# select 모듈을 이용한 소켓 프로그래밍

- 데이터의 입출력은 비동기적으로 발생하는데, 메인 프로세스에서 입출력의 발생을 기다렸다 처리하면 처리 시간보다 대기 시간이 훨씬 많아진다.
  - 따라서 입출력을 비동기적으로 처리하면 효율적 프로그래밍이 가능하다.
- 비동기적 입출력
  - 호출된 입출력 함수는 반환되고 다른 프로세스(작업)을 수행하다가 나중에 입출력을 처리하는 방법 사용
- 다른 작업을 수행하는 중에 비동기 입출력의 발생을 알 수 있는 방법이 필요
  - select 모듈의 select() 함수
    - 입출력이 발생할 수 있는 소켓 목록을 인수로 지정하고 select()를 호출하면 select()는 입출력을 기다리지 않고 반환
    - 프로그램이 다른 일을 처리하는 동안 select() 함수는 읽기, 쓰기, 오류 이벤트가 발생한 소켓을 소켓 목록에 저장
      - select() 함수의 인수로 소켓 리스트를 전달하면 이벤트 발생 소켓 리스트를 반환 받음.
    - 다른 일의 처리가 끝나면 입출력이 발생한 이벤트 발생 소켓 리스트를 조사하여 처리

# select 모듈을 이용한 서버 프로그램

- 이벤트 발생 소켓 조사 방법

```
r_sock, w_sock, e_sock = select.select(s_list1, s_list2, s_list3, timeout)
```

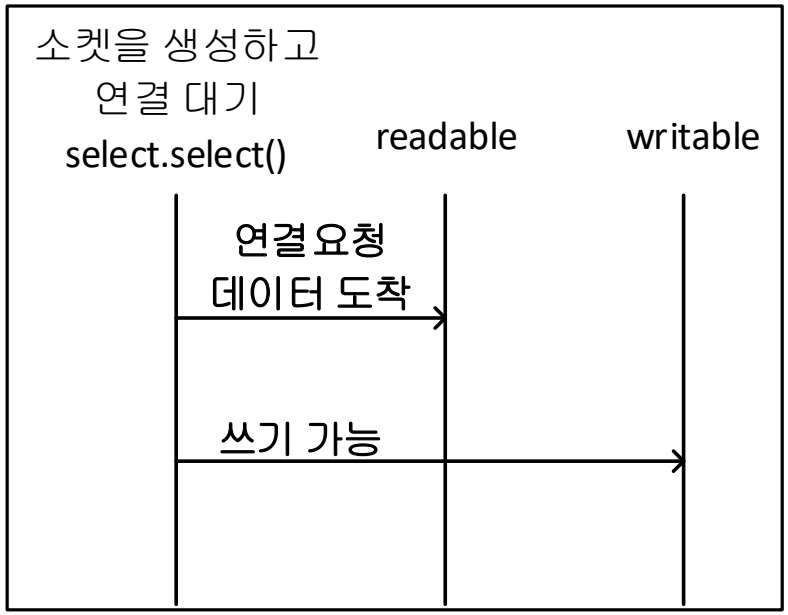
> r_sock : 읽기 이벤트 발생 소켓 목록
> w_sock : 쓰기 이벤트 발생 소켓 목록
> e_sock : 오류 이벤트 발생 소켓 목록

> s_list1 : 읽기 이벤트 조사 소켓 목록
> s_list2 : 쓰기 이벤트 조사 소켓 목록
> s_list3 : 오류 이벤트 조사 소켓 목록

- 각 소켓 리스트에 저장된 소켓에서「읽기」,「쓰기」,「오류」 이벤트가 발생하면 해당 소켓 목록이 r_sock, w_sock, e_sock 리스트에 저장되어 반환됨
  - 반환 소켓 리스트를 조사하여 데이터 송수신을 처리하면 블로킹없이 송수신이 가능
- timeout을 지정하지 않으면 블로킹 모드로 동작.
  - timeout을 지정하면 이벤트가 발생하거나 timeout 후 select()가 반환됨.
  - timeout을 0으로 설정하면 넌블록킹 모드로 동작.

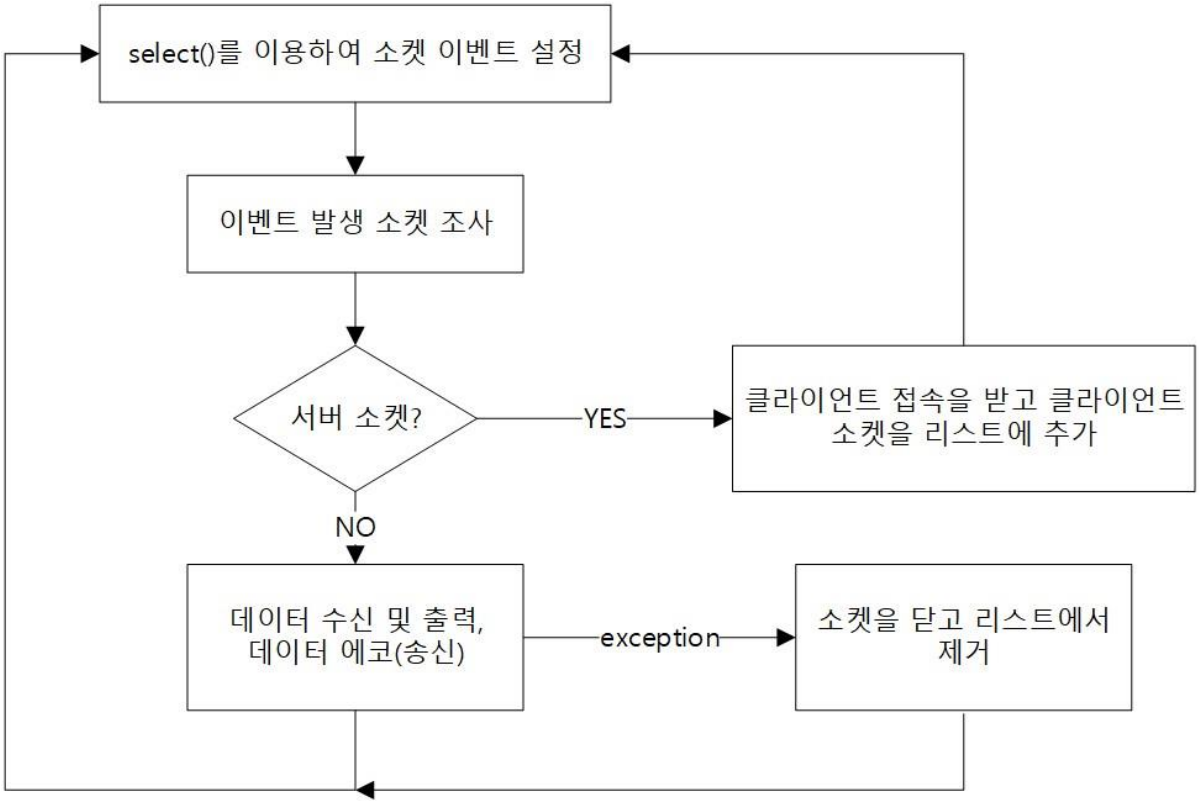
# select() 함수를 이용한 서버 프로그램의 동작 순서

- 서버 소켓을 생성하고 클라이언트의 연결 요청을 기다리는 listen() 함수를 호출 후, select.select()를 호출한다.
- 함수 호출 후의 동작
  - 클라이언트의 연결 요청이나 데이터가 도착하는 이벤트가 발생하면 r_sock에 연결 요청 또는 데이터 도착 소켓 목록 을 저장한다.
  - 쓰기 가능 이벤트가 발생하면 w_sock에 쓰기 가능한 소켓 목록이 저장된다.
    - r_sock의 내용을 조사하면 읽기 가능 이벤트가 발생한 소켓을 알 수 있고, w_sock의 내용을 조사하면 쓰기 가능 이벤트가 발생한 소켓을 알 수 있다.



# select() 함수를 이용한 에코 서버의 동작 순서

- 읽기 가능 이벤트만 사용한다.
- select()가 반환한 소켓 목록을 조사하여 서버 소켓에서 이벤트가 발생했으면 연결 요청 이 벤트이므로 연결된 클라이언트 소켓을 목록에 추가한다.
- 클라이언트 소켓에서 이벤트가 발생했으면 데이터를 수신하여 출력하고 재전송한다. 입출력이 비동기적으로 처리되므로 다중 클라이언트를 서비스할 수 있다.



# select() 함수를 이용한 클라이언트 프로그램

- select 모듈을 이용하여 클라이언트를 구현한다.
  - 블로킹 모드에서 데이터 수신을 대기하는 것이 아니라 수신 데이터가 없으면 송신 등 다른 일을 처리할 수 있다
  - select 함수를 호출할 때 timeout=0으로 설정하면 넌블로킹 모드로 동작

# select 모듈을 이용한 GUI 소켓 클라이언트 프로그래밍

- tkinter의 mainloop() 함수가 처리하지 못하는 소켓 이벤트를 select() 함수를 이용하여 GUI에서 처리하는 프로그램
  - [ON]/[OFF] 버튼을 클릭할 때마다 색상이 변하고 메시지(ON 또는 OFF)를 에코 서버로 전송한다
  - 에코 서버 메시지를 수신하여 라벨로 표시한다
  - mainloop()가 실행되면 메시지 도착 이벤트는 더 이상 처리하지 못한다
  - tkinter 모듈의 after() 함수를 이용하여 소켓 이벤트를 주기적으로 처리한다
    - after(delay_time, method)
      - delay_time 경과 후 method를 주기적으로 실행
- 프로그램 구성

```
handle()
# 주기적으로 메시지를 수신하여 라벨로 표시하는 함수
# after(delay_time, handle)

button_command()
# 버튼 클릭 콜백 함수

main 함수
# 소켓을 만들고(전역 변수) 화면을 구성한다
# handle() 호출
```

# select와 queue 모듈을 이용한 소켓 프로그래밍

- 큐 모델은 멀티스레드 프로그램에 필요한 임시 데이터 장소이다.
  - FIFO 큐 : 먼저 저장된 데이터가 먼저 인출되는 방식
  - LIFO 큐 : 마지막으로 저장된 데이터가 먼저 인출되는 방식
  - 우선순위 큐: 저장된 데이터를 분류하여 가장 값이 낮은 데이터가 먼저 인출되는 방식
- 종류에 따라 큐를 생성하는 클래스는 다음과 같다.

```
queue.Queue()           FIFO 큐 생성
queue.LifoQueue()       LIFO 큐 생성
queue.PriorityQueue()   우선순위 큐 생성
```

# 큐에 데이터 저장 및 인출

- queue.Queue()를 사용하여 선입선출 데이터 큐를 만들 수 있음.
- Queue 클래스의 put() 메서드를 사용하여 데이터를 큐에 저장하고, get() 메서드를 사용하여 큐에서 데이터를 인출한다.
- 큐를 이용하면 멀티 클라이언트의 요청을 효율적으로 처리 가능
  - 클라이언트의 요청이 들어오면 일단 요청 큐에 저장하고 시간적 여유가 있을 때 요청 큐에 저장된 요청을 처리하여 응답 큐에 저장한다.
  - 저장된 응답은 나중에 송신한다. 처리에 시간이 많이 소요되므로 우선순위를 정하여 할 수도 있고, 간단한 처리를 먼저 할 수도 있다.

```
fifo_q = queue.Queue()  # FIFO 큐 생성
data = ['Hello', 'World', 'Python', 'Korea']
for value in data:    # 큐에 저장
    fifo_q.put(value)

while not fifo_q.empty():  # 큐에 저장된 값이 있는지 확인
    print(fifo_q.get())    # 저장된 데이터를 인출하여 출력
```

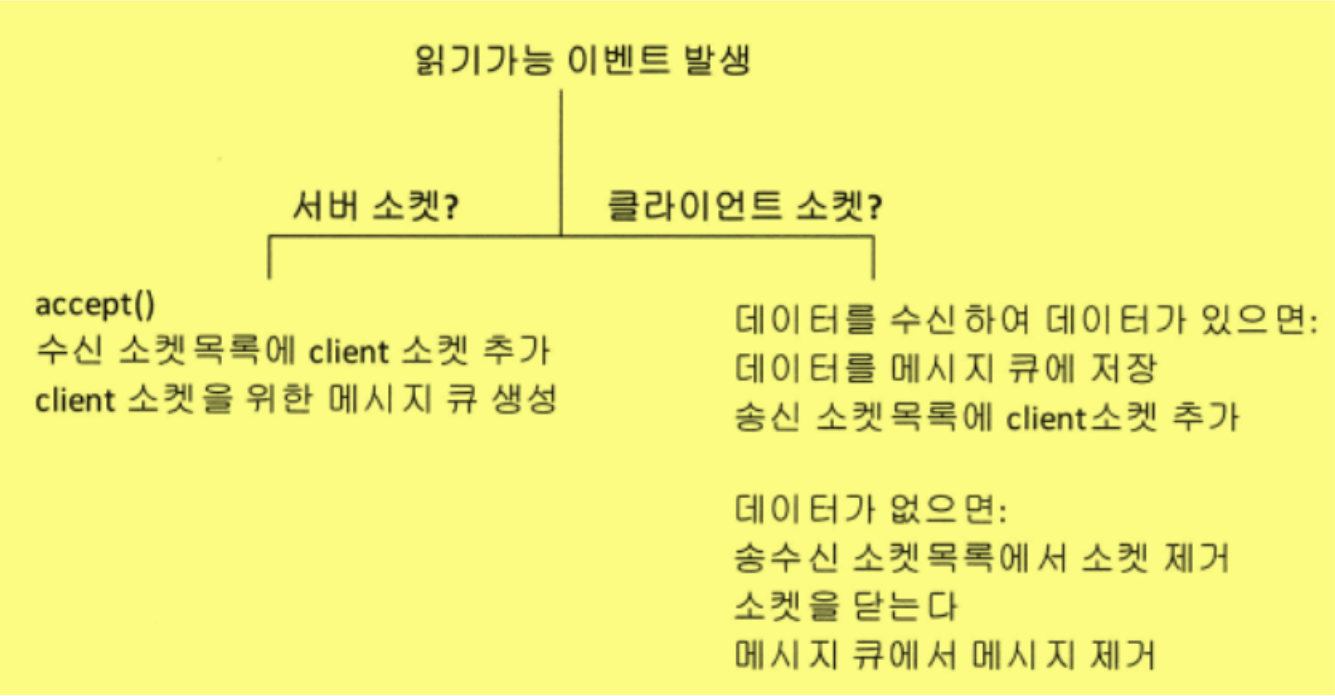
# select와 queue 모듈을 이용한 에코 서버 프로그래밍

- 큐를 사용하여 다중 클라이언트의 데이터를 받아 재전송하는 에코 서버이다. 데이터가 수신되면 큐에 저장하고, 소켓이 쓰기 가능 상태가 되면 큐에 저장된 메시지를 전송한다.
- 소켓 목록
  - r_sock[]는 요청이나 데이터를 수신할 수 있는 소켓의 목록이다. 서버 소켓을 생성하여 r_sock[]에 저장. 클라이언트와 연결되면 클라이언트 소켓도 이 목록에 저장. 연결이 해제되면 해당 소켓은 목록에서 제외
  - w_sock[]는 데이터를 전송할 소켓의 목록이다.
  - msgQueue는 key가 소켓이고, value가 미시지 큐인 딕셔너리 변수이다.

```
r_sock[] : 수신 소켓 목록
w_sock[] : 송신 소켓 목록
msgQueue{} : key: 소켓, value:전송 데이터 큐
```

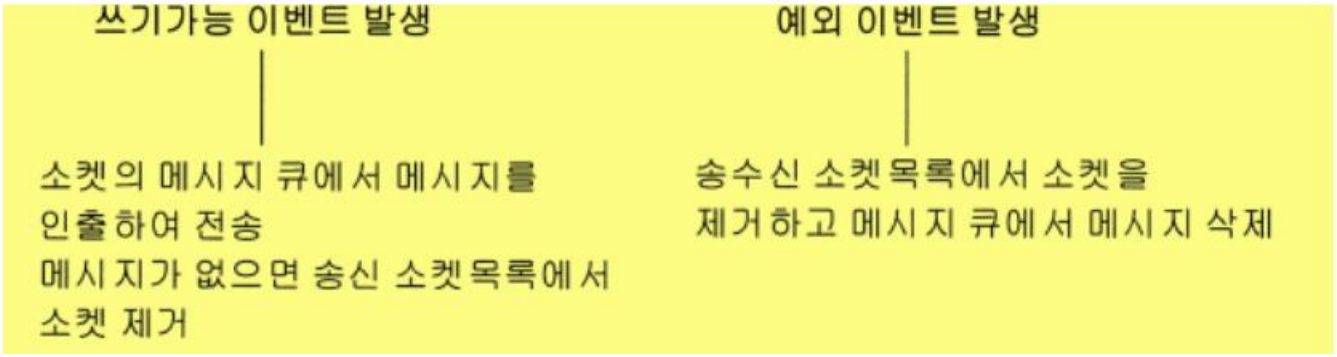
# select()와 queue를 이용한 read 동작

- select() 함수가 읽기 가능 소켓을 반환했을 때 이벤트 발생 소켓이 소버 소켓인 경우와 클라이언트 소켓인 경우의 처리를 나타낸다.



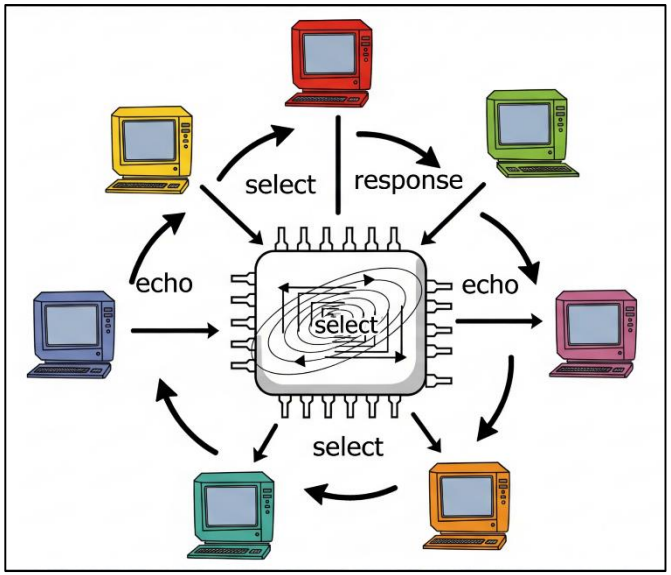
# select()와 queue를 이용한 write와 exception 처리

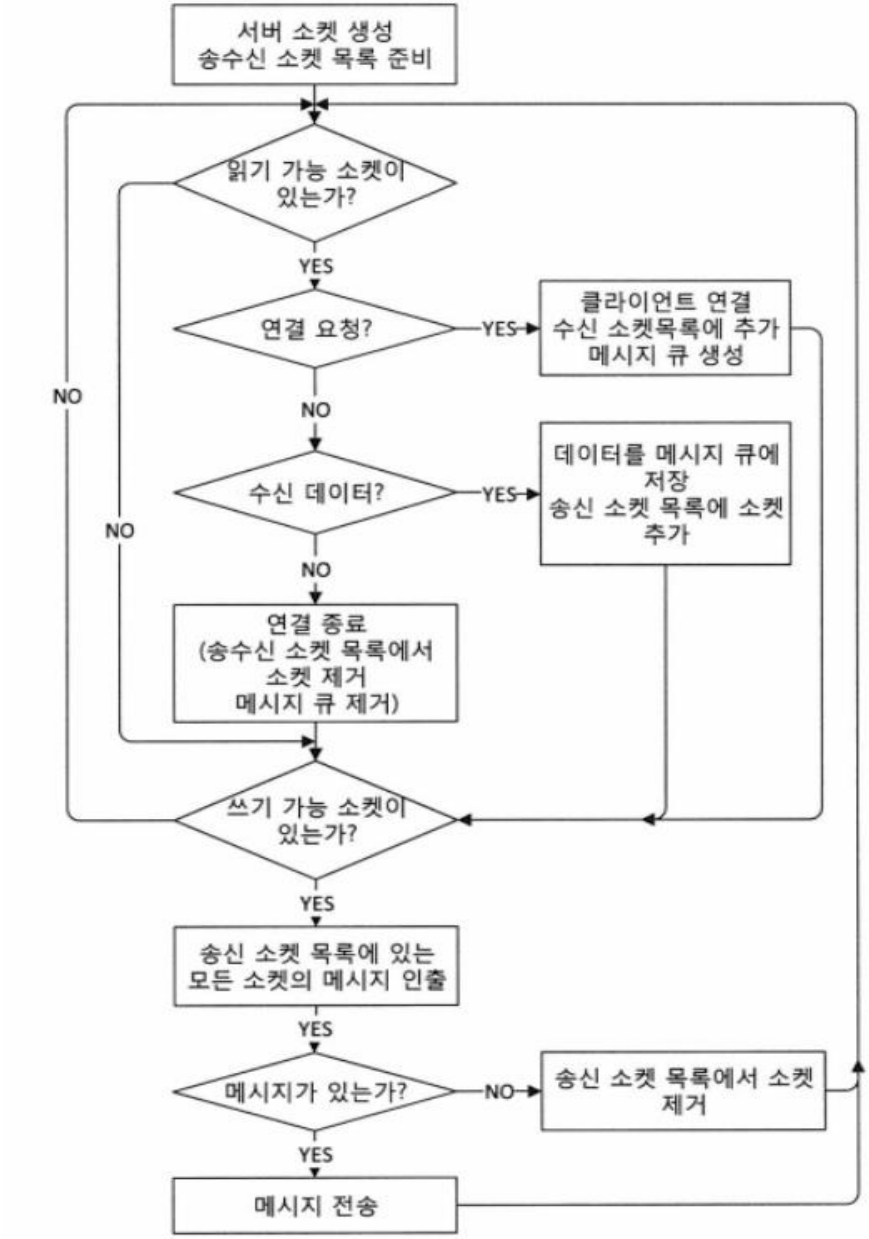
- 쓰기 가능 이벤트와 오류 이벤트에 대한 처리를 나타낸다.



# 에코 서버 프로그램의 순서도

- 예외 이벤트는 생략하고 읽기와 쓰기 이벤트만을 표시한다.



# socketserver 모듈을 사용한 서버 프로그래밍

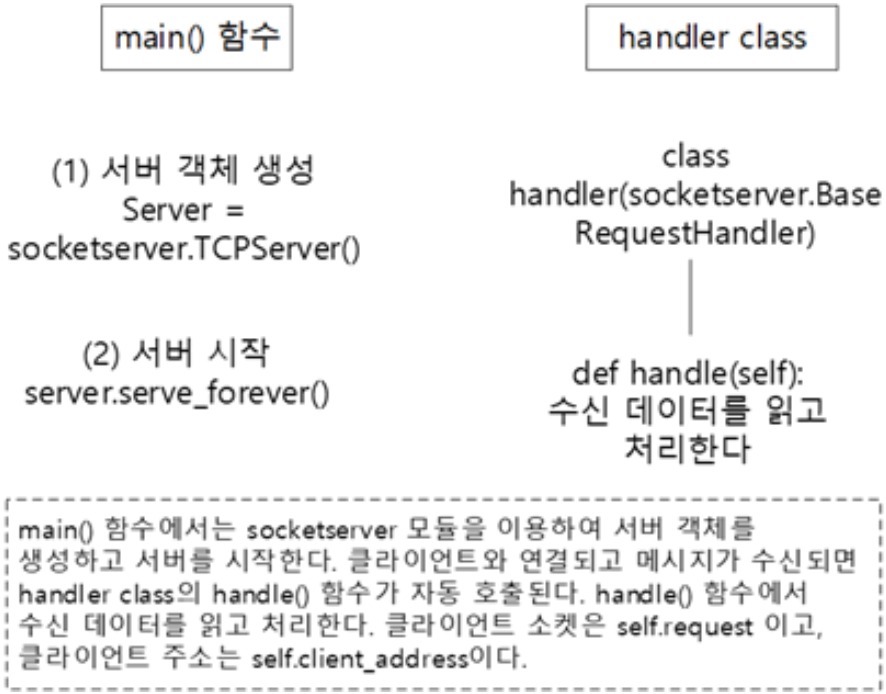
- socketserver 모듈은 비동기 서버 프로그램을 작성하기 위한 모듈이며, 서버를 만들기 위하여 **서버 객체와 핸들러 클래스**가 필요.
- socketserver 모듈을 이용하여 TCP 또는 UDP 서버를 만들 수 있음
  - TCP 서버 객체 생성: socketserver.TCPServer 클래스 이용
  - UDP 서버 객체 생성: socketserver.UDPServer 클래스 이용
- 핸들러 클래스(=이벤트 클래스)
  - **클라이언트와의 연결 이후에 발생하는 송수신과 같은 이벤트를 처리함**
  - TCP 서버 객체와 UDP 서버 객체를 생성할 때 지정
  - socketserver 모듈의 BaseRequestHandler 클래스에서 상속받음
  - 재정의(redefinable) 가능 메서드 임
    - setup(): 연결되면 실행되는 메서드
    - **handle(): 데이터를 수신하고 처리한 후 응답을 생성하고 전송**
      - 데이터 송수신을 위해 필요한 소켓과 상대방 주소는 BaseRequestHandler 클래스의 request와 client_address 인스턴스 변수로 정의되어 있음.
    - Finish(): 연결이 해제되면 실행되는 메서드, setup()에서 처리한 내용을 지움.

# socketserver 모듈의 동작

⊖ TCPServer 서버 소켓 객체를 생성하고 주소 정보와 handle() 함수가 정의된 핸들러 클래스를 인수로 전달
  - server = socketserver.TCPServer((ip, port), HandlerClass)

⊖ serve_forever()를 호출하여 서버 실행
  - server.serve_forever()

⊛ BaseRequestHandler의 파생 클래스(HandlerClass)에서 handle() 메소드 재정의하여 소켓 이벤트를 처리한다



# socketserver 모듈을 사용한 TCP 서버 프로그램

- socketserver.TCPServer 클래스를 이용하여 서버 객체를 생성하고 핸들러 클래스를 정의하여 작성
  - server_address: (host, port)
  - RequestHandlerClass: 핸들러 클래스
    - 클라이언트에서 데이터가 도착하면 핸들러 클래스의 handle() 메서드가 실행
    - handle() 메서드 주요 인자
      - self.request: 클라이언트 소켓
      - self.client_address: 클라이언트 주소와 포트(ip, port)
  - bind_and_activate: True이면 연결과정이 자동 처리됨

```
class socketserver.TCPServer(server_address, RequestHandlerClass,
                             bind_and_activate=True)
```

# socketserver 모듈을 사용한 UDP 서버 프로그램

- socketserver.UDPServer 클래스를 이용한 에코 서버 프로그램
  - UDP 서버는 연결이 없기 때문에 클라이언트의 연속 서비스가 가능
  - UDP 서버에서 self.request 튜플 인수턴스 변수의 기능 (TCP 서버와 다름)
    - self.request[0]: 수신 데이터
    - self.request[1]: 클라이언트 소켓
    - client_address: 상대방 주소

```
class socketserver.UDPServer(server_address, RequestHandlerClass,
                             bind_and_activate=True)
```

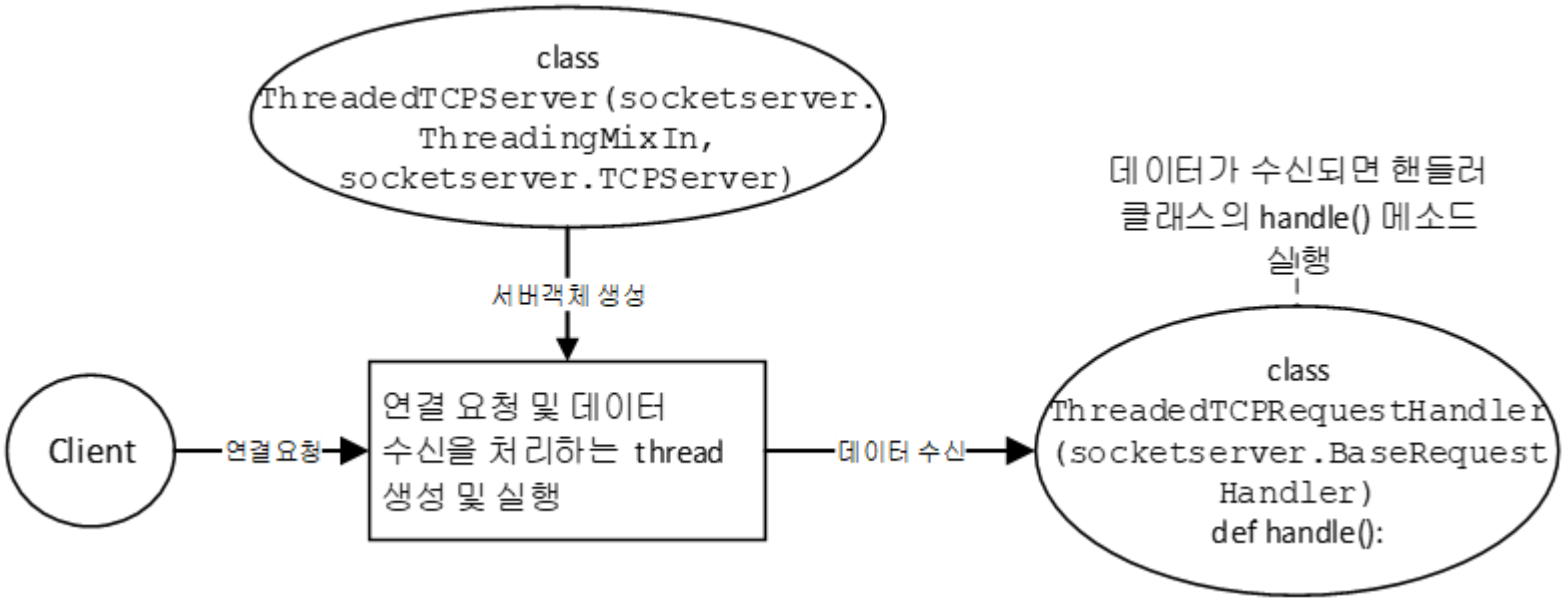
# ThreadingMixIn 클래스를 이용한 멀티 클라이언트 지원 서버

- socketserver.TCPServer 클래스의 문제점
  - 단일 스레드로 실행되므로 한 번에 하나의 클라이언트만 서비스
- 멀티 클라이언트를 서비스를 위하여
  - socketserver.ThreadingMixIn 클래스와 TCPServer 또는 UDPServer 클래스를 부모 클래스로 갖는, 파생 클래스를 정의하여 사용

```
class ThreadedTCPServer(socketserver.ThreadingMixIn,
                        socketserver.TCPServer):
    pass
```

  - TCP 서버를 생성하려면 TCPServer 클래스를 부모 클래스로 함께 지정하고, UDP 서버를 생성하려면 UDPServer 클래스를 부모 클래스로 함께 지정함.
  - ThreadedTCPServer의 기능
    - TCPServer의 일반적인 서버 기능을 상속 받음
    - ThreadingMix In이 제공하는 각 클라이언트 요청에 대한 스레드 생성 및 관리 기능을 상속 받음

# ThreadingMixIn 클래스를 이용한 멀티 클라이언트 서버 동작




- ThreadedTCPServer 파생 클래스는 서버 객체를 만들고, 클라이언트 연결을 요청을 받아 처리할 수 있도록 mother socket thread를 생성하고 실행
  - 서버가 연속적인 연결 서비스를 할 수 있도록 server.serve_forever() 함수를 실행

```
server = ThreadedTCPServer((HOST, PORT), ThreadedTCPRequestHandler)
server_thread = threading.Thread(target=server.serve_forever)
server_thread.start()
```

# ThreadingMixIn 클래스를 이용한 멀티 클라이언트 서버 동작

- socketserver.ThreadingMixIn은 이름 그대로 믹스인(Mixin) 클래스로서, TCPServer의 기본 동작에 스레드 기반의 동시 처리 기능을 "혼합"하는 역할
- ThreadedTCPServer가 socketserver.ThreadingMixIn을 상속받으면서, 이 서버는 클라이언트 요청이 들어올 때마다 자동으로 새로운 스레드를 생성하여 처리하는 기능을 가짐

```
class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
```

- server.serve_forever() 메서드가 server_thread(mother socket 역할)에서 실행될 때, 이 메서드는 클라이언트의 연결 요청을 계속해서 감지
- 새로운 클라이언트가 연결될 때마다 socketserver.ThreadingMixIn에 의해 내부적으로 새로운 스레드가 생성되고, 이 스레드에서 ThreadedTCPRequestHandler 클래스의 handle() 메서드를 호출하여 해당 클라이언트의 요청을 처리

```
server = ThreadedTCPServer((HOST, PORT), ThreadedTCPRequestHandler)
server_thread = threading.Thread(target=server.serve_forever)
server_thread.start()
```

# socketserver의 멀티스레드 클래스를 이용한 에코 서버

- 멀티스레드 서버를 위한 서버 객체 생성 클래스
  - class socketserver.ThreadingTCPServer
  - class socketserver.ThreadingUDPServer
- 요청 처리를 위한 핸들러 클래스(BaseRequestHandler 클래스의 파생 클래스)
  - socketserver.StreamRequestHandler
  - socketserver.DatagramrequestHandler
- 핸들러 클래스에서 데이터 수신과 송신 방법
  - self.rfile.read() 또는 self.rfile.readline() 함수를 이용하여 수신
  - self.wfile,write() 함수를 이용하여 송신

# ThreadingMixIn과 http.server 모듈을 이용한 감시 카메라 구현

- 컴퓨터에 웹 카메라를 설치하고 촬영된 영상을 웹브라우저에 표시하여 감시 카메라를 구현하는 프로그램
- 이 프로그램은 서버 객체를 생성하기 위한 팩토리 클래스와 이벤트 처리를 위한 프로토콜 클래스가 필요
  - 팩토리 클래스(Factory Class) : StreamingServer
    - 다중 클라이언트를 지원하기 위해 socketserver.ThreadingMixIn과 http.server.HTTPServer 클래스를 상속하여 정의하며 서버 객체 생성을 위해 사용
  - 프로토콜 클래스(Protocol Class) : StreamingHandler
    - http.server.BaseHTTPRequestHandler를 상속하여 정의하며 클라이언트 요청을 처리하는 핸들러 클래스
    - 웹 브라우저에서 영상을 제공하는 서비스를 구현, 특히 MJPEG(Motion JPEG) 방식으로 웹 브라우저에 실시간 비디오 스트림을 전송하는 기능을 담당

# ThreadingMixIn과 http.server 모듈을 이용한 감시 카메라 구현

- 팩토리 클래스인 StreamingServer의 서버 객체 server를 생성
- 서버 주소는 (‘’, 8000)이고 클라이언트의 요청은 프로토콜 클래스에서 처리

```
#서버 객체 생성을 위한 팩토리 클래스
class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
allow_reuse_address = True    # socketserver 속성. 주소 재사용
daemon_threads = True   # ThreadingMixIn 속성. 서브 스레드가 종료되면 즉시 종료

address = ('', 8000)
server = StreamingServer(address, StreamingHandler)    # server 객체를 생성
server.serve_forever()
```

# ThreadingMixIn과 http.server 모듈을 이용한 감시 카메라 구현

- BaseHTTPRequestHandler는 파이썬의 라이브러리 http.server 모듈에 포함된 클래스로서 HTTP 요청의 파싱과 처리에 필요한 기능을 제공
- 클라이언트로부터 HTTP 요청이 들어오면, BaseHTTPRequestHandler는 요청 라인을 파싱하여 속성들을 인스턴스 변수로 저장
  - 요청 라인: http://loalhost:8000
  - self.command: HTTP 메서드 (예: "GET", "POST")
  - self.path: 요청된 URL 경로 (예: "/index.html", "/stream.mjpg")
  - self.request_version: HTTP 버전 (예: "HTTP/1.0", "HTTP/1.1")

```
class StreamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
```

- do_GET 메서드가 호출될 때 self.path는 이미 클라이언트가 요청한 URL의 경로 부분을 담고 있음.
  - do_GET 메서드는 클라이언트로부터 GET 요청이 왔을 때 호출되는 함수. 요청된 URL(self.path)에 따라 다른 동작을 수행.
  - self.path는 http.server.BaseHTTPRequestHandler 클래스에서 제공하는 속성
- self.path 값을 읽어서 조건문(if self.path == '/') 등을 통해 요청된 경로에 따라 적절한 동작을 수행

# ThreadingMixIn과 http.server 모듈을 이용한 감시 카메라 구현

- <img src="/stream.mjpg" />
  - 클라이언트의 요청: 웹 브라우저가 HTML 페이지를 로드할 때, 해당 페이지에 <img src="/stream.mjpg" />와 같은 태그가 있다면, 브라우저는 이 src 속성에 지정된 URL 경로로 서버에 HTTP GET 요청을 보냄
  - 요청 URL은 대략 http://[서버_주소]:[포트번호]/stream.mjpg와 같은 형태
  - BaseHTTPRequestHandler는 이 요청 라인에서 /stream.mjpg 부분을 추출하여 self.path라는 인스턴스 변수에 자동으로 할당
- multipart/x-mixed-replace; boundary=FRAME 헤더
  - 브라우저에게 여러 장의 이미지를 연속적으로 보낼 것이고, 각 이미지는 'FRAME'이라는 문자열로 구분되고 새 이미지가 오면 기존 이미지를 대체(replace)하라고 알림
- **self.wfile.write(frame)**
  - **self.wfile은** 클라이언트와의 네트워크 연결(소켓)에 연결된 파일과 유사한(file-like) 객체
  - 일반 파일에 데이터를 쓰는 것처럼, 이 self.wfile 객체에 데이터를 쓰면 그 데이터가 네트워크를 통해 클라이언트에게 전송
  - 이 frame 데이터를 받아서 <img src="/stream.mjpg"> 태그가 있는 위치에 실시간으로 이미지를 업데이트