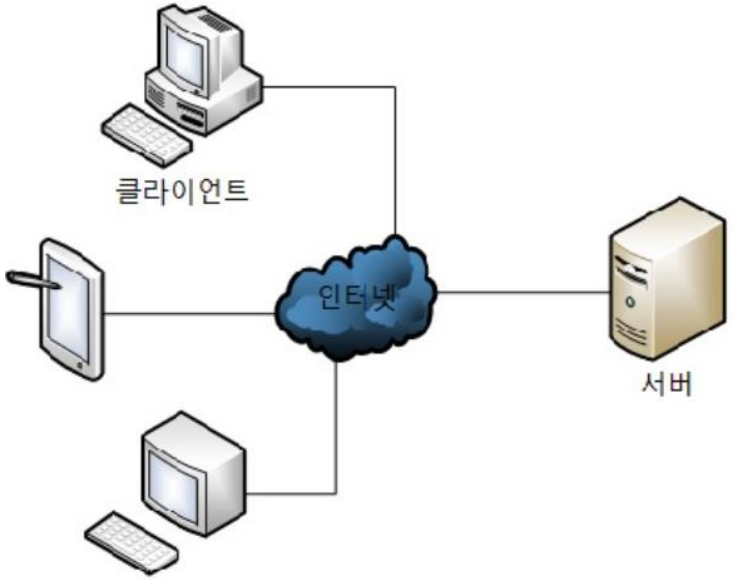
네트워크 모델

네트워크에 연결된 컴퓨터의 역활을 정하는 방법을 네트워크 모델이라한다

- 서비스 제공자(서버)
- 서비스 요청자(클라이언트)



1. 피어-투-피어(peer-to-peer) 구조

- 모든 컴퓨터가 동등하게 요청과 응답이 가능한 구조
- 각 노드가 자원을 분산해서 관리하고 제공

2. 클라이언트-서버 구조

- 서비스 제공자를 제공하는 서버와 서비스를 요청하는 클라이언트로 구성
- 모든 자원이 서버에 집중(증앙 집중식 구조)
- 인터넷에서 사용되는 가장 일반적인 네트워크 구조

동시성(Concurrent) 소켓 프로그래밍

- 동시성 프로그램: 여러 가지 작업을 동시에 처리하는 프로그램
- 서버가 클라이언트의 요청을 처리할 때
  - 서버가 연결된 클라이언트 요청을 모두 처리한 후에, 순차적으로 새로운 클라이언트의 요청을 처리하면 새 클라이언트의 대기 시간이 증가
  - 만약 클라이언트 요청이 입출력을 포함하면 처리 시간이 무한히 길어질 수 있음
  - 어떤 클라이언트의 입출력을 기다리는 동안에 다른 클라이언트를 서비스하면 효율적인 서비스가 가능
- 동시성 소켓 프로그램 구현
  - 멀티스레드(Multi-Thread) 이용
  - select 모듈 이용
  - socketserver 모듈 이용
  - selectors 모듈 이용
  - asyncio 모듈 이용

서버 종류

- 반복 서버(iterative server)
  - 클라이언트의 요청을 하나씩 서비스하는 서버
  - A가 서비스 받고 있는 동안에 B는 대기
  - A 서비스가 길어지면 B의 대기시간도 증가

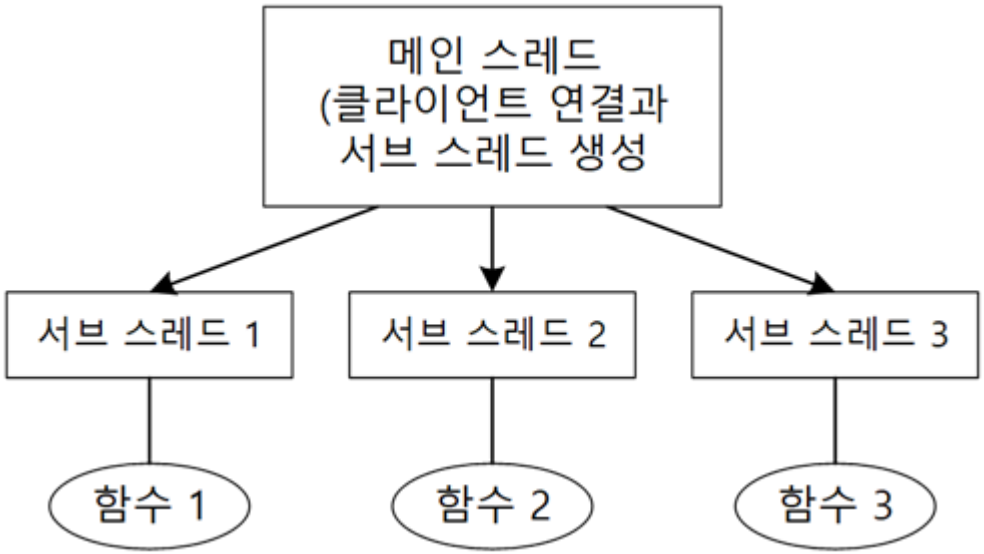
- **병행 서버(concurrent server)**
  - 다수의 클라이언트를 동시에 서비스하는 서버
  - 병행 서버 구현 방법
    - 스레드(Thread) 방식
      - 클라이언트 마다 별도의 스레드 사용: 멀티스레드
    - 이벤트 구동(Event driven) 방식
      - 다른 작업을 진행하다가 소켓 관련 이벤트가 발생하면 이벤트를 처리하는 방식

멀티스레드 소켓 프로그래밍

- **프로세스와 스레드**
  - 컴퓨터에서 실행 중인 프로그램을 프로세스(process)라 하며
  - 대부분의 경우, 한 개의 프로그램은 한 개의 프로세스로 실행
  - 그러나 한 개의 프로세스는 한 개 이상의 스레드(thread)로 실행될 수 있음
- **스레드(thread)**
  - 프로그램 실행 단위이며 운영체제에 의해 시간이 배분되고 관리됨
  - 프로세스가 시작되면 메인 스레드가 자동 생성되어 실행
  - 프로세스는 메인 스레드 외에 여러 개의 스레드로 나누어 실행
    - 프로세스가 메인 스레드 하나로 실행될 경우, 블로킹(blocking) 함수를 호출하면 함수가 반환될 때까지 실행이 중지되므로 다른 부분을 실행할 수 없음
    - 두 개의 블로킹 함수를 별도의 스레드로 실행하면 적절히 시간 배분이 되어 함수가 모두 실행
  - 스레드 관리는 운영체제가 하므로 사용자 개입이 필요 없음

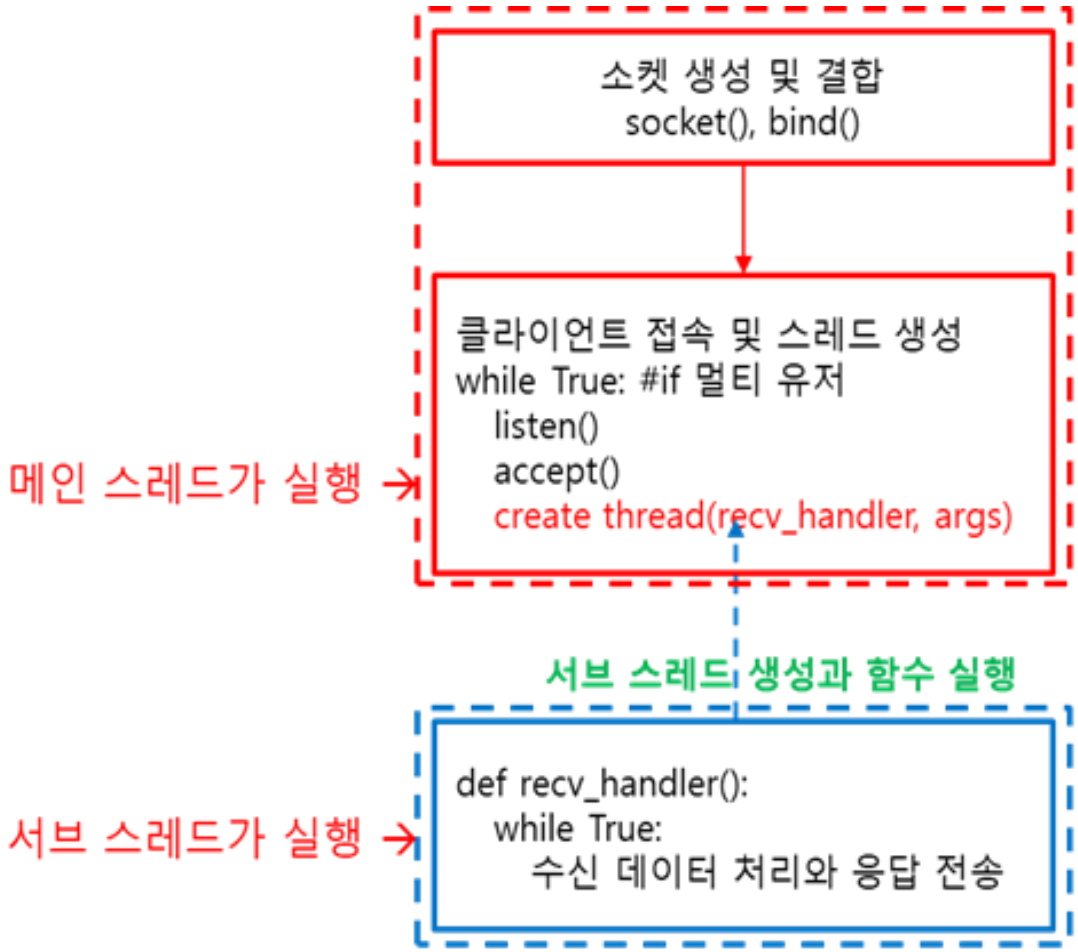
서버 소켓 프로그램에서 스레드의 역할

- 메인 스레드는 클라이언트를 연결하고 서브 스레드를 생성하여 클라이언트의 요청 처리(함수로 구현)을 서브 스레드에게 맡김
- 서브 스레드는 배분된 시간 동안에 할당된 함수를 실행
- 소켓 프로그램은 외부 입출력(데이터 송수신)을 포함하므로 프로그램 실행이 블로킹 될 수 있지만, 데이터 송수신 부분을 다른 스레드로 실행하면 전체 프로세스가 블로킹 되지는 않음



네트워크 프로그램에서 각 스레드의 역할

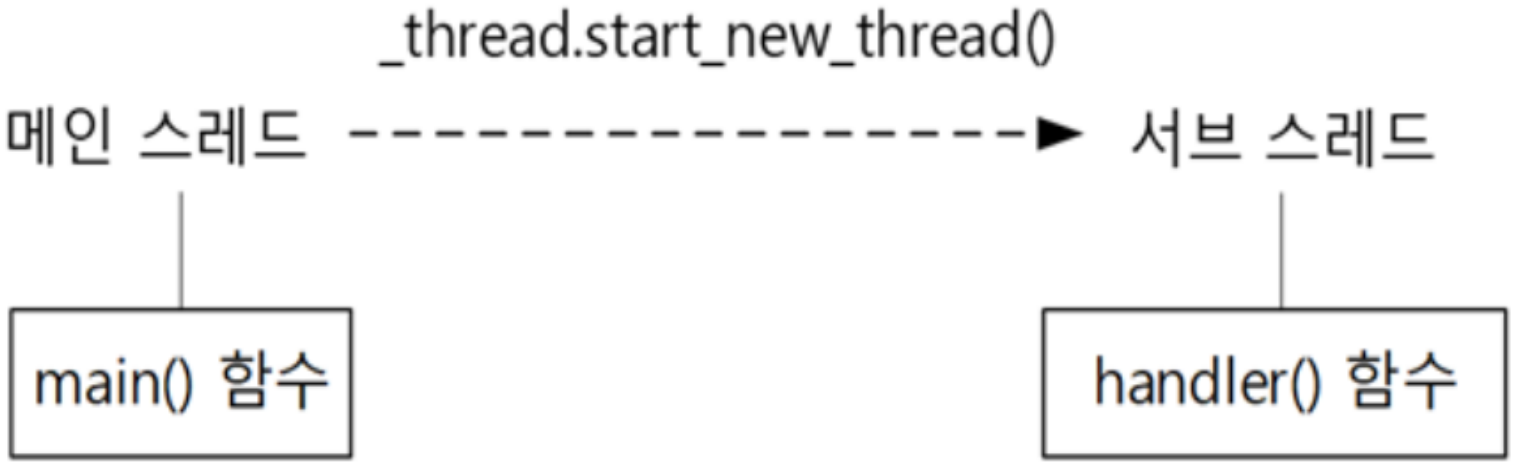
- 메인 스레드는 클라이언트를 연결하고 서브 스레드를 생성하여 클라이언트의 요청 처리(함수로 구현)을 서브 스레드에게 맡김
- 서브 스레드는 배분된 시간 동안에 할당된 함수를 실행
- 소켓 프로그램은 외부 입출력(데이터 송수신)을 포함하므로 프로그램 실행이 블로킹 될 수 있지만, 데이터 송수신 부분을 다른 스레드로 실행하면 전체 프로세스가 블로킹 되지는 않음



<멀티스레드를 이용한 네트워크 프로그램 순서>

_thread 모듈을 이용한 멀티스레드 구현

- 프로그램을 실행하면 메인 스레드가 자동 실행되며, 메인 함수(main())를 실행한다
- 메인 스레드로 실행되는 main() 함수에서 _thread 모듈을 이용하여 서브 스레드를 생성하고 실행한다(fork)
- 서브 스레드 생성과 실행
  - _thread.start_new_thread(handler 함수, (arguments)) 함수로 스레드를 생성하고 실행한다
    - handler 함수 : 서브 스레드로 실행할 함수
    - (arguments) : 함수로 전달할 인수



- handler() 함수
  - 서브 스레드로 실행되며 클라이언트 메시지를 수신하여 처리한다

_thread를 이용한 TCP 서버

- 메시지를 수신하여 출력하고 응답을 클라이언트에게 송신
  - 클라이언트 연결은 **메인 함수**에서 실행
  - 메시지 송수신은 **handler() 함수**에서 처리
  - 이 함수는 **서브 스레드**가 실행

- 서브 스레드 작업

```
def handler(clientsock, addr):
  while True:
    #데이터 수신 및 출력
    #응답 전송
```

- 메인 스레드 작업

```
if __name__ == "__main__":
  # socket을 생성(serversock)하고 클라이언트 접속 대기
  serversock = socket.socket()
  serversock.listen(5)

  while True:
    clientsock, addr = serversock.accept()
        # 서브 스레드 생성 실행
        _thread.start_new_thread(handler, (clientsock, addr))
```

threading 모듈을 이용한 멀티스레드 프로그래밍

1. threading.Thread 클래스의 객체를 정의하고 스레드를 실행하는 방법

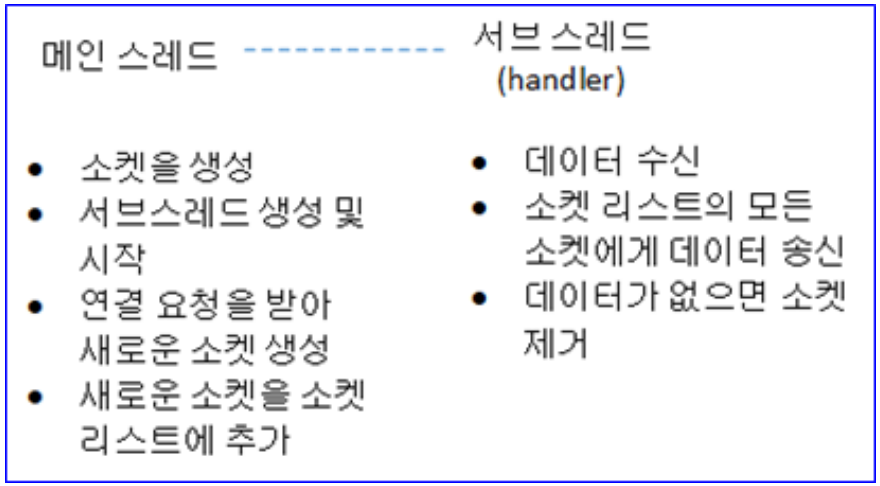
```
th = threading.Thread(target=함수, args=(인수))
th.start()
```

2. threading.Thread 클래스의 파생 클래스를 정의하고 run() 메서드를 재정의 하는 방법
   run() 메서드는 파생 클래스 객체의 start() 메서드를 호출하면 자동 실행된다

```
class subclass(threading.Thread):
    def __init__():
    def run():
th = subclass()
th.start()
```

threading 모듈 사용한 TCP 채팅 서버 프로그램

- 수신된 클라이언트 메시지를 연결된 다른 모든 클라이언트에게 전송
  - 모든 클라이언트가 동일한 메시지를 수신
- **메인 함수**
  - 클라이언트의 연결 요청을 수락
  - 연결된 클라이언트를 연결 목록에 추가
  - 클라이언트와 송수신하는 데이터는 handler() 함수를 이용하여 처리하고 이를 서브스레드로 실행
- **handler() 함수**
  - 수신 데이터를 다른 클라이언트에게 전송
  - 연결이 해제된 클라이언트는 연결 목록에서 제외



threading.Thread의 서브 클래스를 이용한 TCP 에코 서버 프로그램

서브 클래스는 연결된 소켓으로부터 메시지를 받아 출력하고 이를 다시 클라이언트에게 전송

- 서브 클래스 메서드
  - *_init_*()
  - run()

    run() 메서드는 스레드객체.start()를 호출하면 실행된다

threading 모듈을 이용한 GUI 서버/클라이언트 프로그램

클라이언트에서 섭씨온도를 서버로 전송하면 서버에서 화씨로 변환하여 전송하는 GUI(Graphic User Interface) 프로그램

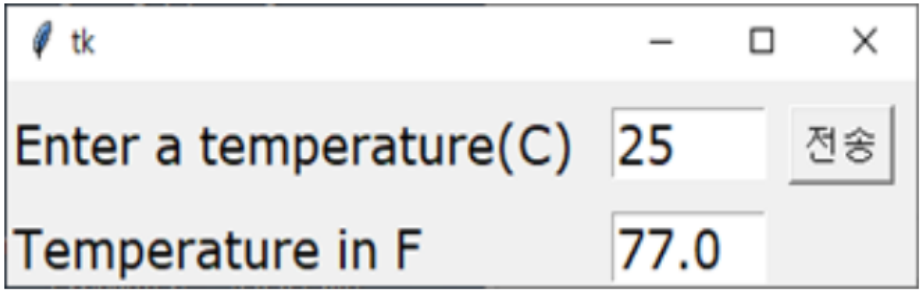
- 서버는 섭씨를 화씨로 변환하여 전송
- 클라이언트 프로그램은 사용자가 섭씨온도를 입력하는 창과 화씨온도를 출력하는 창, 전송 버튼으로 구성

서버 프로그램(콘솔 모드)

수신한 섭씨 온도를 float형으로 변환한 다음, 화씨온도를 계산하여 송신한다.

GUI 클라이언트

- 섭씨 온도 창에 온도를 입력하고 [전송] 버튼을 클릭하면 섭씨 온도를 서버로 전송
- 서버로부터 받은 화씨 온도를 화씨 온도 창에 표시한다
  - 화씨 온도 창에 표시하는 것은 tkinter 루프(mainloop)가 처리하지 않으므로 스레드로 처리해야 한다



threading 모듈을 이용한 TCP 멀티 채팅 GUI 프로그램(서버)

- 클라이언트가 보내온 메시지를 다른 클라이언트에게 중계하여 클라이언트 간 채팅 가능
- 클래스를 정의하고 accept_client, receive_message, send_all_clients 메서드를 포함

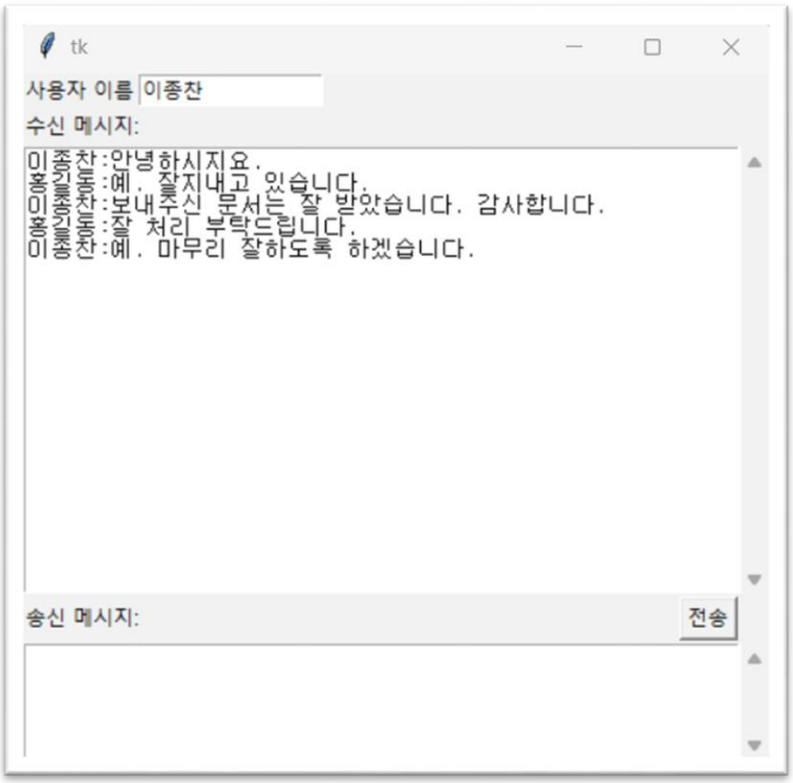
```
class ChatserverMulti:
    def __init__(self):
        #소켓 생성
        #연결 대기
        #accept_client()를 호출하여 클라이언트와 연결

    #연결을 수락하고 서브 스레드를 생성하여 실행
    def accept_client(self):
      while True:
        #연결 수락
        #연결 소켓을 목록에 추가
        #메시지 수신 스레드 생성/실행(receive_message 함수)

    #메시지를 수신하고 모든 클라이언트에게 전송
    def receive_message(self, sock):
      while True:
        #메시지 수신
        #send_all_clients() 호출

    #발신자를 제외한 모든 연결 소켓으로 메시지 전송
    def send_all_clients(self, senders_socket):
```

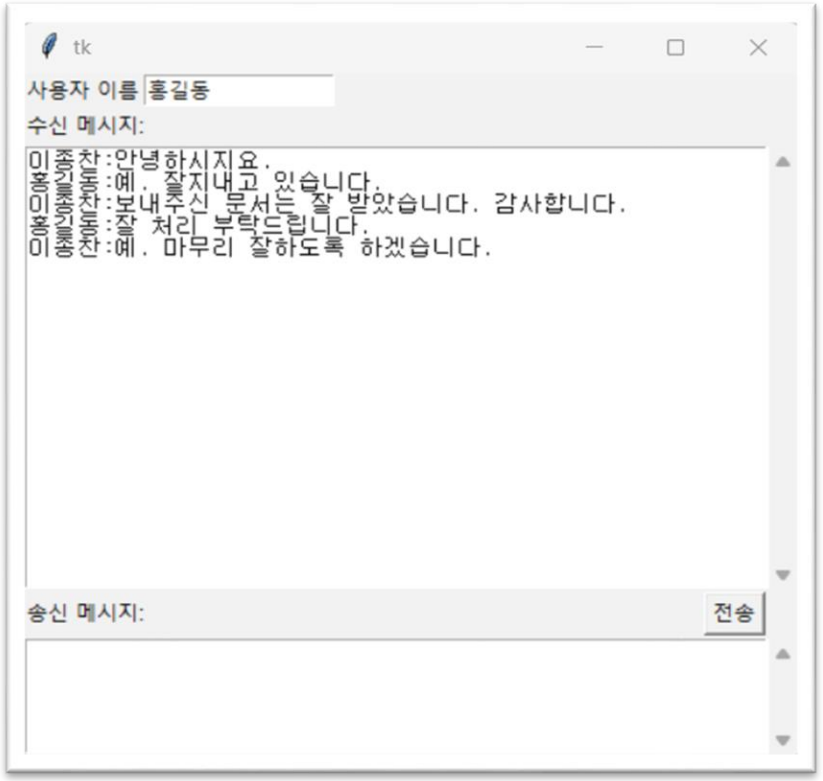
threading 모듈을 이용한 TCP 멀티 채팅 GUI 프로그램(클라이언트)

- ChatClient 클래스의 메서드
  - initialize_socket()
    소켓을 생성하고 서버로 연결
  - initialize_gui()
    5개의 프레임을 이용하여 사용자 화면 구성
    - 프레임 1: 사용자 이름 라벨(name_label) + 이름 입력 창(name_widget, Entry)
    - 프레임 2: 수신 메시지 라벨(recv_label)
    - 프레임 3: 수신 메시지 창(chat_transcript_area, ScrolledText)
    - 프레임 4: 송신 메시지 라벨(send_label) + [송신] 버튼(send_btn, Button)
    - 프레임 5: 송신 메시지 창(enter_text_widget, ScrolledText)
  - send_chat()
    송신 메시지 창에서 메시지를 읽어 수신 메시지 창에 표시
  - receive_message()
    소켓에서 메시지를 읽어 수신 메시지 창에 표시
  - listen_thread()
    receive_message()를 실행하는 스레드

threading 모듈을 이용한 TCP 멀티 채팅 GUI 프로그램(클라이언트)

- GUI 환경에서 전송 데이터와 수신 데이터를 표시
- 서용자 이름 입력창, 송신 메시지 입력창, 수신 메시지 출력 창, [전송] 버튼을 클래스로 구현





threading 모듈을 이용한 UDP 채팅 프로그램

- **UDP 채팅 서버**
  - 수신 메시지를 발신자를 제외한 다른 클라이언트에게 송신
  - 상대방이 “quit” 메시지를 전송하면 채팅을 종료
  - 연결이 종료되었을 때 recvfrom()을 호출하면 예외가 발생할 수 있으므로 recvfrom()은 try-except 구문 속에 포함되어야 함
- **UDP 채팅 클라이언트**
  - 송신 메시지를 입력받아 전송하고 수신 메시지는 화면에 표시
  - 메시지를 수신하여 화면에 출력하는 기능을 스레드로 작성하여 실행하면 송신 메시지 입력 중이라도 수신 메시지를 표시 가능

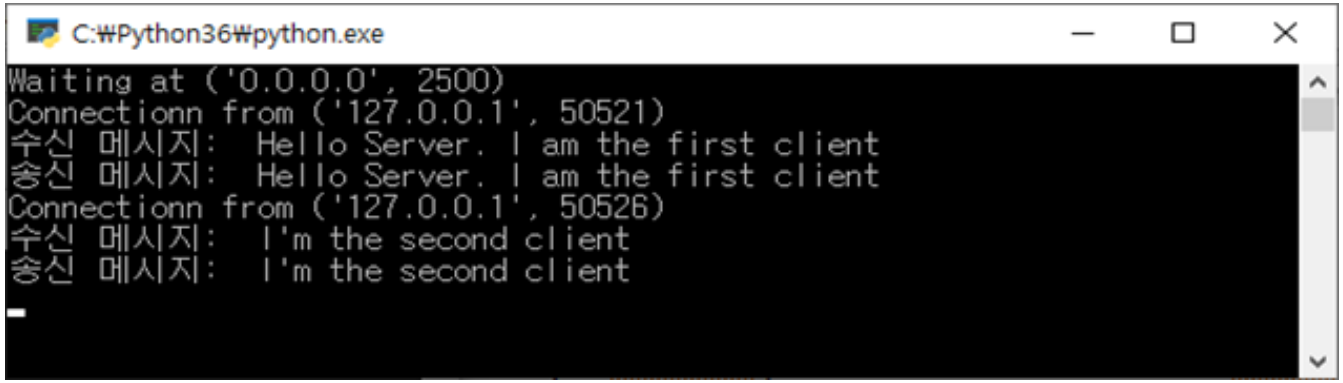
concurrent.future 모듈을 이용한 멀티 에코 프로그램

- concurrent.futures 모듈의 ThreadPoolExecutor 클래스를 이용하여 멀티스레드를 구현
  - ThreadPoolExecutor 클래스는 스레드 풀(Thread Pool)에 등록된 함수를 비동기적으로 실행하는 Executor 클래스의 서브 클래스
  - Executor의 submit() 메서드를 이용하여 실행 함수를 스레드 풀에 등록
  - submit() 메서드로 등록된 함수는 스레드로 실행

```
import concurrent.futures as cf
❶ def receive_message(sock, addr): #스레드로 실행할 함수
.........

❷ with cf.ThreadPoolExecutor(max_workers=10) as th #스레드 풀 생성
❸     th.submit(receive_message, sock, addr) #실행 함수를 스레드 풀에 등록
```

멀티 에코 서버 프로그램

- 다수의 클라이언트를 서비스하는 에코 서버 프로그램
  - receive_message(sock, address)
    - 메시지를 수신하여 화면에 출력하고 다시 전송하는 함수
    - 무한 루프로 실행되며, 연결이 해제되면 종료
  - 메인 함수
    - 최대 스레드 수(max_works)를 10으로 설정하여 스레드 풀을 생성
    - 클라이언트 연결을 수락하고 처리 함수(receive_message)를 스레드 풀에 등록
    - 등록된 처리 함수는 스레드에 의해 계속 서비스 된다



멀티프로세스 서버 프로그램

- 멀티스레드는 단일 프로세스로 실행되며 코드, 데이터, 힙(heap) 메모리를 공유하지만, 멀티프로세스 프로그램에서 각 프로세스는 각자의 메모리 영역을 가짐
  - 멀티프로세스 장점
    - CPU 코어를 최대로 활용하여 처리 속도가 빠르다
  - 멀티프로세스 단점
    - 프로세스마다 별도 자원을 가지므로 프로세스를 스위칭할 때 상당한 부담이 발생

- main process와 child processes로 구성
- main process는
  if __name__ == "__main__": 으로 구성
- child process는 main process 내에서 생성하고 시작한다

```
from multiprocess import  Process
def handler(arg):
    .....
p=Process(target=handler,  args=(arg,)) #프로세스 생성
p.start() #child process 시작
p.join() #child process 종료 대기
```

멀티프로세스 서버 프로그램

- c_sock.close()
  - 부모 프로세스에서는 클라이언트 소켓을 닫아야 함
  - 자식 프로세스가 이미 c_sock의 복사본을 가지고 사용하므로, 부모는 더 이상 이 소켓을 유지할 필요가 없음
  - 이를 닫지 않으면 불필요한 리소스 누수가 발생할 수 있음