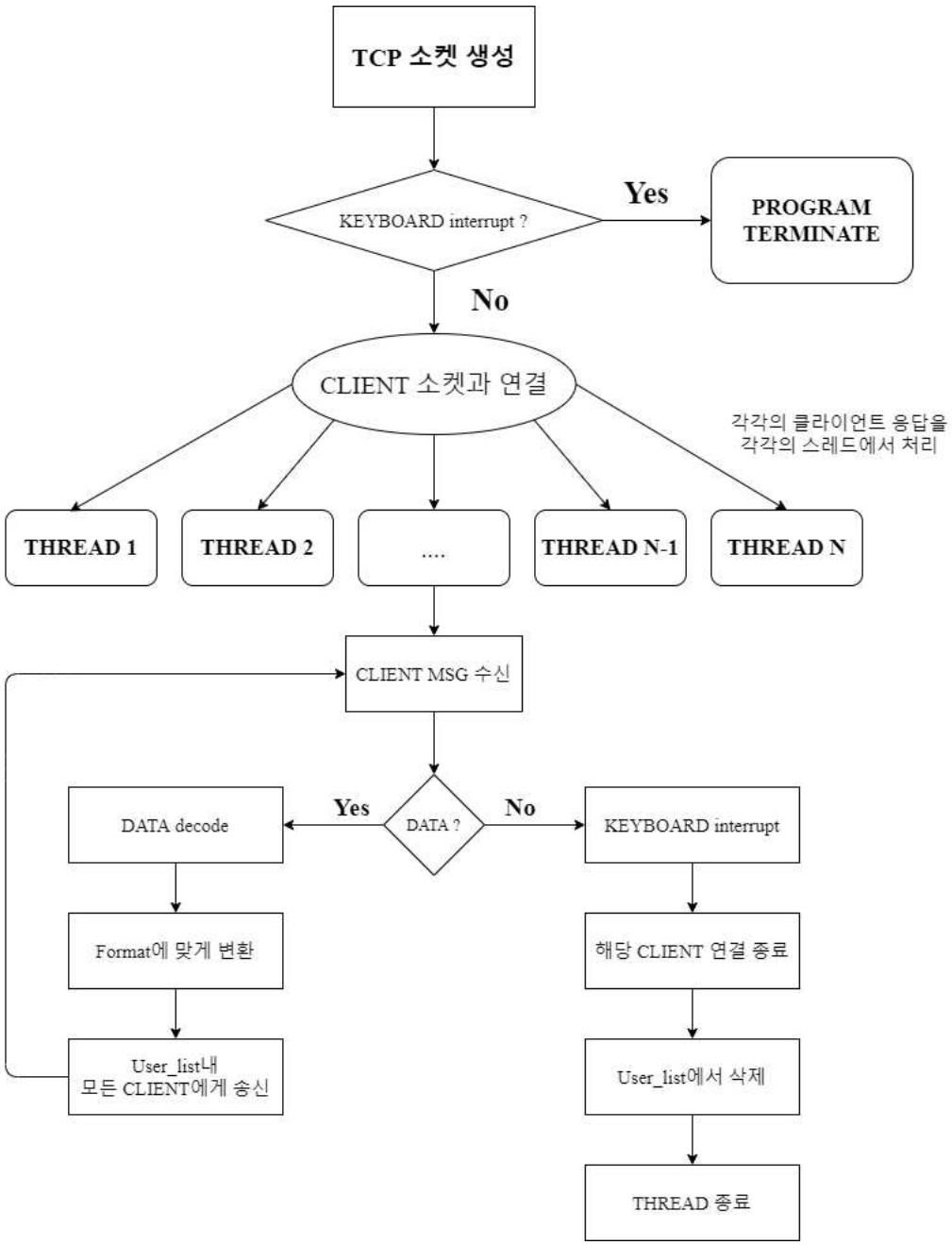
```
giwon@giwon-virtual-machine:~/Desktop/CSNetwork/pj2$ python3 --version
Python 3.8.5
```

2. Flow chart / Diagram

2-1. Client



2-2. Server


TCP 소켓 생성
KEYBOARD interrupt ?
Yes
PROGRAM TERMINATE
No
CLIENT 소켓과 연결
각각의 클라이언트 응답을 각각의 스레드에서 처리
THREAD 1
THREAD 2
....
THREAD N-1
THREAD N
CLIENT MSG 수신
DATA ?
Yes
No
DATA decode
KEYBOARD interrupt
Format에 맞게 변환
해당 CLIENT 연결 종료
User_list내 모든 CLIENT에게 송신
User_list에서 삭제
THREAD 종료

3. Detail explanation

3-1. cli.py

```
global user, userNum
user = 0
userNum = 0
```

전역 변수로서 클라이언트를 구분하기 위한 Key로 사용되는 주소를 저장하기 위한 변수 user와 서버에 접속 중인 클라이언트의 수를 저장하기 위한 변수 userNum을 설정하였다.

```
def main(serverName, serverPort):
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.connect((serverName, serverPort))
    enterChat(clientSocket)
    try:
        receive_thread = threading.Thread(target = handle_receive, args = (clientSocket,))
        receive_thread.daemon = True
        receive_thread.start()

        send_thread = threading.Thread(target=handle_send, args=(clientSocket,))
        send_thread.daemon = True
        send_thread.start()

        send_thread.join()
        receive_thread.join()

    except KeyboardInterrupt:
        clientSocket.close()
        print("\nexit")
        return
```

해당 파일의 메인 함수로 서버와 통신하는 데 필요한 clientSocket을 생성하고 송수신을 스레드를 생성하여 각각에 역할을 배정하였다. “ctrl + C”인 KeyboardInterrupt를 try except 구문을 통해 예외처리의 형태로 프로그램의 종료를 설계하였다.

```
def enterChat(clientSocket):
    global user, userNum
    # To know own ID
    ForID = clientSocket.recv(1024)
    ForID = ForID.decode()
    parseString = ForID.split(" ")
    index = parseString[3].find(':')
    user = parseString[3][index+1:]
    userNum = int(parseString[5][1:])
    enterString1 = "> Connected to the chat server (%d user online)"%(userNum)
    enterString2 = "> Connected to the chat server (%d users online)"%(userNum)
    if userNum < 2:
        print(enterString1)
    else:
        print(enterString2)
```

해당 클라이언트가 서버에 접속하였을 때 나타나는 문구를 출력하는 용도의 함수이다. 클라

이언트가 서버에 접속하게 되면 “New user server_name:Client_addr entered (N users online)” 문구를 접속 중인 모든 클라이언트에게 보내도록 설정을 하였기 때문에 그 문장을 수신하여 자기 자신의 Client주소와 현재 서버에 접속 중인 클라이언트의 개수를 전역 변수 user와 userNum에 저장한다. 이후, 접속 중인 클라이언트의 수에 따라 접속 문구를 다르게 출력한다.

```
def handle_receive(clientSocket):
    global user
    while True:
        data = clientSocket.recv(1024)
        data = data.decode()
        if not user in data:
            print(data)
```

서버로부터 수신한 데이터를 처리하는 함수이다. While문을 사용하여 keyboard interrupt가 일어나지 않는 이상 계속 서버로부터 데이터를 수신할 수 있도록 하였다. 이때, 서버에서 수신한 데이터를 수신받은 클라이언트를 포함한 모든 클라이언트에게 송신하도록 설계하였기 때문에 자기 자신이 보낸 데이터인지 확인하는 작업이 필요했다. 그리고 그 방법은 수신한 데이터에 포함된 Client_addr을 확인하는 것이다.
수신한 데이터의 형태는 다음과 같은 세 가지의 경우로 나뉘게 된다.

“[Server_name:Client_addr] Message”

“〉 New user Server_name:Client_addr entered (N users online)”

“〈 The user Server_name:Client_addr left (N users online)”

따라서, 수신한 데이터가 Client_addr 부분에서 자기 자신의 addr을 가지고 있다면 자기가 송신한 데이터이기에 출력하지 않고 나머지는 출력하도록 설계하였다.

```
def handle_send(clientSocket):
    while True:
        data = input()
        print("\033[A[You]", data)
        clientSocket.send(data.encode())
```

PYTHON의 표준 입력 함수를 활용하여 data 변수에 사용자 input을 저장하고 출력창에 보여주는 과정에서 입력한 줄이 사라지고 “[YOU] MSG”의 형태를 출력하기 위해 실행 창에서 현재 커서를 한 줄 위로 올려주는 역할의 아스키코드 “\033[A”를 활용하였다. 이후, 데이터를 인코딩하여 서버로 송신한다.

3-2. server.py

```
global user_list, clientNum
user_list = {}
clientNum = 0
lock = threading.Lock()
```

전역 변수로 서버에 접속 중인 클라이언트들을 관리하기 위한 딕셔너리 형의 변수 user_list와 서버에 접속 중인 클라이언트의 수를 나타내는 정수형 변수 clientNum을 설정하였다. 멀

티스레드 활용한 프로그래밍이므로 데드락을 방지하기 위한 락 변수 또한 설정하였다.

```python
def main(Host, Port):
    global user_list
    startString = "Chat Server started on port %d."%(Port)
    print(startString)

    # Create Socket
    serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    serverSocket.bind((Host, Port))
    serverSocket.listen()

    while True:
        try:
            # if client access, return socket
            client_socket, addr = serverSocket.accept()
            receive_thread = threading.Thread(target=handle_client, args=(client_socket, addr))
            receive_thread.daemon = True
            receive_thread.start()

        except KeyboardInterrupt:
            for con in user_list.values():
                con.close()
            serverSocket.close()
            print("\nexit")
            return
```

TCP 소켓을 생성하고 실행 시 입력받은 Host Ip와 Port 번호를 해당 소켓에 bind한 이후 서버에 접속하고자 하는 클라이언트의 유무를 listen 함수를 통해 확인한다. cli.py와 마찬가지로 “try except” 구문을 활용하여 Keyboard Interrupt를 확인하여 프로그램 종료 여부를 확인한다. 종료 시 접속된 모든 클라이언트의 연결을 닫고 자신의 소켓도 닫은 후 프로그램을 종료한다.
서버에 접속하는 클라이언트들의 데이터 송수신을 동시에 처리하기 위해 연결된 소켓을 멀티스레드를 활용하여 데이터 송수신을 처리한다.

```python
def handle_client(client_socket, addr):
    global user_list, clientNum
    lock.acquire()
    handle_Enter(client_socket, addr)
    lock.release()
    while True:
        data = client_socket.recv(1024)
        if not data: break
        msg = data.decode()
        string = "[%s:%s] %s"%(addr[0], addr[1], msg)
        print(string)
        for con in user_list.values():
            con.send(string.encode())
    lock.acquire()
    handle_Left(client_socket, addr)
    lock.release()
```

해당 스레드에서 처리하는 클라이언트의 접속과 퇴장을 처리하기 위해 handle_Enter 함수와

handle_Left 함수를 해당 함수 처음과 마지막에 실행한다. While문을 활용하여 데이터의 송수신이 일회성의 성질을 띄지 않게 하며 수신한 데이터가 Keyboard Interrupt일 경우 클라이언트가 접속을 종료했다고 인지하고 While문을 탈출한다. 수신한 데이터를 포맷에 맞게 변환한 후 출력을 하고 접속 중인 모든 클라이언트에게 송신한다.

```
def handle_Enter(client_socket, addr):
    global user_list, clientNum
    clientNum += 1
    printEnter = returnEnter(addr)
    print(printEnter)
    user = addr[1]
    user_list[user] = client_socket
    for con in user_list.values():
        con.send(printEnter.encode())
```

```
def returnEnter(addr):
    global clientNum
    enterString1 = "> New user %s:%s entered (1 user online)"%(addr[0], addr[1])
    enterString2 = "> New user %s:%s entered (%d users online)"%(addr[0], addr[1], clientNum)
    if clientNum == 1:
        return enterString1
    else:
        return enterString2
```

클라이언트가 서버에 접속했을 때 출력되어야 하는 문구를 처리하는 함수이다. 전역 변수 clientNum을 사용하여 문구를 완성하고 서버에 접속 중인 모든 클라이언트에게 전달한다.

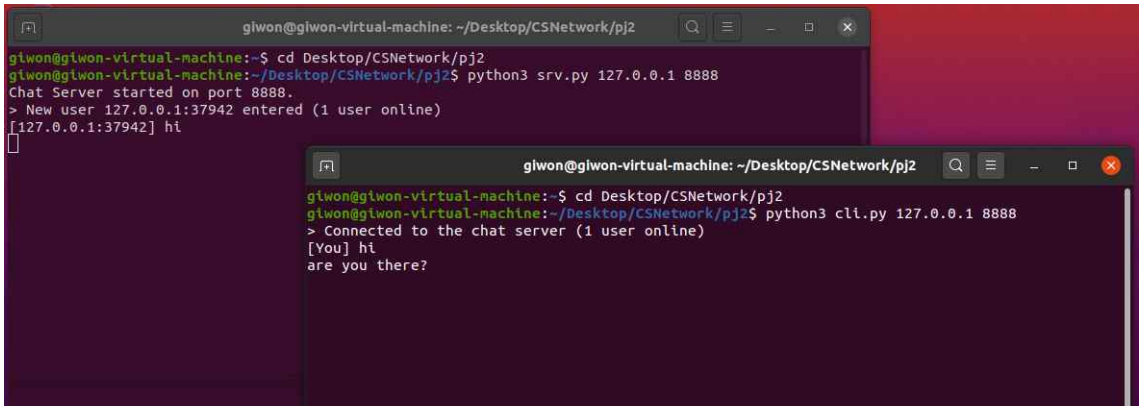
```
def handle_Left(client_socket, addr):
    global user_list, clientNum
    clientNum -= 1
    printLeft = returnLeft(addr)
    print(printLeft)
    for con in user_list.values():
            con.send(printLeft.encode())
    del user_list[addr[1]]
    client_socket.close()
```

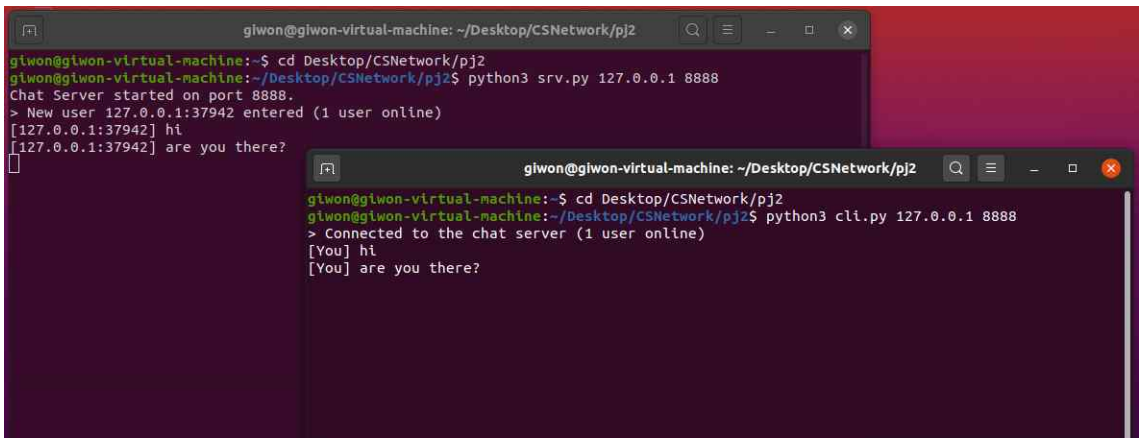
```
def returnLeft(addr):
    global clientNum
    LeftString1 = "< The user %s:%s left (%d user online)"%(addr[0], addr[1], clientNum)
    LeftString2 = "< The user %s:%s left (%d users online)"%(addr[0], addr[1], clientNum)
    if clientNum < 2:
        return LeftString1
    else:
        return LeftString2
```

클라이언트가 서버에서 퇴장할 때 출력되어야 하는 문구를 처리하는 함수이다. 위 함수와 마찬가지로 완성한 문구에 대하여 서버에 접속 중인 모든 클라이언트에게 전달하고 해당 클라이언트는 종료하였기 때문에 user_list에서 삭제한다.

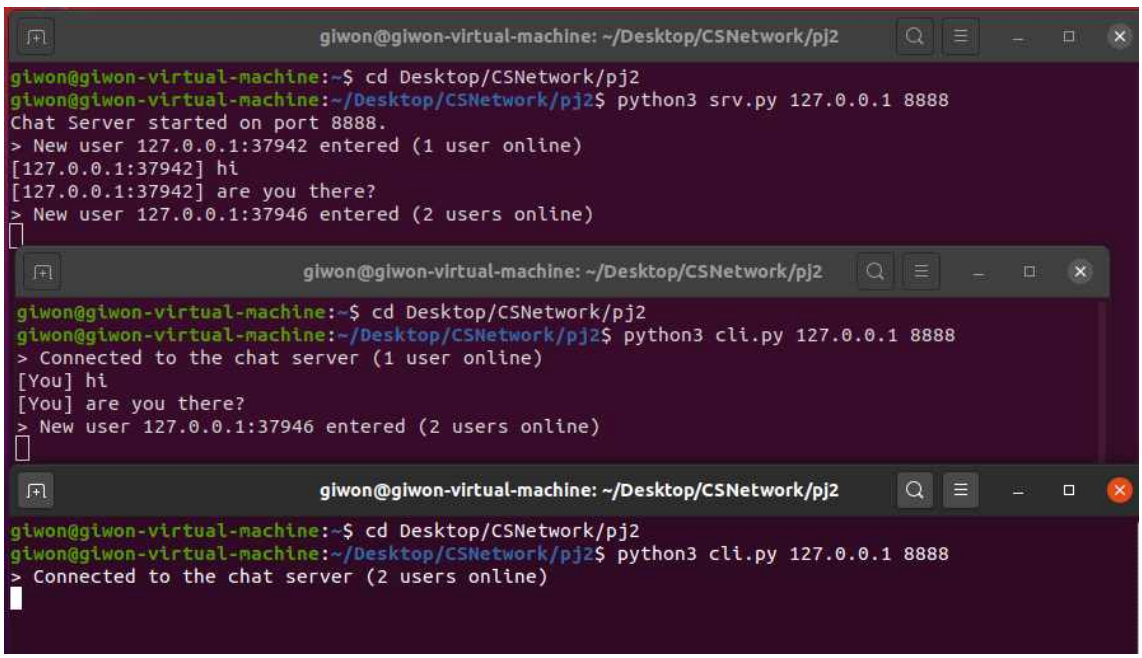
4. Snapshot

4-1. 1개의 클라이언트가 서버에 데이터를 전송하는 과정

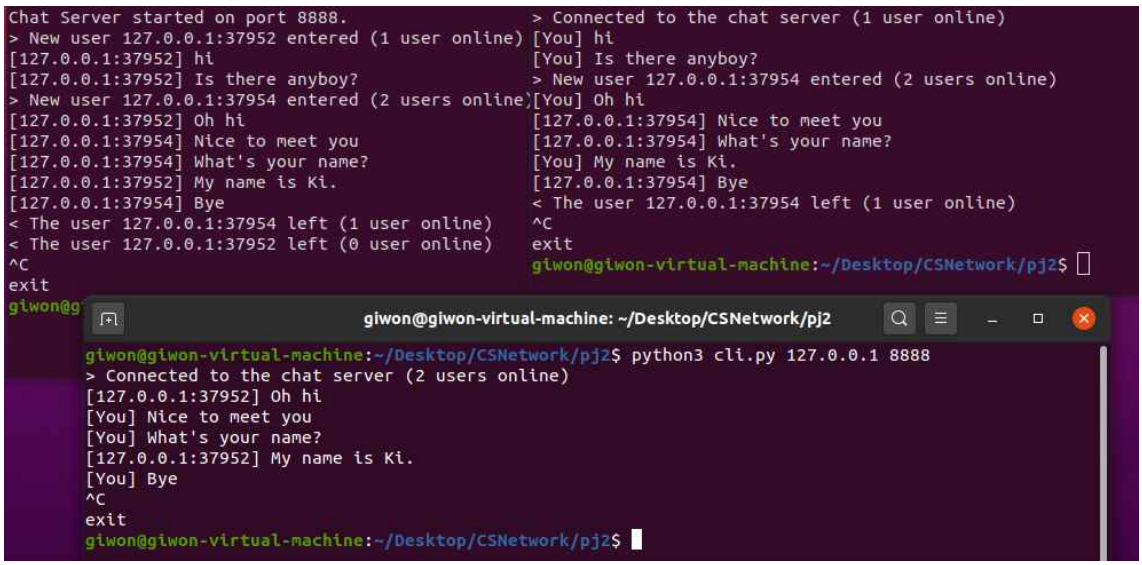




4-2. 2번째 클라이언트 접속



4-3. 2개의 클라이언트 대화



5. What are these functions?

socket(socket.AF_INET, socket.SOCK_STREAM)

매개변수의 의미는 각각 주소 패밀리, 소켓 유형이다. 주소 패밀리는 AF_INET을 기본값으로 가지며 AF_INET6, AF_UNIX, AF_CAN, AF_PACKET, AF_RDS 중 한 개를 선택하여 값으로 지정해야 한다. 소켓 유형은 SOCK_STREAM을 기본값으로 가지며 SOCK_DGRAM, SOCK_RAW, SOCK_상수 중 한 개를 선택하여 값으로 지정해야 한다. 이 매개변수를 이용하여 새로운 소켓을 생성하는 역할의 함수이다.

setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

소켓의 송수신 동작 옵션을 설정하는 함수이다. 첫 번째 매개변수는 소켓의 level로 어떤 레벨의 소켓 정보를 불러오거나 수정할 것인지를 의미하고 SOL_SOCKET 또는 IPPROTO_TCP를 주로 사용한다. 소켓이 강제 종료되거나 비정상 종료되는 경우에 다시 프로그램을 실행시키는 경우에 Bind error가 발생하게 된다. 프로그램이 비정상 종료되었음에도 커널이 소켓의 bind 정보를 유지하고 있기 때문이다. 따라서, SO_REUSEADDR 옵션을 설정하여 커널이 기존에 할당한 소켓 bind 정보를 재사용할 수 있도록 한다.

bind((“0.0.0.0”, 7777)

소켓을 주어진 매개변수인 ADDRESS에 바인드하는 함수이다. 이미 바인드 되어 있는 경우 에러가 발생하게 된다. ADDRESS의 경우 (host, port)의 주소 패밀리 형식을 따른다.

listen(5)

클라이언트가 바인드된 포트로 연결을 요청할 때까지 대기하는 블록킹 함수이다. 클라이언트로부터 연결 요청이 오면 블록킹을 해제하고 다음 행에 해당 연결을 받아들이기 위한 accept( ) 함수를 호출하는 부분이 보통 이어진다. 매개변수의 의미는 backlog로 0 이상의 정수로 새로운 연결을 진행하기 전의 연결 대기열의 길이를 의미한다.

connect((host, port))
클라이언트 소켓에서 매개변수로 주어진 주소 패밀리에 위치한 서버 소켓과의 연결을 요청한다. 클라이언트 소켓은 바인드할 필요 없이 직접 주어진 소켓 정보를 통해 서버 소켓을 찾아 연결을 요청한다. 시그널로 연결이 방해받을 경우, 해당 연결이 완료될 때까지 대기하거나 예외 처리가 발생하지 않아서 소켓이 블로킹하거나 시간 제한이 존재할 경우 socket.timeout을 발생시킨다.

accept( )
서버 소켓이 클라이언트 소켓의 연결 요청을 받아들이는 함수이다. 이때, 반환하는 값은 실제 통신하는 소켓을 (연결에서 데이터를 송수신할 수 있는 소켓 객체, 클라이언트 소켓의 주소)의 형태로 반환한다.

close( )
해당 소켓이 닫힌 상태를 띄게 하는 함수이다. 해당 함수의 실행 이후의 소켓에 대한 모든 연산은 실패하여 실행되지 않는다. 프로그램이 종료된 이후 자동으로 소켓은 닫히지만, 명시적으로 close( )함수를 이용하여 해당 소켓의 연결 종료를 나타내는 것이 좋다.

Thread( )
스레드 객체를 생성하는 함수이다. 매개변수로 스레드에서 실행할 target 함수와 해당 함수에서 필요한 매개변수를 의미하는 args 튜플과 kwargs 딕셔너리가 있다. 생성된 객체의 start() 함수를 사용하여 스레드 활동을 시작한다. 객체 당 단 한 번 이하로 호출되어야 하므로 두 번 이상 호출될 경우 RuntimeError를 발생시킨다. 스레드는 daemon 스레드 플래그가 존재하는데 이것의 의미는 오직 데몬 스레드만 남았을 때 전체 PYTHON 프로그램이 종료된다는 것이다. 이번 과제의 경우 Keyboard interrupt 발생 시 메인 스레드가 종료됨과 동시에 모든 서브 스레드가 종료되어야 하므로 daemon 스레드를 활용하였다.

6. Multi-thread vs. select( )

소켓 프로그래밍에서는 서버 소켓을 생성한 뒤 특정 네트워크 포트에 바인드하여 listen( ) 함수를 통해 해당 포트로 접속할 수 있는 대기열의 크기를 지정하고 accept( ) 함수를 통해 서버 내 클라이언트 소켓을 생성하여 접속 요청을 한 클라이언트 소켓과 연결을 하여 송수신을 진행합니다. 이렇듯 서버 소켓은 클라이언트가 서버에 접속할 때마다 서버용 클라이언트 소켓을 각각 생성하여 대상과 peer-to-peer 방식으로 실제 통신을 하게 됩니다.
기본적으로 send( ), recv( ), accept( ) 등이 모두 블록킹 함수이기 때문에 하나의 소켓의 입출력을 기다리는 동안 다른 소켓과의 통신을 진행할 수 없습니다. 이를 해결하기 위한 방법의 하나가 바로 Multi-thread입니다.
서버 소켓의 accept( ) 함수가 값을 반환하는 시점에 스레드 핸들러 함수에 통신용 클라이언트 소켓을 넘겨주어 새롭게 생성된 스레드에서 해당 클라이언트와의 통신을 처리하는 방식입니다. 이 방식의 장단점은 다음과 같습니다.
장점: 우선 스레드는 전체 코드를 실행하는 것이 아니라 코드 조각을 실행하는 것이므로 프로세스를 복사하는 멀티 프로세스 방식보다 좀 더 가볍고 빠르게 작동시킬 수 있다. 또한, 스레드는 거의 동시에 작업이 가능하므로 서버에 접속 중인 클라이언트의 수와 관계없이 동

시에 통신할 수 있다.
단점: 서버에 접속 중인 클라이언트의 수만큼 스레드를 생성하여 처리하므로 서버의 시스템 자원을 많이 사용한다. 또한, 스레드가 작동하기 위해 켄텍스트 스위칭이 계속 발생하여 사용되는 스레드의 수가 많아지면 성능이 하락한다. 마지막으로 하나의 프로세스 내의 다수의 스레드가 존재하기 때문에 그중 하나라도 문제가 생긴다면 전체 프로세스에 악영향을 끼칠 수 있다.

위의 단점들을 극복하기 위해 한 개의 스레드에서 여러 개의 클라이언트 송수신 작업을 돌아가면서 작업하는 방식을 고안했는데 이것이 바로 Select( ) 함수를 활용한 입출력 다중화를 사용하는 모델이다. 입출력 다중화란 여러 개의 입출력 클라이언트가 존재할 때 여러 개의 스레드에 할당하는 것이 아니라 대상들을 확인하면서 입출력이 필요한지 아닌지를 검사하여 작업을 수행하는 것이다.
리눅스의 경우 소켓을 파일의 형태로 다루게 되고 시스템이 파일을 열면 file descriptor로 관리를 한다. select( ) 함수는 file descriptor의 fd_set을 확인하면서 값이 1인 파일에 대해서는 읽기 또는 쓰기를 검사하고 변화가 있다면 1로 세팅한다. 이런 식으로 1로 세팅된 소켓들에 대해 통신을 진행한다. 이때 함수가 블록킹되는 것을 방지하기 위해 타임 아웃을 설정한다.
장점: 별도의 프로세스나 스레드의 생성 없이 여러 개의 서비스를 제공할 수 있으므로 시스템 자원 부분에서 많은 이득이 생긴다.
단점: 모든 file descriptor의 fd_set을 확인해야 하므로 실행 시간이 길어진다. 커널에 의해 완성되는 기능이 아닌 순수한 함수의 기능이므로 전달되는 정보가 커널에 등록되지 않아 함수를 호출할 때마다 커널이 관련 정보를 확인해야 한다. 또한 검사할 수 있는 fd의 개수가 1024개로 한정되어 있다.

7. Reference

https://docs.python.org/ko/3/library/socket.html
http://www.w3big.com/ko/python/python-socket.html
https://soooprmx.com/archives/8737
https://docs.python.org/ko/3.8/library/threading.html
https://soooprmx.com/archives/10919
https://jongmin92.github.io/2019/02/28/Java/java-with-non-blocking-io/