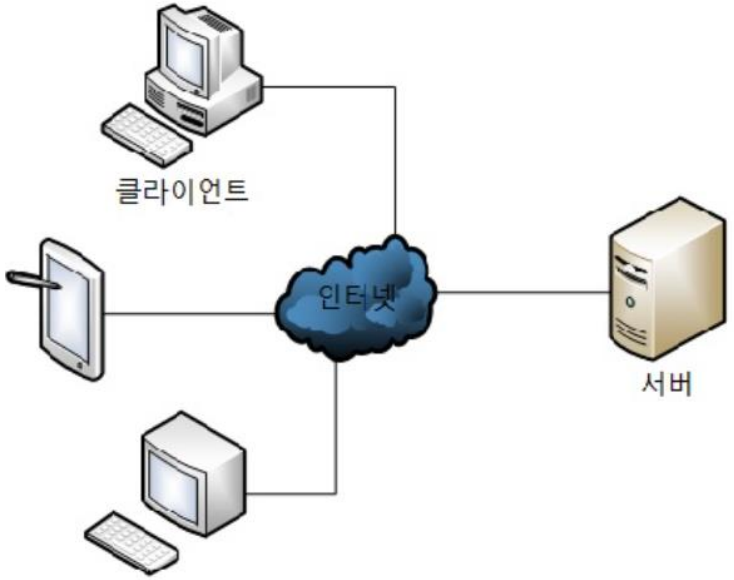
네트워크 모델

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

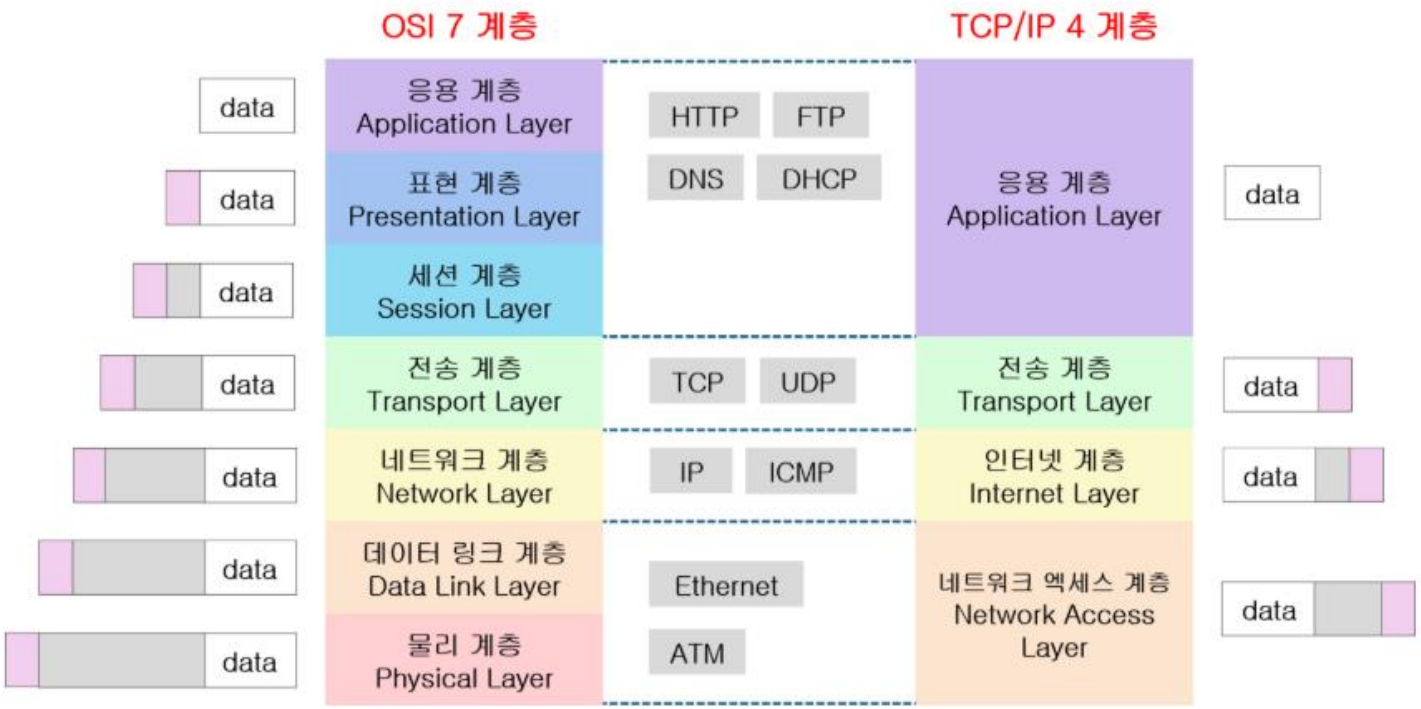
- 서비스 제공자를 제공하는 서버와 서비스를 요청하는 클라이언트로 구성
- 모든 자원이 서버에 집중(증앙 집중식 구조)
- 인터넷에서 사용되는 가장 일반적인 네트워크 구조

클라이언트_서버 모델

- 클라이언트는 서비스를 요청하고 서버는 클라이언트의 요청을 받아 처리하고 응답을 전송하는 방식
- 하나의 서버와 다수의 클라이언트 통신 가능
- 인터넷에서 사용되는 네트워크 모델
- 서버
  - 클라이언트의 요청이 접수되면 처리하여 응답을 전송하는 동작
  - 클라이언트와 통신을 위해 소켓(server socket)을 사용
  - 소켓은 데이터 통신을 하기 위한 일종의 인터페이스이다
  - 소켓의 식별: (IP주소 + 프로세스 식별 포트번호)
- 클라이언트
  - 클라이언트(client) 소켓을 사용하여 서버와 연결(서버의 주소를 알고 있어야 함)
  - 서버에게 데이터를 요청하고 서버로부터 응답을 받는다

클라이언트_서버 네트워크 프로그래밍

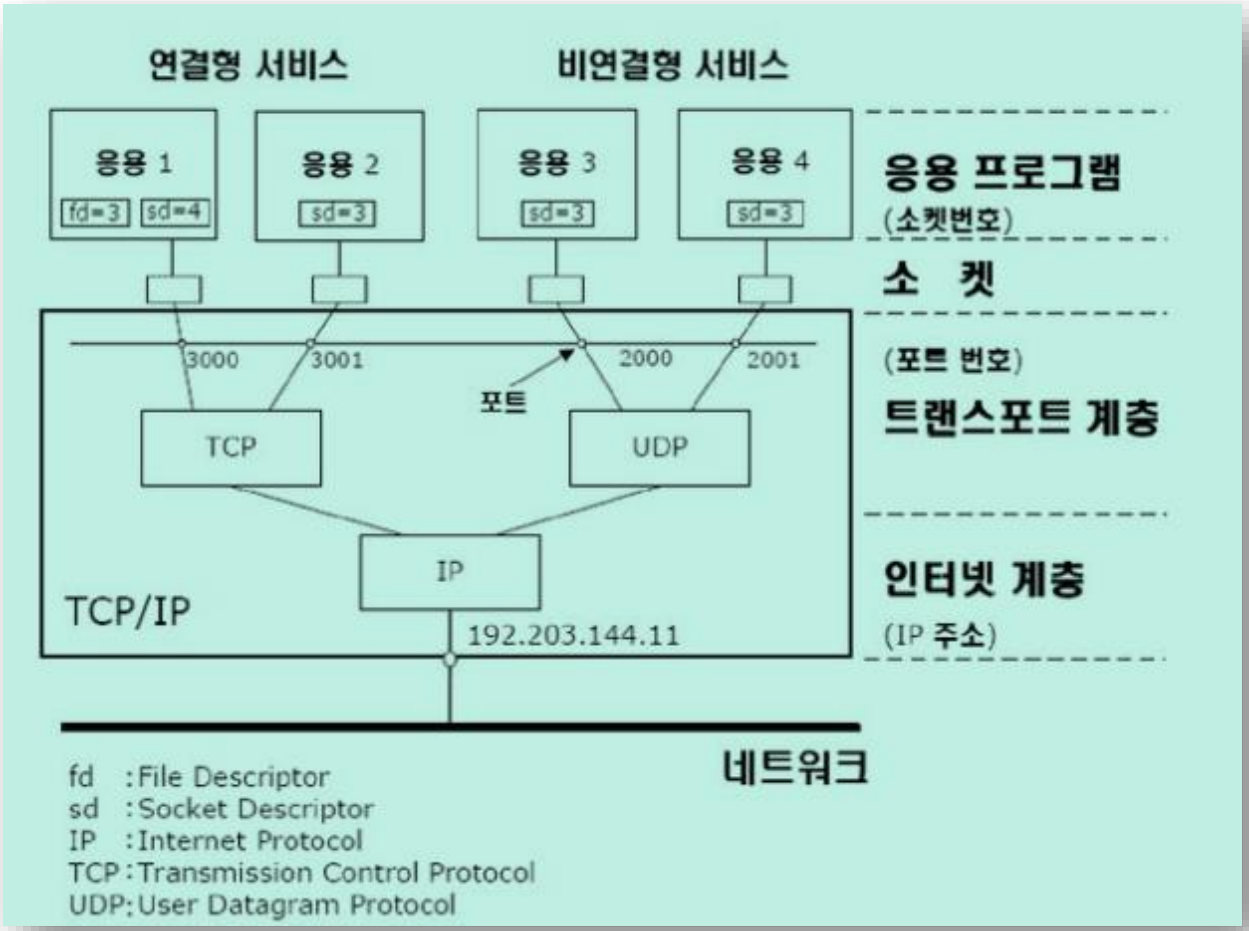
- TCP/IP 프로토콜의 네트워크 접근 계층(데이터링크 계층), 네트워크 계층, 전송 계층은 운영 체제 속에 포함되어 있음
- 사용자 프로그램은 전송 계층의 서비스를 받아 동작하는 응용 프로그램이다
- 전송 계층 서비스는 소켓을 통해 제공된다
  통신을 위한 종단점(endpoint)을 **소켓(socket)** 이라 한다
- 전송 계층의 서비스를 이용하려면 파이썬 모듈의 API 함수를 이용



참고: https://westahn.com/osi-7-계층이란/

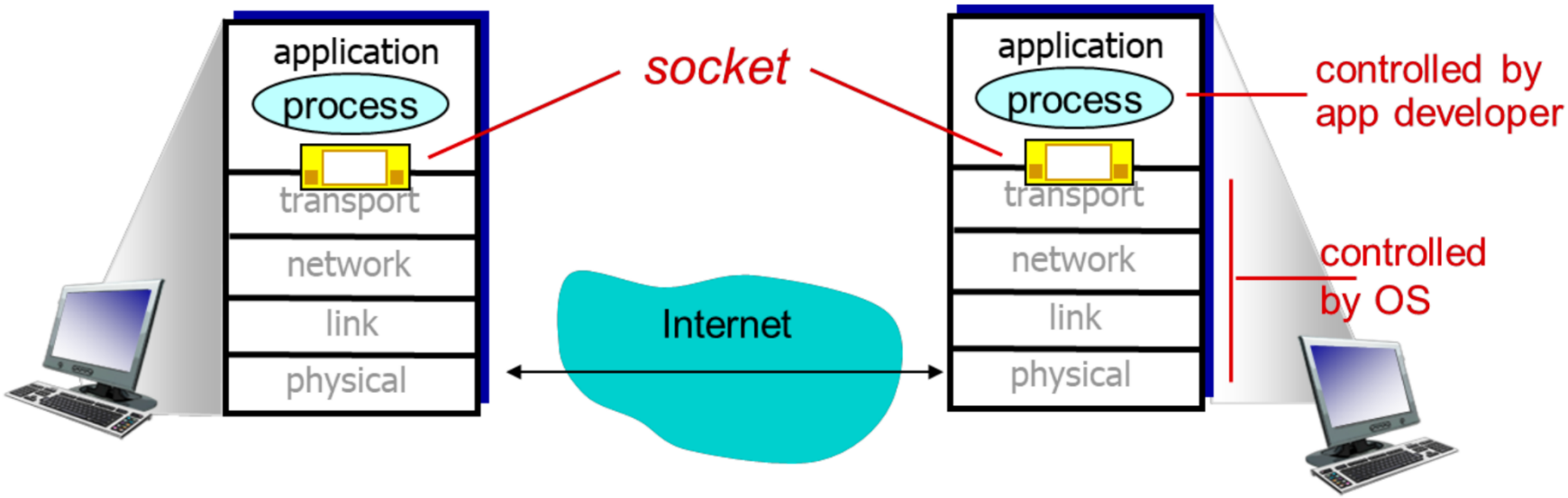
응용 프로그램의 식별

- 응용 프로그램이 연결된 소켓을 식별하기 위해서는 IP주소와 함께 포트 번호가 필요하다



소켓의 위치

- 소켓이란 응용 프로세스(앱)와 전송 계층 사이의 API를 지칭하는 것이다.
  - 웹 브라우저는 프로세스로 사용자계층(응용계층)에 실행되고 있고, 그 아래는 운영체제로 커널에서 제공하는 기능이다.
  - 커널(OS)에서 네트워크 사이에 서버-클라이언트 통신 기능을 구현하는데 그 응용계층과 커널의 연결점이 소켓이다. 소켓에 데이터를 읽고, 쓰고, 종료하는 것이다.



참고: https://dodenkr.tistory.com/83

소켓 관련 모듈

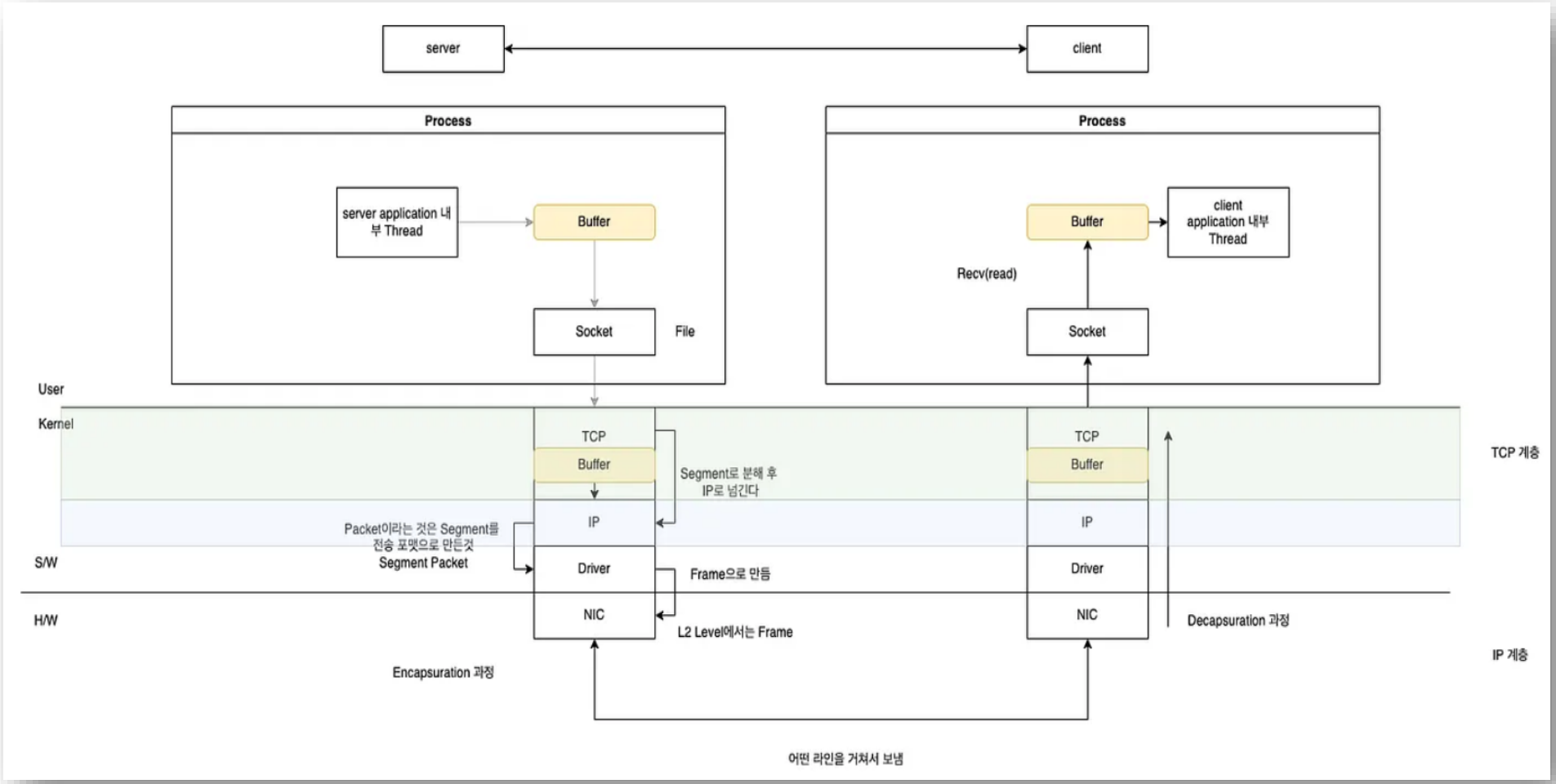
- socket 모듈
  - 소켓 프로그램을 작성하기 위한 기본 모듈
  - 동기식 I/O 수행(I/O 작업이 완료될 때까지 대기)
- asyncio 모듈
  - 코루틴(coroutine)과 이벤트 루프를 사용하여 비동기식 I/O 수행
  - 코루틴이란?
    - 실행 도중에 반환되며 다시 돌아오는 지점을 기억하고, 그 지점에서 실행을 재개할 수 있는 함수
    - 이벤트 루프와 함께 사용
    - 비동기 I/O 작업을 처리하는데 유용
- select 모듈
  - 소켓 중에서 읽기, 쓰기, 오류 이벤트가 발생한 소켓을 알려줌
  - 소켓 목록을 조사하여 이벤트를 나중에 비동기식으로 처리할 수 있다
  - socket 모듈과 함께 사용
- selectors 모듈
  - socket 모듈과 함께 비동기 프로그래밍을 위해 사용
  - 이벤트 구동 방식: 이벤트와 콜백을 등록하고 이벤트가 발생하면 콜백 함수 실행
- socketserver 모듈
  - 서버 프로그램을 쉽게 작성할 수 있는 모듈
  - 데이터가 수신되면 실행되는 handle() 함수를 정의하여 서버 구현
- twisted 모듈
  - 표준 모듈은 아니지만 이벤트 구동 방식의 3rd Party 모듈

동기식과 비동기식의 차이

- 동기식(Synchronous I/O)
  - 동기 입출력(Synchronous IO)은 호출된 함수가 입출력 작업이 완료될 때까지 블로킹(Blocking)된 상태로 대기하는 방식이다.
  - send 함수는 데이터를 보내고 난 후에 데이터가 완전히 전송될 때까지 대기하고, recv 함수는 데이터를 받을 때까지 대기한다.
  - 이러한 동기 입출력 방식은 호출한 함수가 입출력 작업을 처리하는 동안 다른 작업을 수행하지 못하고 대기해야 한다는 특징을 가지고 있다.

- 비동기식(Asynchronous I/O)
  - 클라이언트가 서버로 요청을 보낸 후, 서버의 응답을 기다리지 않고 바로 다른 작업을 수행한다. 후에 응답이 도착하면 그때 해당 응답을 처리하는 **로직(콜백 함수, 비동기 호출)**이 실행된다.
  - 입출력 작업을 호출한 후에 호출한 함수가 바로 반환되어 다른 작업을 수행할 수 있다. 입출력 작업의 완료는 이벤트 또는 콜백 함수를 통해 알려지며, 알림을 받은 후에 데이터를 확인하거나 추가 작업을 수행할 수 있다.
  - 호출한 함수가 입출력 작업을 대기하지 않고도 다른 작업을 수행할 수 있기 때문에 효율적인 자원 활용과 더 높은 처리량을 가질 수 있다.

TCP/IP와 소켓통신 send/recv 동작원리



참고: https://medium.com

TCP/IP와 소켓통신 send/recv 동작원리

① 서버 어플리케이션은 소켓을 오픈하고 applicatoin 레벨에서 보내고자 하는 데이터를 저장할 메모리를 할당한다.

② 할당한 메모리에 데이터를 담고 send를 한다. (send() 함수는 소켓이라는 file에 데이터를 write하는 행위이다).

③ Send() 함수에 대한 systemcall이 커널 레벨로 전달이 되면 TCP 계층에서 해당 데이터를 버퍼에 복사(copy)한다. (Buffer IO라고 한다.)

④ 버퍼에 쌓인 데이터를 세그먼트(Segment) 단위로 나눈다. 그리고 이 데이터를 IP 계층으로 전달한다. (TCP 헤더 + Segment)

⑤ IP 계층에서는 수신 받은 데이터에 송수신 IP 데이터를 헤더에 담아서 보낸다. 여기서 IP 헤더 + (TCP헤더 + Segment)를 패킷이라고 부른다.

⑥ 네트워크 인터페이스로 넘어가면서 패킷은 프레임이된다. 여기서도 계층단계별로 헤더가 추가된다. 추가 된 헤더는 최종적으로 1, 0으로 이루어진 전기신호로 전송된다.

⑦ 클라이언트에서 수신한다. 그리고 서버와는 반대로 IP계층, TCP 계층을 거쳐 헤더를 하나씩 벗겨내고, 헤더에 담긴 정보의 어플리케이션에서 할당한 소켓으로 데이터를 전달한다.

⑧ TCP 버퍼에서 segment를 받으면 서버로 잘 받았다는 Ack를 보내고, 받은 세그먼트 다음 번호를 붙여서 보낸다. 여기에 TCP 버퍼가 데이터를 더 받을 수 있는지를 나타내는 윈도우 크기(window size)도 함께 보낸다.

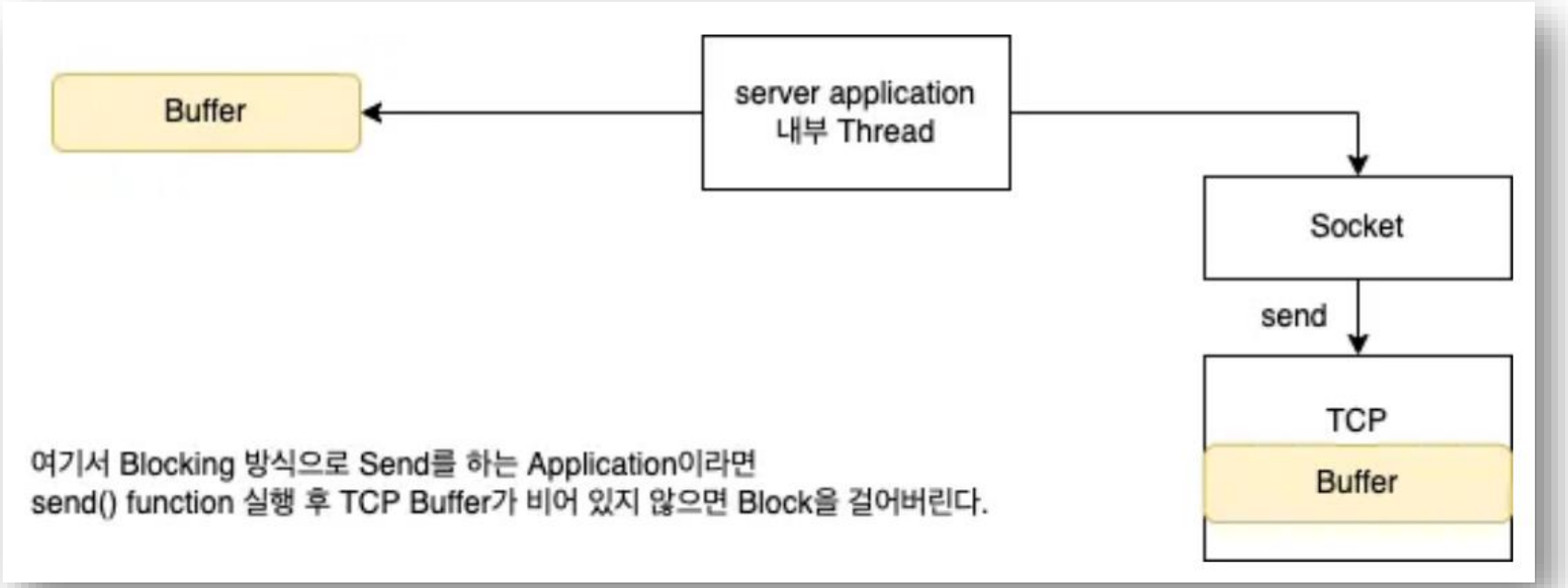
⑨ 서버는 ACK를 수신하고 window size를 확인한다. 데이터를 더 보낼 수 있으면 정상적으로 다음 segment를 전송한다. 반대로 보낼 수 없으면 보내지 않는다.

⑩ 그래서 클라이언트 application은 window size가 유지되도록 빠르게 데이터를 처리 해야 한다.

TCP/IP와 소켓통신 send/recv 동작원리

- 사용자 공간(user space) 내의 버퍼
  - 역할
    - 애플리케이션 단에서 데이터를 주고받을 때 임시로 데이터를 저장하는 영역
    - 애플리케이션이 소켓을 통해 데이터를 송수신할 때 임시 저장 공간 역할을 하며, 사용자 공간에서 소켓과 직접적으로 연관된 버퍼
  - 이름 예: 애플리케이션 버퍼, 사용자 버퍼, 소켓 버퍼
- 커널 공간(kernel space) 내의 TCP 버퍼
  - 역할
    - TCP 프로토콜 스택 안에서 전송할 세그먼트(segment)나 수신한 세그먼트 데이터를 임시 저장
    - 송신 시에는 TCP 세그먼트를 분해하고 IP 계층으로 넘기기 전에 버퍼에 저장하고, 수신 시에는 IP에서 전달된 데이터를 TCP 버퍼에 저장한 후 애플리케이션 쪽으로 전달
    - TCP 통신의 신뢰성과 흐름 제어를 담당
  - 이름 예: TCP 송신 또는 수신 버퍼

TCP/IP와 소켓통신 send/recv 동작원리



소켓 모듈

- 소켓을 생성하고 연결 설정, 데이터 송수신, 연결 해제 등의 기능을 수행하는 모듈
- 소켓 객체는 socket 모듈의 socket 클래스를 이용하여 생성

```
import socket
sock = socket.socket(family, type) #socket 객체
```

- family: 소켓의 주소 유형, 기본값: AF_INET → Address Family_Internet
- type: 소켓 유형,
  - SOCK_STREAM: TCP 프로토콜 사용(기본값)
  - SOCK_DGRAM: UDP 프로토콜 사용

```python
import socket

# TCP 프로토콜을 사용하는 STREAM 소켓 생성
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#sock = socket.socket()
```

```python
# UDP 프로토콜을 사용하는 DATAGRAM 소켓 생성
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```python
help(socket)
```

socket 클래스 메서드

socket 클래스는 연결, 데이터 송수신, 연결 해제, 소켓 통신 방식 지정 등을 위한 메서드를 제공한다

- 서버 소켓에서 사용하는 연결 메서드

메소드	기능
sock.bind()	종단점(address, port)를 소켓과 결합
sock.listen()	클라이언트 연결 대기
sock.accept()	클라이언트 연결 수용

- 클라이언트 연결 메서드

메소드	기능
sock.connect()	서버에 연결 요청. 문제 발생시 예외 반환
sock.connect_ex()	connect()와 기능은 비슷하나 문제 발생 시 오류 코드 반환

socket 클래스 메서드

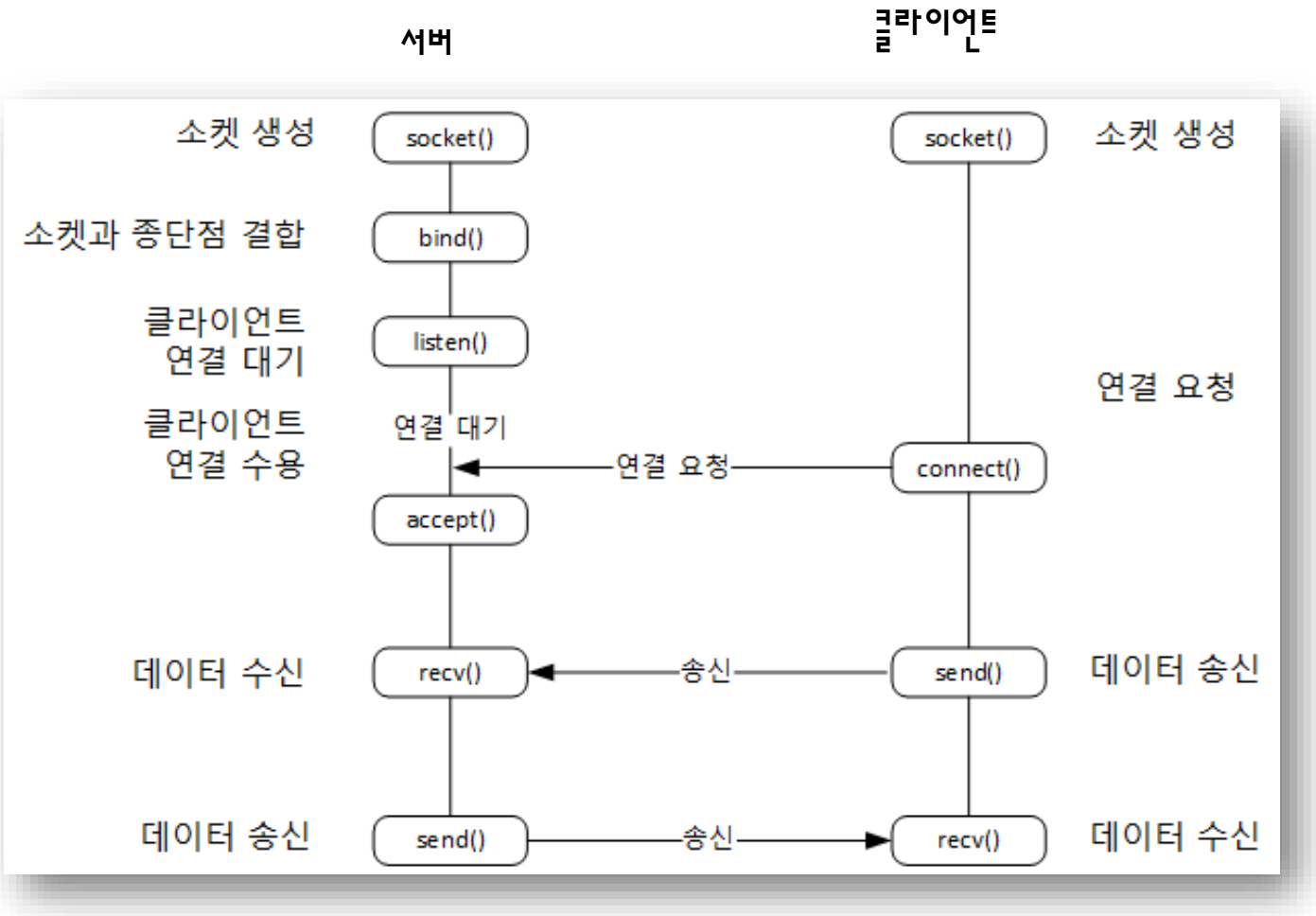
- 데이터 송수신 메서드

메소드	기 능
sock.recv()	TCP 소켓을 통해 메시지를 수신한다. 수신 데이터 반환
sock.recv_into()	TCP 소켓을 통해 메시지를 수신하여 버퍼에 저장한다
sock.send()	TCP 소켓으로 메시지를 전송한다. 송신 바이트 수 반환
sock.sendall()	TCP 소켓으로 메시지를 버퍼에 남기지 않고 모두 전송한다
sock.recvfrom()	UDP 소켓을 통해 메시지를 수신한다
sock.recvfrom_into()	UDP 소켓을 통해 메시지를 수신하여 버퍼에 저장한다
sock.sendto()	UDP 소켓으로 메시지를 송신한다

- 기타 연결 종료, 타임 아웃, 동작 모드 설정을 위한 메서드

메소드	기 능
sock.close()	소켓을 닫는다
sock.setblocking()	소켓을 차단 또는 비차단 모드로 설정한다
sock.settimeout()	차단 소켓의 타임아웃을 설정한다
sock.gettimeout()	차단 소켓의 타임아웃을 읽어온다
sock.detach()	소켓을 실제로는 닫지 않고 닫힌 상태로 설정한다
sock.setsockopt()	소켓 옵션을 설정한다

TCP 소켓 프로그래밍 순서



listen(): 서버 대기 큐(Waiting Queue) 지정

TCP 소켓 프로그래밍 순서

- 서버 소켓(Server socket)
  - socket(AF_INET, SOCK_STREAM) : 리스닝 소켓 객체 생성
  - Socket.bind(address) : 생성한 소켓을 서버 IP 및 포트를 튜플 형태로 맵핑
    - address = (IP, Port 번호)
  - socket.listen() : 소켓을 연결 요청에 대한 대기 상태로 전환
    - 소켓을 **수동 연결(Passive Open)** 상태로 전환하여, 클라이언트의 연결 요청을 수신 대기.
    - 이 단계에서 대기열(backlog) 크기를 지정하여 동시에 처리할 수 있는 보류 중인 연결 요청 수를 설정
  - socket.accept() : 연결 승낙 후 실제 통신 소켓 반환
    - 클라이언트의 연결 요청(Connection Request)을 수락하고, 해당 클라이언트와의 통신을 전담할 **새로운 소켓(Connected Socket)을 생성**하여 반환
    - 실제 통신에 사용하는 새로운 소켓 객체를 생성
    - 새로운 소켓은 리스닝 소켓과 동일한 주소 패밀리/타입/프로토콜을 가지며, 독립적인 소켓으로 send/recv, close 등을 별개로 수행
    - 원래의 리스닝 소켓은 계속해서 다른 클라이언트의 요청을 대기
  - socket.send(byte) or socket.sendall(byte) : 데이터 송신
  - socket.recv(bufsize) : 데이터 수신. bufsize는 한번에 수신되는 데이터 크기
  - socket.close() : 연결 종료

TCP 소켓 프로그래밍 순서

- 클라이언트 소켓(Client socket)
  - socket(AF_INET, SOCK_STREAM) : 소켓 객체 생성
  - socket.connect(address) : 서버 소켓에 연결 요청, 인자로 address를 튜플 형태로 구성
    - address = (서버 IP, 서버 지정 Port 번호)
    - 생성된 소켓을 이용하여 특정 서버의 **IP 주소와 포트 번호**로 연결을 요청(Active Open). 서버가 이 요청을 accept()하면 TCP 3-Way Handshake 과정을 거쳐 연결을 확립
  - Socket.send(byte) or socket.sendall(byte) : 데이터의 송신
  - Socket.recv(bufsize) : 데이터의 수신. bufsize는 한번에 수신되는 데이터의 크기

time 서버 프로그램

클라이언트가 접속하면 현재 시간을 전송하는 서버 프로그램(time_server.py)

(1) TCP 소켓 생성

```
import socket ❶ socket 모듈을 불러온다
import time
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)❷TCP 소켓 생성
```

(2) 소켓에 IP 주소와 포트 결합

- 소켓 주소는 반드시 (addr, port) 튜플로 표시

```
address = ('', 5000) #''=서버의 임의 주소, 포트 번호=5000
s.bind(address) ❸ 소켓과 주소 결합
```

(3) 클라이언트 연결 대기

```
s.listen(5) ❹ 연결 대기. 5개 연결까지 소켓에 대기 가능
```

time 서버 프로그램

(4) 연결 요청을 수락하고 데이터 송신

해당 클라이언트와의 통신을 전담할

- 클라이언트가 연결요청하면 accept() 함수로 요청을 수락하고 클라이언트 소켓과 클라이언트 주소를 반환
- 데이터 송수신은 반환받은 클라이언트 소켓을 통해 이루어진다
- 서버 소켓은 계속 연결 요청을 받는 용도로 사용됨
- send(message) 함수로 데이터를 전송한다
- 전송 후에는 소켓을 닫는다

```
while True:
    client, addr = s.accept() ❺ 연결 수락. (client socket, rem_addr) 반환
    print("Connection requested from ", addr)
    client.send(time.ctime(time.time()).encode()) ❻ 현재 시간을 전송
    client.close() ❼ 소켓 종료
```

(파이썬에서 문자열은 객체임)

- **송신 메시지는 바이트형**이어야 하므로 encode()함수를 사용하여 문자열을 바이트형으로 변환해야 한다

```
send(msg.encode()) #send a message to server
```

- 수신 메시지를 출력하려면 decode() 함수를 사용하여 바이트형을 문자열로 변환해야 한다

```
r_msg.decode()
```

time 서버 프로그램(time_server.py)

```python
# time_server.py
import socket
import time

# 컴퓨터 IP 주소 확인
print("서버 주소:",socket.gethostbyname(socket.gethostname()))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #소켓 생성
address = ("", 5000)#소켓 주소
s.bind(address)#소켓과 주소 결합
s.listen(5)#연결 대기
print("클라이언트 연결 대기...")

while True:
    client, addr = s.accept() #연결 요청 수용
    print("Connection requested from ", addr)
    client.send(time.ctime(time.time()).encode()) #현재 시각을 전송
    client.close() #연결 해제
```

- time_server.py를 실행하고 콘솔창에서 "C:>telnet localhost 5000" 실행
- telnet 명령이 실행되지 않으면 "제어판 > 프로그램 > Windows 기능 켜기/끄기 > 텔넷 클라이언트" check 후 명령을 다시 실행

time 클라이언트 프로그램 (time_client.py)

타임 서버에 접속하여 시간을 읽어 오는 클라이언트 프로그램

- 서버에 연결하기

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
address = (“localhost”, 5000)
sock.connect(address) ❸ 서버에 연결
```

- 메시지 수신하기

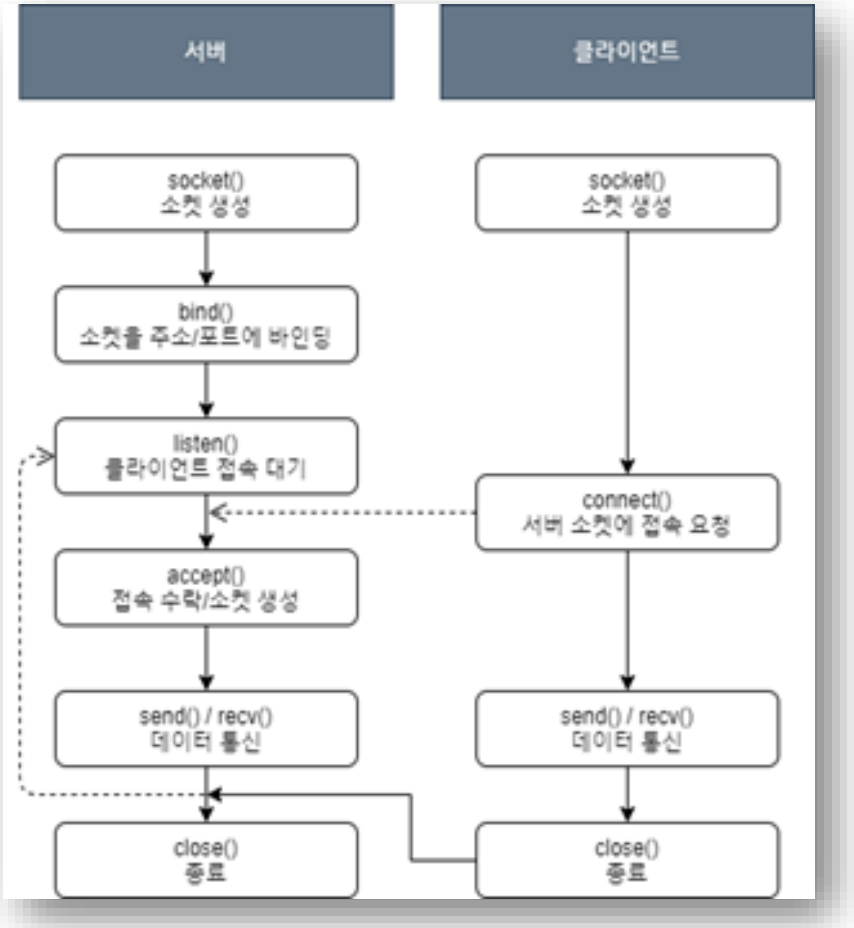
```
sock.recv(1024) ❹ 수신 바이트를 문자열로 변환하여 출력
```

```python
# time_client.py
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 5000))
print("현재 시각: ", s.recv(1024).decode())
s.close()
```

TCP 에코 프로그램

- TCP 서버 프로그램: 수신 데이터를 출력하고 다시 상대방에게 그대로 전송하는 서버 프로그램
- TCP 클라이언트 프로그램: 에코 서버에 연결하여 메시지를 전송하고, 수신 메시지를 출력하는 클라이언트 프로그램



- recv() 함수로 데이터를 수신했을 때 함수가 반환은 되지만 수신 데이터가 없으면 연결이 끊어진 것.

```
data = client.recv(1024)  #최대 1024 바이트 수신
if not data:   #수신 데이터 없음
    break      # 연결 종료
```

recv() 함수의 반환값

- recv() 함수는 기본적으로 데이터를 받을 때까지 **기다리는(blocking)** 역할
- recv() 함수가 0바이트를 반환하는 것은 TCP 소켓 통신에서 연결의 정상적인 종료를 의미.
  1. 연결 종료 신호: TCP 연결에서 한쪽 피어(peer)가 통신을 완료하고 close() 시스템 콜을 호출하면, 해당 피어는 더 이상 전송할 데이터가 없음을 나타내는 FIN(Finish) 세그먼트를 상대방 피어에게 전송. 이는 연결 종료 프로세스(TCP 4-way handshake)의 시작
  2. recv()의 해석: recv() 함수는 수신 버퍼에 데이터가 도착할 때까지 블로킹 상태로 대기. 그러나 상대방 피어로부터 FIN 세그먼트가 수신되면, 이는 해당 피어가 더 이상 데이터를 전송하지 않을 것임을 알리는 명확한 시그널. 이러한 상황에서 recv() 함수가 호출되면, 읽어올 실제 데이터가 없으므로 0을 반환
  3. 신뢰성 있는 종료: recv()의 0 반환은 상대방 피어에 의한 정상적 연결 종료를 나타냄. 이는 네트워크 오류나 비정상적인 연결 단절(예: RST 패킷 수신)로 인한 음수 값 반환과는 명확히 구분
  4. 자원 관리: 애플리케이션은 recv()의 0 반환을 통해 상대 피어가 연결을 종료했음을 인지하고, 본인 측의 소켓 리소스를 정리하기 위해 close()를 호출하여 연결 종료 프로세스를 완료 함.

TCP 에코 서버 프로그램(TCP_echoserver_demo.py)

TCP 에코 클라이언트 프로그램(TCP_client_demo.py)

실행 방법 1

-아나콘다 명령창을 두 개 열어서 각각 실행한다. 그러면 각 창에 실행 결과가 출력된다.
  - python TCP_echoserver_demo.py
  - python TCP_client_demo.py

실행 방법 2

-Spyder 콘솔 창에서 서버 파일 용 콘솔과 클라이언트 파일 용 콘솔을 연다.
-각 파일을 해당 콘솔과 연결 후 실행한다. 그러면 각 콘솔에 실행 결과가 출력된다.

서버와 클라이언트의 예외처리

- 상대방과 연결이 끊어졌을 때 send()/recv()를 호출하면 예외가 발생하여 프로그램이 중단된다
- 정상적인 프로그램 실행을 위해 예외 처리 필요

```
while True:
    #수신할 때 예외 발생 처리
    try:
        data = conn.recv(BUFSIZE)
    # 예외가 발생하면 소켓을 닫고 프로그램을 종료한다
    except:
        conn.close()
        break

    #정상 처리
    else:
        print(data.decode())

    # 송신할 때 예외 발생 처리
    try:
        conn.send(data)
    #예외가 발생하면 종료한다
    except:
        conn.close()
        break
```

예외처리 _ 비정상종료

- c_sock.recv() 중에 except 블록이 실행되는 이유
  - 대부분 네트워크 연결이 FIN을 통한 정상 종료가 아니라, RST 패킷 수신 등 비정상적인 방식으로 중단되었거나, 또는 지정된 시간 내에 데이터가 도착하지 않았을 때(타임아웃) 발생한다.
- sock.send() 중에 except 블록이 실행되는 이유
  - 대부분 송신 시점에 이미 상대방이 연결을 끊었거나 (특히 RST를 보내며 강제적으로), 네트워크 상에 문제가 발생하여 연결이 더 이상 유효하지 않을 때이다.
- 두 경우, recv() 함수가 0을 반환하여 정상 종료를 알리는 것과는 달리, 예기치 않은 비정상 종료 상황에 해당한다.

예외 처리 에코 서버(TCP_echoserver.py)

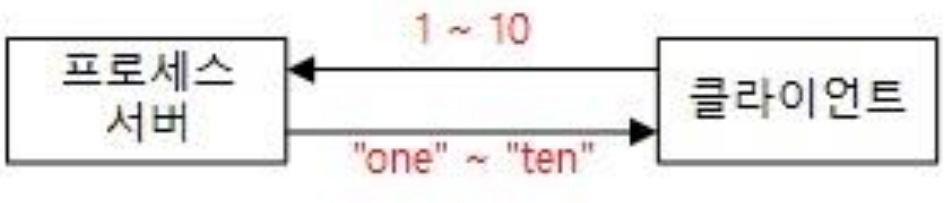
예외 처리 클라이언트(TCP_client.py)

1. 요점 사항

① 서버를 실행하고 클라이언트를 실행해 보자. 클라이언트에서 몇 개의 메시지를 전송한 후, 'q'를 입력하여 정상 종료했을 때 서버의 반응을 확인한다.
② 다시 클라이언트를 실행한 후 터미널을 종료했을 때 서버의 반응을 확인한다. 어떤 차이가 있는가?

요청 처리 TCP 서버 프로그램

- 클라이언트로부터 1~10까지의 숫자를 받으면 영어를 전송하는 서버 프로그램을 작성한다. 예를 들어 1을 받으면 one을 전송한다.



TCP 프로세싱 서버 프로그램(TCP_processs_server.py)

TCP 프로세싱 클라이언트 프로그램(TCP_client.py)

사용자 모듈을 이용한 TCP 서버 프로그램

TCP 서버 프로그램을 작성하기 위한 클래스로 구성된 모듈

메서드:

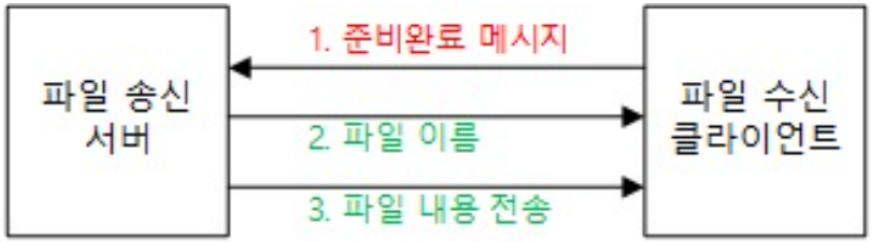
- **init**(self, port)
  소켓을 생성하고 클라이언트 연결 대기
- accept(self)
  연결을 수락하고 클라이언트 소켓과 주소 반환
- send(self, msg) 메시지를 전송하고 성공(True)/실패(False) 반환
- receive(self, r_sock) r_sock에서 메시지를 수신하여 반환
- disconnect(self) 연결 해제

TCP 서버 프로그램(MyTCPServer.py)

TCP 사용자정의 프로그램(MyTCPServer_ex01.py)

TCP 프로세싱 클라이언트 프로그램(TCP_client.py)

TCP 소켓을 이용한 파일 송수신



서버

준비완료메시지를 수신한 후, 파일 이름을 전송하고 파일을 열어 내용을 전송함

클라이언트

서버에 연결 후, 준비완료메시지를 전송하고 수신된 파일 이름으로 파일을 열고 수신 파일 내용을 저장함

파일 송신 프로그램

(1) TCP 소켓을 생성하고 클라이언트로부터 준비완료 메시지를 받는다

```
msg = c_sock.recv(1024) #클라이언트로부터 준비 완료 수신
print(msg.decode())
```

(2) 전송파일 이름 전송하기

```
c_sock.sendall(filename.encode()) #파일 이름 전송
```

(3) 파일 열고 내용 송신하기

- with open() as 문으로 파일을 오픈한다.
- sendfile() 메소드를 사용하면 한 번에 전송 가능

```
with open(filename, 'rb') as f: #바이너리 읽기 모드로 열기
   c_sock.sendfile(f,0) #파일 전송, 0=offset
```

또는 데이터를 한 줄씩 읽어 전송

```
data = f.read()
while data:
    c_sock.send(data)
    data = f.read()
```

파일 수신 프로그램

- 소켓을 생성하여 서버로 접속하고 준비완료 메시지를 전송한다
- 파일 이름을 받아 바이너리 쓰기 모드로 파일 오픈

```
fn = s_sock.recv(1024).decode() #파일 이름 수신
with open('new_'+fn, 'wb') as f: #쓰기 모드로 수신 파일 열기
```

- 일정한 크기로 파일 내용을 받아 파일에 저장

```
print('receiving file...')
while True:
    data = s_sock.recv(8192)
    if not data: #더 이상 내용이 없으면 루프를 빠져나간다
        break
    f.write(data)
```

파일 전송 프로그램(TCP_sendfile.py)

파일 수신 프로그램(TCP_receivefile.py)

TCP 소켓을 이용한 동영상 송수신 프로그램

- 웹카메라의 동영상을 TCP 소켓을 이용하여 실시간으로 송신하는 프로그램
- 프로그램을 위해 OpenCV와 imutils 모듈을 이용한다.

  C:>pip install –U —user opencv-python imutils
- 동영상은 프레임 단위로 전송되고 프레임 구성은 다음과 같다

- 웹카메라 동영상 획득

  cv2.VideoCapture(0)
- 프레임의 해상도 수정

  imutils.resize(frame, width)
- 프레임 길이는 비번호 8바이트로 변환

  struct.pack('Q', len(frame_bytes))

  'Q'는 객체를 비번호(unsigned) 8바이트로 변환하는 변환 지정 문자
- 프레임을 바이트 스트림으로 변환하여 전송

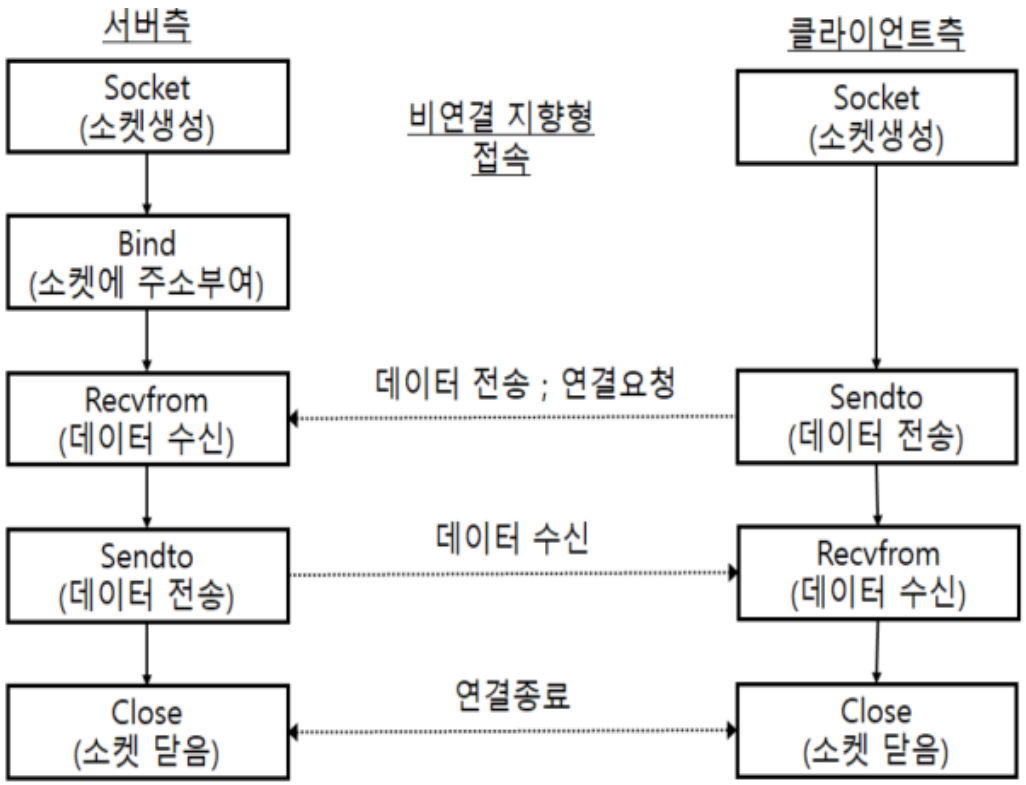
  pickle.dumps()
- **수신 프로그램은 메시지(프레임)를 수신하여 프레임 길이 정보(8바이트)를 알아내고 길이 만큼의 데이터를 잘라 화면에 표시하는 동작을 반복한다.**

동영상 송신 프로그램(video_server.py)

동영상 수신 프로그램(video_client.py)

UDP 소켓 프로그래밍

- UDP 프로토콜을 이용한 데이터 전송 프로그래밍
  - 전송 전 연결을 하지 않고 즉시 송수신함
  - 전송 신뢰성을 보장하지 않음
- UDP 소켓 유형
  - SOCK_DGRAM
- 송신 메서드: sendto()
- 수신 메서드: recvfrom()



UDP 에코 서버 및 클라이언트 프로그램

• UDP 에코 서버 프로그램

- UDP 프로토콜을 사용한 에코 서버 프로그램
  - 클라이언트의 메시지를 받아 다시 그대로 전송한다.
  - 연결이 없이 송수신되므로 다수의 사용자와 송수신 가능
- 데이터 수신
  - recvfrom() 함수를 사용
  - 이 함수는 수신 데이터(data)와 상대방 주소(addr)을 반환한다
  - addr에는 ip주소와 port가 포함되어 있다(addr = (ip_addr, port))

```
data, addr = sock.recvfrom(maxsize)
```

- 데이터 송신
  - sendto(data,addr) 함수를 사용

```
sock.sendto(data,addr)
```

• UDP 클라이언트 프로그램

서버로 메시지를 송신하고 응답을 수신하는 UDP 클라이언트

- UDP 프로토콜을 사용하므로 connect()와 같은 연결이 없다

UDP 소켓을 이용한 파일 송수신

- UDP 소켓을 이용하여 파일을 송수신하는 프로그램
  - 프로그램을 실행할 때 인자를 이용하여 기능을 선택한다("py program s/r")
  - 's' 인자와 함께 실행하면 송신 동작을 하고, 'r' 인자와 함께 실행하면 수신 동작
  - 'r'인자 --> Receiver() 함수 호출하여 파일 수신
    - 파일 이름을 수신하고 'OK"를 송신한다
    - 파일 내용을 받아 저장한다
  - 's'인자 --> Sender() 함수를 호출하여 파일 송신
    - 파일 이름을 전송하고 'OK'를 수신한 후 파일 내용 전송

UDP 소켓을 이용한 동영상 송신 프로그램

- UDP 소켓을 이용하여 클라이언트에서 웹 카메라 영상을 서버로 전송하고, 서버에서 수신 동영상을 표시하고 MP4 형식으로 저장
- 컬러 동영상(640 x 480)은 고정 길이 프레임 단위로 전송
- 프레임 픽셀 수 = 640 x 480 x 3
  - 640 x 480 x 3 = 46080 x 20이므로 프레임 픽셀을 46080 바이트씩 20개로 나누어 전송
- 프레임을 아래와 같이 구성하여 전송

- 웹카메라에서 읽은 3차원 데이터를 1차원으로 평탄화하고 바이트형으로 변환 전송

```
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
one_d_frame = frame.flatten() #평탄화
send_frame = one_d_frame.tobytes() #바이트형으로 변환
#i=프레임 번호, fSize=46080(프레임 크기)
sock.sendto(bytes([i])+send_frame[i*fSize:(i+1)*fSize],(UD
P_IP, UDP_PORT))
```

UDP 소켓을 이용한 동영상 수신 프로그램

- 동영상 프레임을 저장할 46080 x 20 버퍼 생성

```
buffer = [b'Wxff' * 46080 for x in range(20)]
```

- MPEG4 동영상 파일을 위한 코덱과 압축 방식을 가져와서 동영상을 파일로 저장할 VideoWriter 객체를 생성

```
fourcc = cv2.VideoWriter_fourcc(*'XVID') # XVID MPEG-4 코덱
out = cv2.VideoWriter('output.mp4', fourcc, 25.0, (640,480)) # (파일이름, 코덱, fps, 해상도)
```

- 20개의 프레임이 모두 수신되면 프레임을 모두 합친 후,
  numpy 모듈의 frombuffer()를 이용하여 바이트형 데이터를 1차원 정수 배열로 변환하고
  평탄화된 데이터를 다시 원래대로 640 x 480 x 3 포맷으로 복원한다.

```
frame = numpy.frombuffer(picture, dtype=numpy.uint8) # picture를 1차원 배열로 변환
# 640 x 480 x 3으로 변환
frame = frame.reshape(480, 640, 3)
```

- 변환된 데이터는 화면에 표시하고 MP4 파일로 저장된다.

```
out.write(frame)
```

브로드캐스팅 프로그램

- 브로드캐스팅(Broadcasting)은 로컬 네트워크에 연결된 모든 호스트에게 메시지를 보내는 통신방식
- 네트워크 대역폭을 적게 사용
- 브로드캐스트 주소: "xxx.xxx.xxx.255" 또는 <broadcast>
- UDP 소켓만 사용 가능(SOCK_DGRAM)

```
C:\Users\leeJo>ipconfig /all

Windows IP 구성

무선 LAN 어댑터 Wi-Fi:

   연결별 DNS 접미사. . . . . :
   설명. . . . . . . . . . . . . : Intel(R) Wi-Fi 6 AX200 160MHz
   물리적 주소 . . . . . . . . . : 70-D8-C2-10-1D-6E
   DHCP 사용 . . . . . . . . . . : 예
   자동 구성 사용. . . . . . . . : 예
   링크-로컬 IPv6 주소 . . . . . : fe80::be7e:924:808e:e19e%11(기본 설정)
   IPv4 주소 . . . . . . . . . . : 172.30.1.31(기본 설정)
   서브넷 마스크 . . . . . . . . : 255.255.255.0
   임대 시작 날짜. . . . . . . . : 2025년 8월 11일 월요일 오후 1:43:57
   임대 만료 날짜. . . . . . . . : 2025년 8월 11일 월요일 오후 2:53:27
   기본 게이트웨이 . . . . . . . : 172.30.1.254
   DHCP 서버 . . . . . . . . . . : 172.30.1.254
   DHCPv6 IAID . . . . . . . . . : 175167682
   DHCPv6 클라이언트 DUID. . . . : 00-01-00-01-2F-66-9E-4D-70-D8-C2-10-1D-6E
   DNS 서버. . . . . . . . . . . : 202.30.55.11
                                   202.30.55.55
                                   168.126.63.1
                                   168.126.63.2
   Tcpip를 통한 NetBIOS. . . . . : 사용
```

setsockopt

- 소켓의 송수신 동작을 setsockopt() 함수를 이용해서 다양한 옵션으로 제어
- setsockopt(level, optname, value)
  - level : 옵션의 종류. 보통 SOL_SOCKET와 IPPROTO_IP 중 하나를 사용
    - SOL_SOCKET: 소켓관련 옵션의 설정 시에 사용
    - IPPROTO_IP: 멀티캐스트와 같은 프로토콜 관련 옵션을 설정 시에 사용
  - optname : 설정을 위한 소켓 옵션의 번호
  - value : 설정 값

setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)

1. SOL_SOCKET : 일반적인 소켓 수준(socket-level)**에서 적용되는 옵션
2. SO_REUSEADDR: 특정 주소(IP 주소와 포트 번호)가 TIME_WAIT 상태에 있더라도, 다른 소켓이 이 주소를 즉시 다시 사용할 수 있도록 허용
3. 1 (또는 TRUE): SO_REUSEADDR 옵션을 활성화하겠다는 의미입니다. 즉, "해당 주소를 재사용하도록 허용하라"는 지시

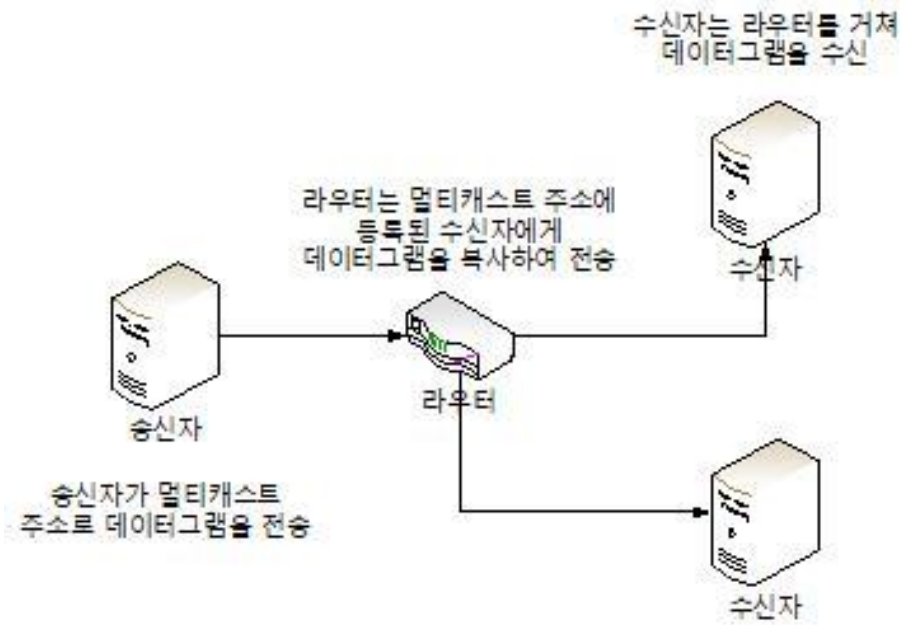
SOL_SOCKET level

값	자료형	설명
SO_REUSEADDR	BOOL	이미 사용된 주소를 재사용하도록 함
SO_RCV_BUF	int	수신용 버퍼의 크기 지정
SO_SND_BUF	int	송신용 버퍼의 크기 지정
SO_RECVTIMEO	DWORD(timeval)	수신시 Blocking 제한 시간을 설정
SO_SNDTIMEO	DWORD(timeval)	송신시 Blocking 제한 시간을 설정
SO_KEEPALIVE	BOOL	TCP 통신에서만 유효. 일정 시간마다 연결 유지 상태를 체크.
SO_LINGER	struct LINGER	소켓을 닫을 때 남은 데이터의 처리 규칙 지정
SO_DONTLINGER	BOOL	소켓을 닫을때 남은 데이터를 보내기 위해서 블럭되지 않도록 함
SO_DONTROUTE	BOOL	라우팅(Routing)하지 않고 직접 인터페이스로 전송
SO_BROADCAST	BOOL	브로드캐스트 사용 가능 여부

멀티캐스팅 프로그램

- 멀티캐스팅이란 한 번의 송신으로 메시지나 정보를 특정 그룹(멀티캐스트 그룹)에 속하는 여러 컴퓨터에 동시에 전송하는 방식
  - 특정 IP 주소를 통하여 멀티캐스트 그룹에게 메시지 전송
  - 멀티캐스팅을 통해 데이터를 수신하려면 멀티캐스트 그룹에 가입 필요
    - 그룹에 참여한 후 사용할 포트와 해당 주소를 묶는 bind 작업 수행
  - 멀티캐스트 주소 범위(1110 0000 ~ 1110 1111): 224.0.0.0~239.255.255.255
  - 네트워크 장비 (특히 라우터)가 멀티캐스트를 지원해야 함
  - UDP 프로토콜 사용

IP TV, 화상회의, 온라인게임 등에 사용



멀티캐스팅 – 동작원리

① **그룹 생성 및 가입:**

- 멀티캐스트 통신을 위해서는 먼저 수신자들이 특정 그룹에 가입해야 한다. 이 과정에서 IGMP (Internet Group Management Protocol)와 같은 프로토콜이 사용된다.

② **데이터 전송:**

- 송신자는 그룹 주소(멀티캐스트 IP 주소)로 데이터를 전송한다. 이 주소는 224.0.0.0 ~ 239.255.255.255 범위의 클래스 D IP 주소를 사용한다.

③ **데이터 복제 및 전달:**

- 라우터는 멀티캐스트 그룹 정보를 기반으로 데이터를 복제하여 해당 그룹에 속한 모든 수신자에게 전달한다.
- 스위치는 같은 네트워크 세그먼트 내에서 브로드캐스트와 유사하게 동작하며, 멀티캐스트를 지원하는 스위치는 그룹 정보를 기반으로 데이터를 전달한다.

④ **수신 확인:**

- 수신자는 데이터를 수신하고, 필요한 경우 IGMP를 통해 그룹 가입 사실을 알린다.

멀티캐스팅 송신 프로그램

- setsockopt(level, optname, value) 함수로 TTL(Time To Live) 옵션 설정
  - level = IPPROTO_IP, optname = IP_MULTICAST_TTL, value=TTL
  - TTL은 바이트형으로 표현

```
from socket import *
s_sock = socket(AF_INET, SOCK_DGRAM)
TTL = struct.pack('B', 2)
s_sock.setsockopt(IPPROTO_IP, IP_MULTICAST_TTL, TTL)
```

- 멀티캐스트 그룹 주소: 224.0.0.255
  - 224.0.0.0/24 범위의 주소들은 LAN 내에서만 유효한 메시지의 전달에 사용
  - 224.0.0.255는 모든 로컬 시스템 (All Systems on This Subnet)을 의미하는 예약된 멀티캐스트 주소
    - 이 주소로 패킷을 보내면 해당 서브넷에 연결된 모든 장치들이 이 패킷을 수신

멀티캐스팅 수신 프로그램

- 멀티캐스트 송신자가 보내는 메시지를 수신한다
- 소켓을 생성하고 멀티캐스트 그룹에 가입
- 멀티캐스트 그룹 가입정보는 (그룹주소, INADDR_ANY)로 구성
  - 그룹 주소는 inet_aton() 함수를 이용하여 문자열 주소를 32비트로 변환
  - INADDR_ANY는 빈 문자열이며 4 바이트로 표시

struct.pack() 함수를 이용하여 이를 다시 4 바이트로 표시

```
group_addr = '224.0.0.255'
mreq = struct.pack("4sl", inet_aton(group_addr), INADDR_ANY)
```

- 멀티캐스트 그룹 가입

```
r_sock.setsockopt(IPPROTO_IP, IP_ADD_MEMBERSHIP, mreq)
```

- 멀티캐스트 그룹 탈퇴

```
r_sock.setsockopt(IPPROTO_IP, IP_DROP_MEMBERSHIP, mreq)
```

struct 모듈

- struct 모듈은 파이썬의 값(예: 정수, 문자열)을 바이트(bytes) 데이터로 변환(packing)하거나, 바이트 데이터를 파이썬 값으로 다시 변환(unpacking)하는 기능을 제공
  - 패킹(Packing)은 특정 형식의 데이터를 다른 시스템이나 저장 매체가 이해하고 처리할 수 있는 이진(binary) 바이트(bytes) 스트림 형태로 변환하는 과정
    - **구조화된 데이터 → 바이트 스트림:** 파이썬의 정수, 실수, 문자열과 같이 인간이 읽고 이해하기 쉬운 형태의 데이터를, 컴퓨터가 직접 처리하는 0과 1로 구성된 바이트의 연속으로 변환
- TTL = struct.pack('@B', 2)
  - 파이썬 객체(여기서는 정수 2)를 이진 데이터(바이트)로 변환 (패킹)
  - 2라는 파이썬의 정수(integer) 값(객체)을
  - ' @B'라는 형식(format)에 따라
    - 현재 시스템의 기본 바이트 순서(엔디안)와 정렬 방식에 따라 1바이트 크기의 부호 없는 정수(unsigned char)로 처리
  - 1바이트 크기의 이진 데이터(b'\x02')로 패킹(packing)하여 TTL 변수에 할당