

Computer Network Project 2

2016147030 송민석

Introduction | Reference

OS : Ubuntu 18.04 (WSL)

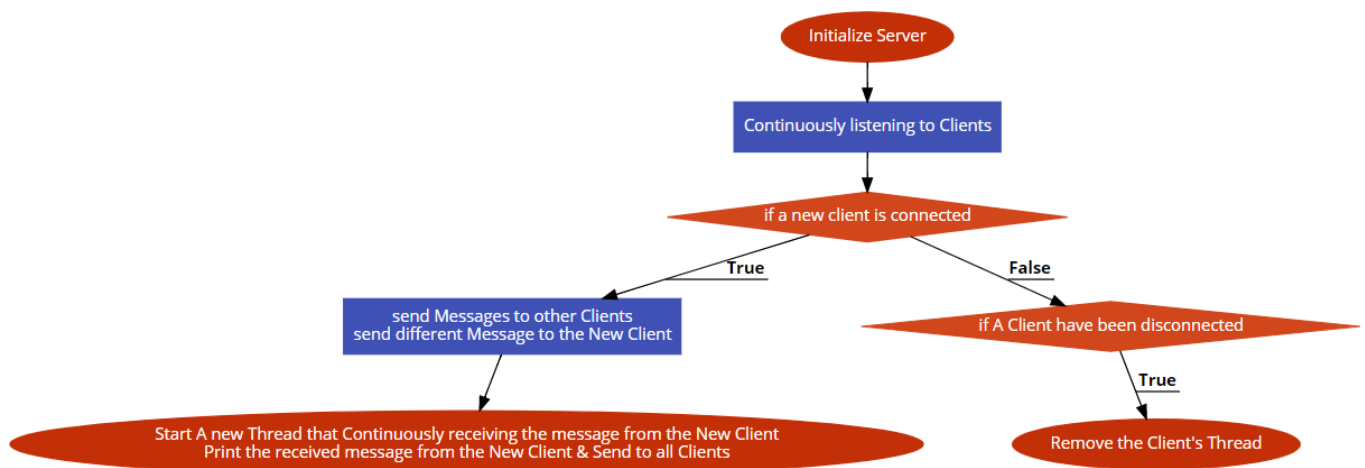
Used Language/Version : Python 3.6.9

Reference : Computer Network : A Top-down Approach CH.2 /

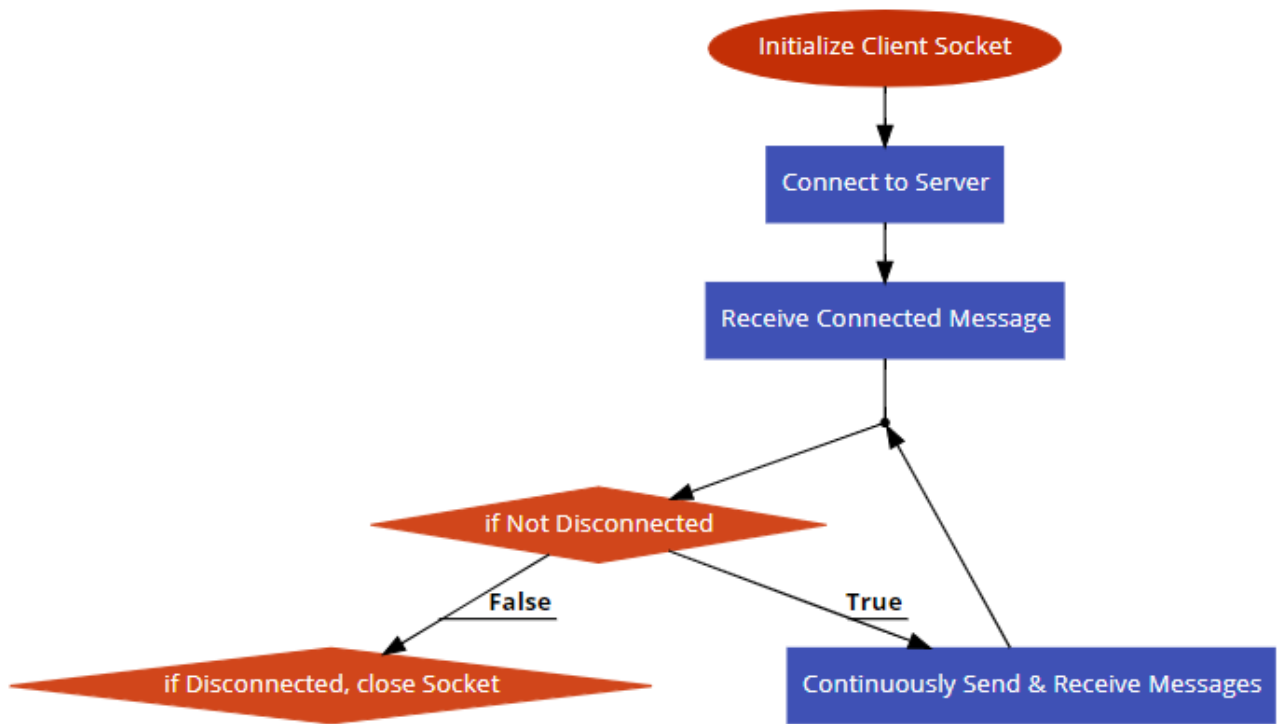
<https://forums.codeguru.com/showthread.php?375180-select-vs-multithreaded-tcp-server-app>

<https://m.blog.naver.com/PostView.nhn?blogId=r1aauddlf200&logNo=30141984634&proxyReferer=https:%2F%2Fwww.google.com%2F>

Flow Chart



srv.py의 흐름도는 위와 같다. 먼저 Chatting Server를 Initialize(인자로 주어진 IP, Port를 이용해 socket을 bind, listen)한다. 이후 종료될 때까지 새로운 클라이언트가 접속하는 것을 계속 기다린다(Continuously listening to Clients). 클라이언트가 접속할 시(if a new client is connected), 다른 클라이언트들에게 이를 메시지로 알리고 새로운 클라이언트에는 서버에 연결되었다는 메시지를 전송한다(send Messages~). 직후 클라이언트의 메시지를 꾸준히 수신하기 위한 Thread를 새로 생성한다(Start new Thread). 이후 각 Thread에서 새로운 메시지가 수신될 때마다 Server 및 다른 Client에게 공유한다(Print received message). 만약 클라이언트가 접속을 종료한다면 (if a Client have been disconnected) 클라이언트에 할당된 Thread를 종료하고, 클라이언트가 접속을 종료했음을 다른 클라이언트들에게 알린다.



cli.py의 흐름도는 위와 같다. 먼저 클라이언트 소켓을 Initialize하고 서버와 connect한다. Connect 직후에는 연결되었다는 메시지를 수령하고, 이후 disconnect되지 않는 한 꾸준히 서버와 메시지를 주고받는다. 만약 disconnect 즉 종료된다면 소켓을 close한다.

Detailed Logical Explanations (with Code blocks)

srv.py

```

srv.py

1  # import required Libraries
2  import socket
3  from threading import Thread
4  import sys
5  import atexit

```

구현에 필요한 내부 라이브러리를 import하는 부분이다. 소켓 프로그래밍을 위한 socket, 멀티스레딩을 이용하기 위한 Thread, linux 명령어 실행 시 인자로 받아오는 ip, port를 가져오기 위한 sys, 프로그램이 종료될 때 소켓을 종료하는 명령을 수행하기 위한 atexit 라이브러리를 import 하였다. 이는 cli.py도 동일하므로 cli.py를 설명하는 부분에서는 생략한다.

```

srv.py
60 # Get IP address & Port to bind
61 IP = sys.argv[1]
62 serverPort = int(sys.argv[2])
63
64 # Set Socket options and initialize Server's Socket
65 serverSock = socket(AF_INET, SOCK_STREAM)
66 serverSock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
67 serverSock.bind(('0.0.0.0', serverPort))
68
69 print("Chat Server started on port {}".format(serverPort))
70 serverSock.listen()
71
72 # Initialize a Client List
73 clientlist = []
74

```

서버의 socket을 initialize하는 부분이다. 명령어 실행시 인자로 받아오는 IP, Port 정보를 이용하여 소켓을 IPv4, TCP 의 형태로 bind하고 옵션을 설정한다(setsockopt, bind등에 대한 설명은 따로 정리해두었다). 이후 채팅 서버가 시작되었음을 출력하고, 종료될 때 까지 소켓으로 들어오는 메시지들을 listen한다. 또한 클라이언트 관리를 위해 global variable로 클라이언트 list를 하나 생성하였다.

```

srv.py
75 # Main
76 while 1:
77
78     # Continuously Accept New Client
79     connectionSock, addr = serverSock.accept()
80     clientlist.append((connectionSock, addr))
81     login(connectionSock, addr, clientlist)
82     # Print New Client's Info
83     if len(clientlist) <= 1:
84         print("> New user {}:{} entered ({} user online)".format(addr[0], addr[1], len(clientlist)))
85     else:
86         print("> New user {}:{} entered ({} users online)".format(addr[0], addr[1], len(clientlist)))
87
88     # Start New Thread for Client
89     newThread = Thread(target=messenger, args=(connectionSock, addr, clientlist))
90     newThread.daemon = True
91     newThread.start()

```

Main 부분, 즉 Flow Chart에서 꾸준히 Client를 받는 역할을 하는 부분이다. 새로운 client가 connect를 시도하면 accept함으로써 새 client에 대한 소켓을 connectionSock에, 그리고 해당 client 식별을 위한 ip, port정보를 addr에 할당한다. 이후 clientlist에 새 client를 추가하고, 새 client가 연결되었음을 client들에게 전송하는 함수인 login을 호출한다. 이후 서버에도 새 client의 연결을 출력하는데, user/users의 구분을 위해 if문을 이용하였다. 연결을 알리는 작업이 끝나면, 해당 client가 접속 종료되기 전까지 계속 메시지를 주고받도록 하는 함수인 messenger를 하나의 새로운 Thread로써 실행한다. Multi-user와의 chat을 위해 이 Threading 과정은 필수적이다.

```

srv.py
7  # A Function to Announce that a new Client has been connected to server
8  def login(sock, addr, clientlist):
9      # Get the New Client's Info
10     clientIP, clientPort = addr[0], addr[1]
11
12     if len(clientlist) <= 1:
13         # Send Connected Message to the new Client
14         sock.send("> Connected to the chat server ({} user online)".format(
15             len(clientlist)).encode())
16         # Send to the other Clients that a new client has been arrived
17         for client in clientlist:
18             if not((client[1][0] == clientIP) & (client[1][1] == clientPort)):
19                 client[0].send("> New user {}:{} entered ({} user online)".format(
20                     clientIP, clientPort, len(clientlist)).encode())
21     else:
22         sock.send("> Connected to the chat server ({} users online)".format(
23             len(clientlist)).encode())
24         for client in clientlist:
25             if not((client[1][0] == clientIP) & (client[1][1] == clientPort)):
26                 client[0].send("> New user {}:{} entered ({} users online)".format(
27                     clientIP, clientPort, len(clientlist)).encode())

```

위에서 언급한 새로운 client의 도착정보를 client들에 전송하는 login 함수이다. 새 client와 연결된 새로운 소켓, 새 client의 정보, 현재까지의 clientlist를 인자로 받는다. 먼저 user/users의 구분을 위해 if문을 이용하여 구분하고, clientlist의 각 client에 대해 for문과 if문을 수행함으로써 새 client와 기존 client들에 각각 다른 형태의 알림을 전송하도록 하였다.

```

srv.py
30 def messenger(connectionSock, addr, clientlist):
31     # Get the New Client's Info
32     clientIP, clientPort = addr[0], addr[1]
33
34     while 1:
35         try:
36             message = connectionSock.recv(1024).decode('utf-8')
37             res = "[{}:{}] {}".format(clientIP, clientPort, message)
38             if len(message) > 0:
39                 print(res)
40                 for client in clientlist:
41                     # Send new Client's message with [You]
42                     if ((client[1][0] == clientIP) & (client[1][1] == clientPort)):
43                         client[0].send("[You] " + message).encode())
44                     # Send res to other clients
45                 else:
46                     client[0].send(res.encode())
47             continue
48         except BrokenPipeError:
49             if len(clientlist) <= 2:
50                 print("< The user {}:{} left ({} user online)".format(clientIP, clientPort, len(clientlist)-1))
51             else:
52                 print("< The user {}:{} left ({} users online)".format(clientIP, clientPort, len(clientlist)-1))
53
54             for client in clientlist:
55                 if client[1] == addr:
56                     clientlist.remove(client)
57             break

```

메시지를 주고받는 함수이다. 각 client와 server가 메시지를 주고받도록 하기 위해, 각 Client에 대해 생성된 Thread내에서 개별적으로 동작한다. 따라서 이 함수가 할당된 Thread의 수는 Client의 수와 일치한다. 각 Client는 자신이 보낸 메시지는 [You]와 같이 보여야하고, 다른 Client의 메시지는 [ip:port]와 같이 보여야하므로 이 역시 if문을 이용하여 각 client마다 다른 형식으로 전송되도록 하였다. Client가 종료되면 소켓의 연결이 끊어져 BrokenPipeError를 출력하는데, 이를 Handle하여 서버와 나머지 client들에게 해당 client가 채팅 서버를 떠났음을 알리도록 하였다.

cli.py

```
cli.py
1  from socket import *
2  from threading import Thread
3  import sys
4  import atexit
```

각 라이브러리가 필요한 이유는 srv.py 와 같다.

```
cli.py
6  # Get Server's IP address & Port to connect
7  IP = sys.argv[1]
8  serverPort = int(sys.argv[2])
9
10 # Connect to Server
11 clientSock = socket(AF_INET, SOCK_STREAM)
12 clientSock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
13 clientSock.connect((IP, serverPort))
```

Client 의 Socket 을 initialize 하는 부분이다. 똑같이 IPv4, TCP 형태이며 setsockopt, connect 를 이용하여 옵션을 설정하고 서버에 연결한다.

```
# Sending Message
def sender():
    while 1:
        sending_message = input()
        clientSock.send(sending_message.encode('utf-8'))
```

```
# Receiving Message
def receiver():
    while 1:
        received_message = clientSock.recv(1024).decode('utf-8')
        print(received_message)
```

```
# Make 2 Threads
sendingThread = Thread(target = sender)
receivingThread = Thread(target = receiver)
receivingThread.start()
sendingThread.start()
```

sender 와 receiver 는 각각 사용자가 Client 에 입력한 메시지를 socket 을 통해 서버로 전송하고, server 로부터 전송되는 모든 메시지를 수신하여 출력하는 기능을 담당한다. 종료될 때까지 실행되어야 하므로 while 문을 이용하였고, 각각 별개로 실행되어야 하므로 Threading 을 이용하여 실행되도록 하였다.

srv.py	cli.py
93	32
94 def disconnect():	33 def disconnect():
95 serverSock.close()	34 clientSock.close()
96	35
97	36 atexit.register(disconnect)
98 atexit.register(disconnect)	

마지막으로 서버와 클라이언트 모두 종료 시 socket 을 close 하기 위해 위와 같이 구현하였다.

8 가지 Key Implementing 함수에 대한 Explanation

`socket(socket.AF_INET, socket.SOCK_STREAM)` : IPv4, TCP 형태의 소켓 프로그래밍을 위한 socket 객체이다. `AF_INET` 은 IPv4 를 의미하며 `SOCK_STREAM` 은 TCP 를 의미한다.

`setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)` : 생성한 소켓의 세부 옵션사항을 조정한다. `SOL_SOCKET` 을 통해 프로토콜이 아닌 소켓에서 제어하는 옵션을 제어하도록 하고, `SO_REUSEADDR` 을 통해 이미 사용중인 ip 주소와 port 를 재사용할 수 있게 한다.

`bind(("0.0.0.0", 7777))` : 사용할 호스트의 포트번호를 지정하여 소켓을 해당 포트에 연결한다. "0.0.0.0"은 모든 ip 로부터 connect 를 기다리도록 한다.

`listen(5)` : 서버가 클라이언트로부터의 TCP 연결 요청을 듣도록 한다. 5 는 queue 되는 최대 연결 수를 5 로 지정하는 것이다.

`connect((host, port))` : 클라이언트-서버 사이에 TCP 연결을 먼저 설정하기 위해 필요하다. (host, port)는 서버의 주소이며 이 함수가 실행된 후 3-way handshake 가 수행되고 TCP 연결이 설정된다.

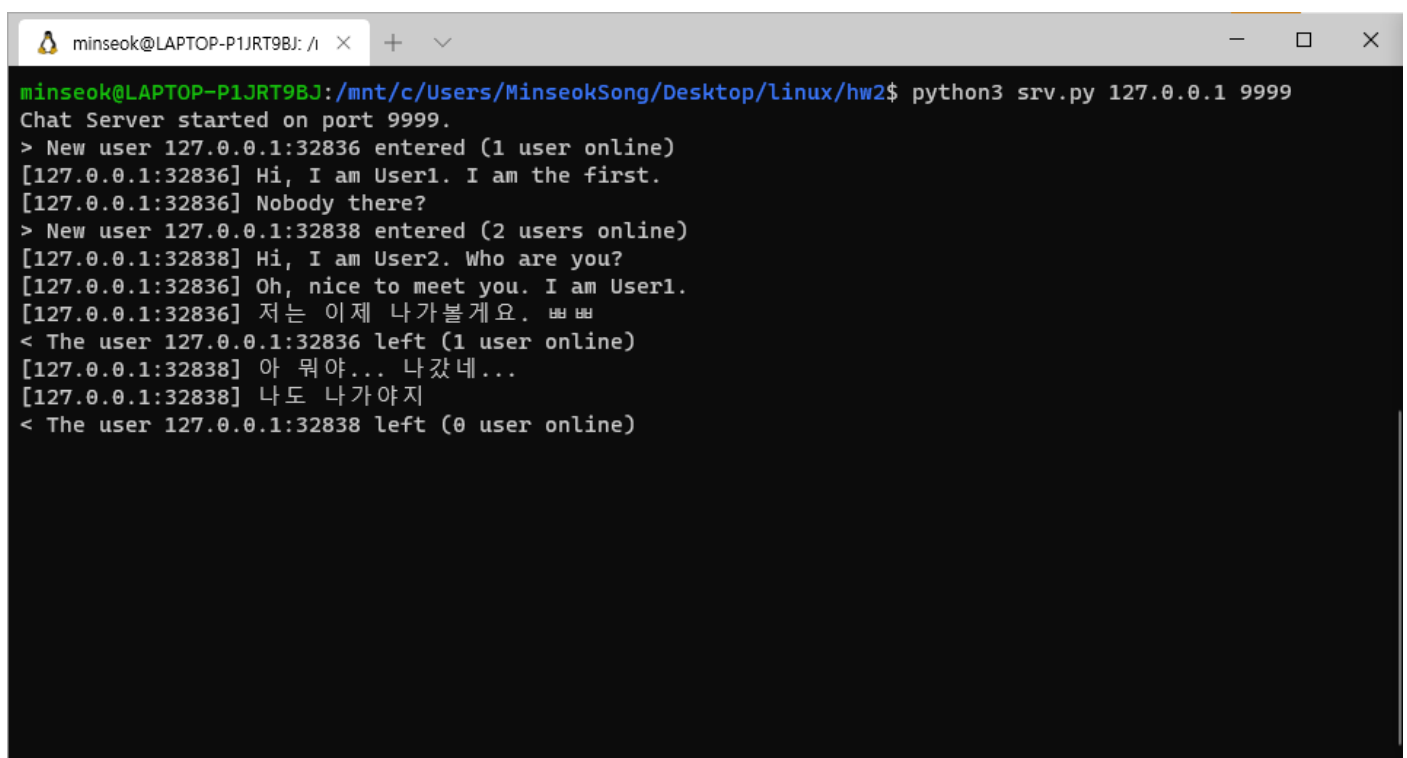
`accept()` : 클라이언트의 연결 요청을 수락한다. Server 에 client 가 connect 를 시도하는 경우 `accept()`를 이용하여 수락하도록 해야한다.

`close()` : 소켓을 닫아준다. 과제 조건에 따라 프로그램 종료 시 필수적으로 호출되어야 한다.

`Thread()` : 특정 Client 와의 메시지 주고 받는 과정이 타 client 들과의 과정과 독립적으로 수행되어야 하므로, Multi-Thread 를 이용하여 구현하기 위해 사용한다.

Result Screenshot with 3 Clients

Server



```
minseok@LAPTOP-P1JRT9BJ: /1 X + v
minseok@LAPTOP-P1JRT9BJ:/mnt/c/Users/MinseokSong/Desktop/linux/hw2$ python3 srv.py 127.0.0.1 9999
Chat Server started on port 9999.
> New user 127.0.0.1:32836 entered (1 user online)
[127.0.0.1:32836] Hi, I am User1. I am the first.
[127.0.0.1:32836] Nobody there?
> New user 127.0.0.1:32838 entered (2 users online)
[127.0.0.1:32838] Hi, I am User2. Who are you?
[127.0.0.1:32836] Oh, nice to meet you. I am User1.
[127.0.0.1:32836] 저는 이제 나가볼게요. ㅎㅎ
< The user 127.0.0.1:32836 left (1 user online)
[127.0.0.1:32838] 아 뭐야... 나갔네...
[127.0.0.1:32838] 나도 나가야지
< The user 127.0.0.1:32838 left (0 user online)
```

Client1 (User1)

```
minseok@LAPTOP-P1JRT9BJ: ~/i X + v
minseok@LAPTOP-P1JRT9BJ:/mnt/c/Users/MinseokSong/Desktop/linux/hw2$ python3 cli.py 127.0.0.1 9999
> Connected to the chat server (1 user online)
Hi, I am User1. I am the first.
[You] Hi, I am User1. I am the first.
Nobody there?
[You] Nobody there?
> New user 127.0.0.1:32838 entered (2 users online)
[127.0.0.1:32838] Hi, I am User2. Who are you?
Oh, nice to meet you. I am User1.
[You] Oh, nice to meet you. I am User1.
저는 이제 나가볼게요. ㅎㅎ
[You] 저는 이제 나가볼게요. ㅎㅎ
^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
minseok@LAPTOP-P1JRT9BJ:/mnt/c/Users/MinseokSong/Desktop/linux/hw2$ |
```

Client2 (User2)

```
minseok@LAPTOP-P1JRT9BJ: ~/i X + v - □ X
minseok@LAPTOP-P1JRT9BJ:/mnt/c/Users/MinseokSong/Desktop/linux/hw2$ python3 cli.py 127.0.0.1 9999
> Connected to the chat server (2 users online)
Hi, I am User2. Who are you?
[You] Hi, I am User2. Who are you?
[127.0.0.1:32836] Oh, nice to meet you. I am User1.
[127.0.0.1:32836] 저는 이제 나가볼게요. ㅎㅎ
[127.0.0.1:32836]
아 뭐야... 나갔네...
[You] 아 뭐야... 나갔네...
나도 나가야지
[You] 나도 나가야지
^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
minseok@LAPTOP-P1JRT9BJ:/mnt/c/Users/MinseokSong/Desktop/linux/hw2$ |
```

Difference between using multi-thread and the function select()

select()를 이용하면 Main 함수를 움직이는 1 개의 쓰레드만으로 여러 클라이언트와 통신이 가능하게 할 수 있다. Client 소켓들을 하나의 배열에 집어넣고, select 를 호출하며 변화가 있는 소켓을 찾아 해당 소켓에서 필요한대로 send, recv, accept 등을 호출한다. 이와 같은 방식은 Multi-Thread 방식에 비해 적은 Thread 를 사용하므로 CPU 자원을 덜 소모하고 더 많은 client 를 처리할 수 있다. 하지만 위 방법대로 작동하기 때문에 소켓이 변화가 있는지 일일이 확인, 비교해야한다는 단점이 있다. 따라서 다수의 Client 로부터 동시에 정보가 들어온다면 지연 등의 문제가 발생할 수 있다.

이에 반해 Multi-thread 방식은 Client 의 수에 비례하여 다수의 쓰레드를 이용하므로 컴퓨팅 자원 활용적인 측면에서는 덜 효율적이거나, 각 Thread 가 병렬적으로 동작하므로 각 Client 와의 통신이 서로 영향을 미치지 않아 동시에 여러 Client 로부터 정보가 들어오는 상황에 대해 select()를 이용한 방식보다 유리하다. 해당 Project 2 와 같은 Chat Server 를 설계한다면 그 Chat Server 의 이용자 수, 동시 사용률 등을 고려하여 방법을 선택하는 것이 바람직할 것이다.