네트워크 프로그래밍

07. 소켓의 내부 동작

TCP 소켓에 존재하는 입출력 버퍼

입출력 버퍼의 특성

- 입출력 버퍼는 TCP 소켓 각각에 대해 별도로 존재한다.
- 입출력 버퍼는 소켓생성시 자동으로 생성된다.
- 소켓을 닫아도 출력버퍼에 남아있는 데이터는 계속해서 전송이 이뤄진다.
- 소켓을 닫으면 입력버퍼에 남아있는 데이터는 소멸되어버린다.

close 호출

쓰기

- write 함수가 호출되는 순간 데이터는 출력버퍼로 이동
- write 함수가 반환되는 시점: 전송할 데이터가 출력버퍼로 이동이 완료되는 시점

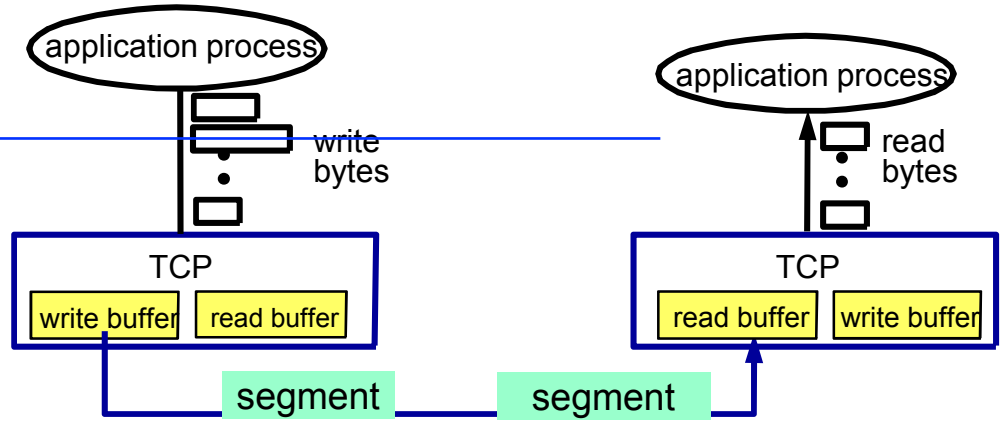
send que

읽기

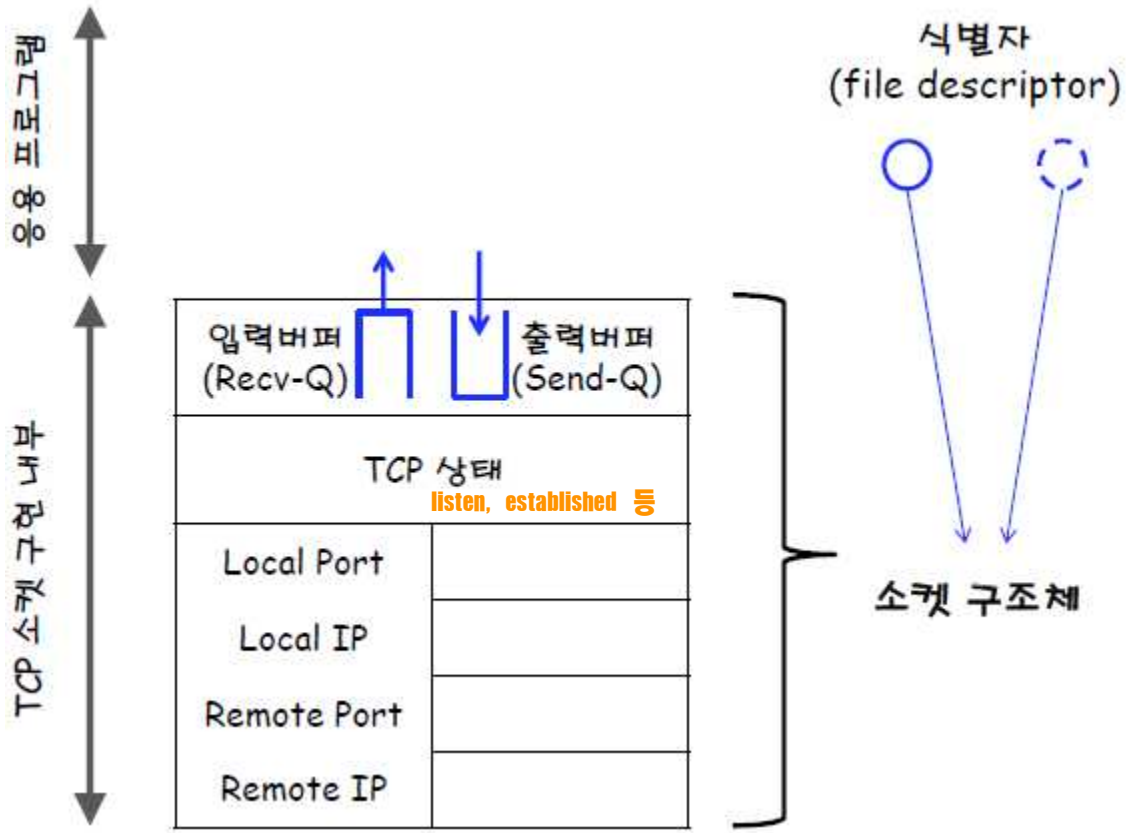
- read 함수가 호출되는 순간 입력버퍼에 저장된 데이터를 읽어 들임

이부분을 자른다고 이해

앞으로 이 버퍼들은 모두
리시브 큐, 샌드 큐로 정의



소켓과 관련된 데이터 구조



netstat -an


```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address  Foreign Address    State
tcp     0      0    0.0.0.0:2049     0.0.0.0:*        LISTEN
tcp     0      0    0.0.0.0:40453    0.0.0.0:*        LISTEN
tcp     0      0    0.0.0.0:9190     0.0.0.0:*        LISTEN
tcp     0      0    192.168.0.26:53  0.0.0.0:*        LISTEN
tcp     0      0    127.0.0.1:53     0.0.0.0:*        LISTEN
tcp     0      0    127.0.0.1:631    0.0.0.0:*        LISTEN
tcp     0      0    127.0.0.1:953    0.0.0.0:*        LISTEN
tcp     0      0    127.0.0.1:9190   127.0.0.1:35437  ESTABLISHED
tcp     0      0    127.0.0.1:35437  127.0.0.1:9190   ESTABLISHED
```

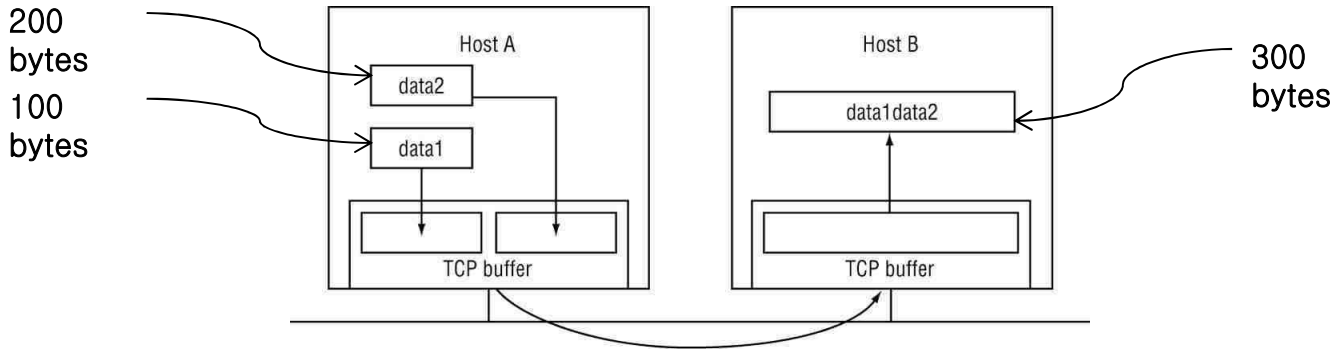
버퍼링과 TCP

경계가 없는 TCP 기반 데이터 전송

- TCP 프로토콜
  - "신뢰성 있는(reliable) 바이트 스트림" 프로토콜
    - TCP 소켓을 통해 전송된 어떤 데이터의 복사본이라도 상대방이 성공적으로 받을 때까지 TCP 내부 구현에 의해 유지 및 관리
    - 전송되는 데이터의 경계(boundary)를 구분할 수 없음

- 송신 측에서 한번의 write()로 전송한 데이터도 수신 측에서는 여러 번의 read() 호출로 나누어져 수신이 될 수 있으며, 여러 번의 write()를 호출하여 전송된 데이터도 단 한번의 read() 호출로 모두 받을 수 있다. read 함수의 매개변수로 지정한 만큼의 데이터를 읽을 뿐이다.

하나의 연결에 있어서 한쪽에서 쓰기를 한 크기와 다른 쪽에서 읽은 크기가 서로 연관이 있다고 가정해서는 안된다.



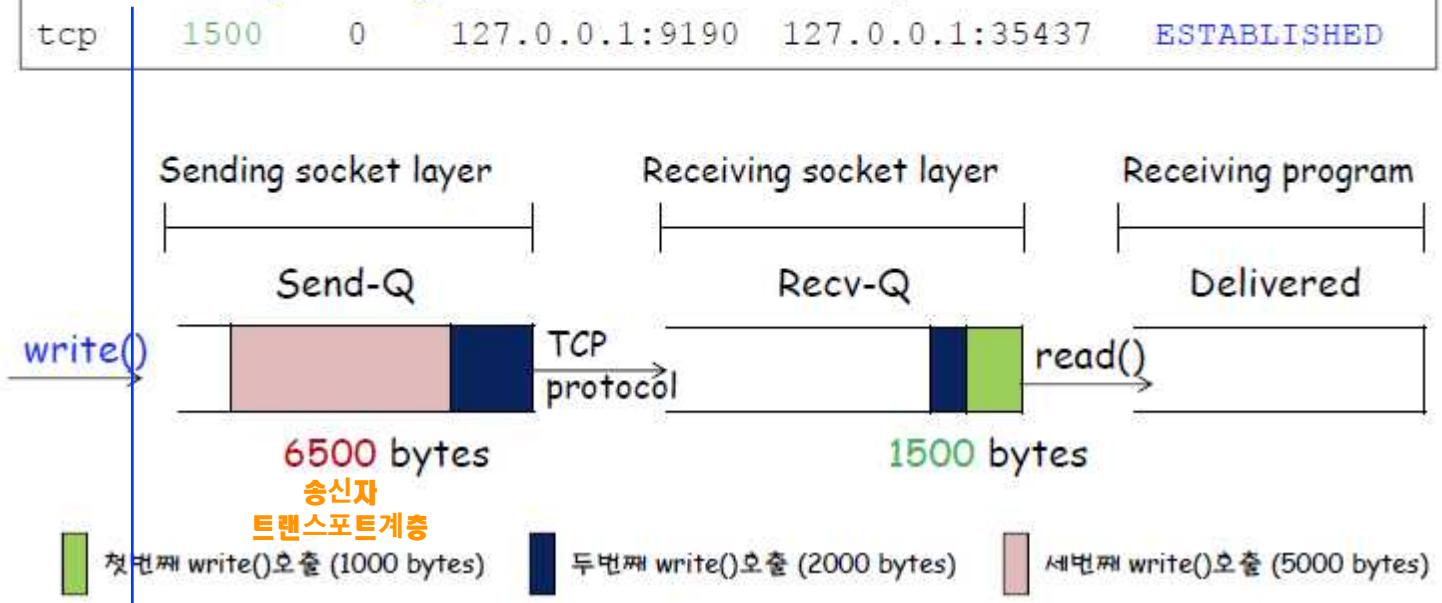
Example


```
rv = connect(sock,…);

rv = write(sock, buffer0, 1000);

rv = write(sock, buffer1, 2000);

rv = write(sock, buffer2, 5000);

close(sock);
```

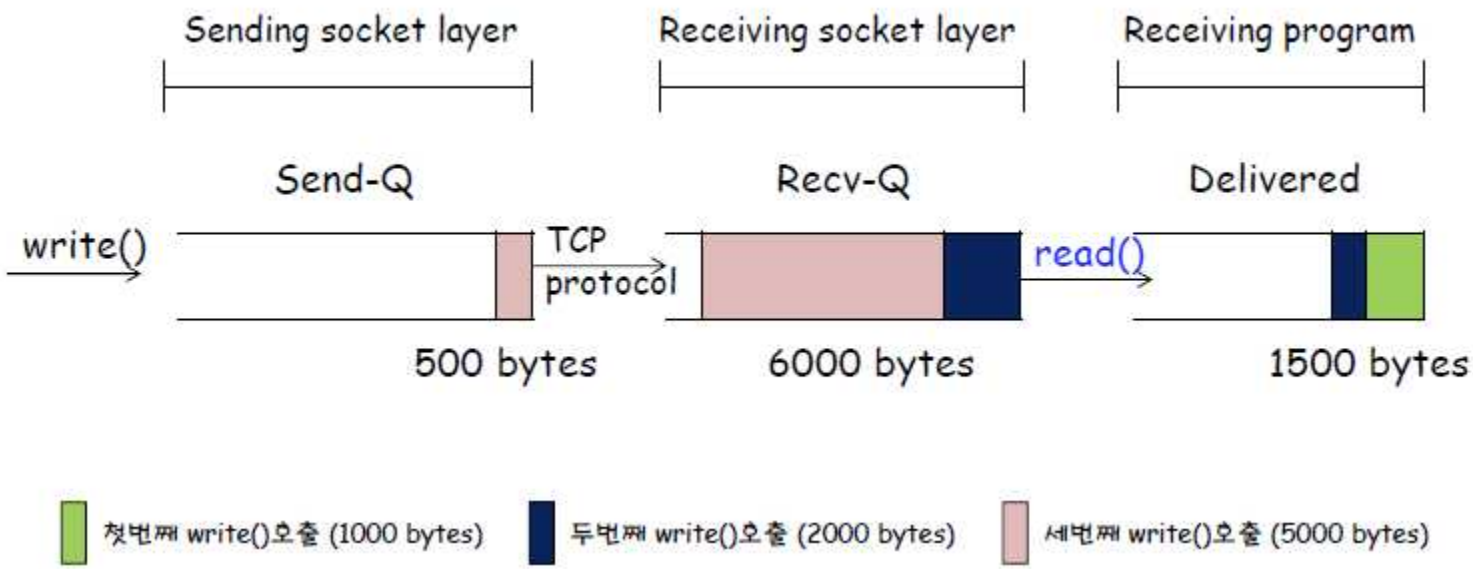
세 번의 write() 호출 뒤 세가지 큐의 상태


```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address   State
tcp       0   6500  127.0.0.1:35437 127.0.0.1:9190    ESTABLISHED
```

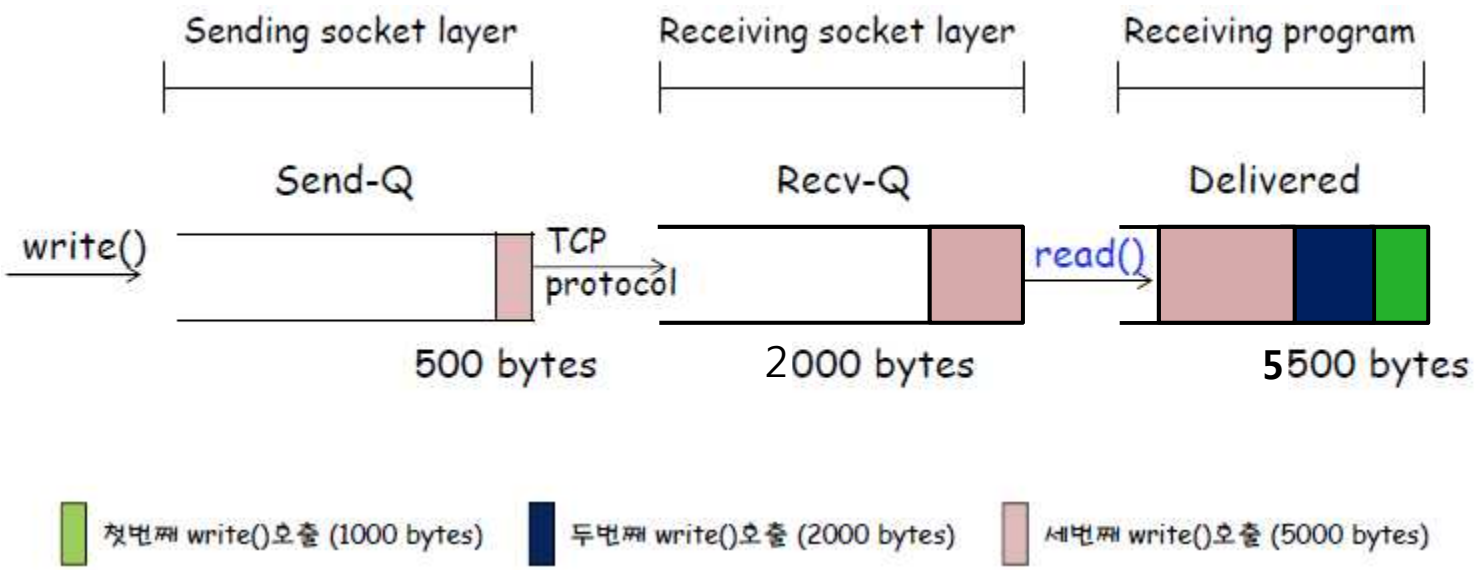
```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address   State
tcp    1500      0  127.0.0.1:9190  127.0.0.1:35437   ESTABLISHED
```



첫 read() 호출 뒤 세가지 큐의 상태



다른 read() 호출 뒤 세가지 큐의 상태



데드락의 위험성

데드락(Deadlock)

우리수준에서의 예시는
서버도 accept하자마자 read
클라도 connect되자마자 read

- 통신 양 종단이 상대방의 동작을 서로 기다리고 있어서 더 이상 진행이 되지 않고 멈추어 있는 현상

  예) 통신이 연결되자마자 클라이언트와 서버가 동시에 메시지를 받으려고만 할 때

flow control
〉수신자의 상황을 보고 보내는 양을 조절하는것
ex〉 슬라이딩 윈도우

- TCP의 흐름 제어(flow control) 기법으로 인한 데드락

congestion control
〉네트워크 상황을보고 보내는 양을 조절하는것
ex〉 slow start

  - 송신 측의 프로그램은 송신큐가 전부 채워질 때까지 write()를 연속적으로 호출할 수 있다.
  - 하지만 일단 송신큐가 가득 차게 되면, 수신큐로 어느 정도 데이터가 전송되어 송신큐에 여유가 생길 때까지 write()가 블로킹된다.

두 상황을 모두 고려해서 윈도우 사이즈를 결정하게됨

  - 만약 이때 수신큐까지 모두 차게 된다면, TCP 흐름 제어 기법이 작동하게 되는데, 수신 측의 read() 호출로 인해 수신큐를 비울 때까지 송신큐에서 어떠한 바이트도 전송을 못하게 한다.
    - 이러한 흐름 제어의 목적은 수신 측이 처리할 수 있는 것보다 송신 측이 더 빠르게 보내지 못하게 하기 위함이다.
  - 즉, 수신 프로그램이 read()를 호출하여 수신큐에 있는 일부 데이터를 전송 완료 큐로 이동시키기 전까지는 모든 데이터의 이동이 중지된다.

상대방(수신자)의 큐의 사이즈는 tcp헤더에 있는 window size를 통해 알 수 있다
〉ack같이 날아오는 데이터들의 헤더에 있으니까

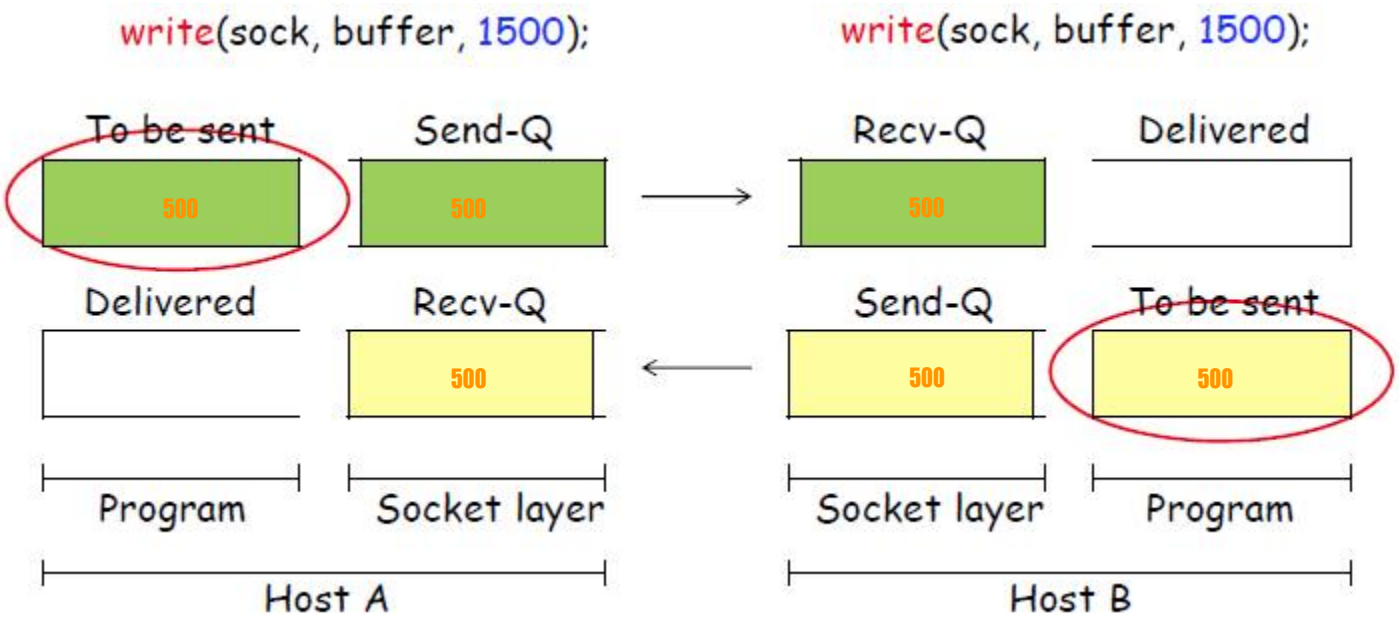
리시브 큐 쪽에서 window size == 0인것을 0보다 큰게 오기 전까지 안보낸다

이상태의 경우
송신측에서 주기적으로 1바이트에 헤더붙여서 보낸다(Prob segement)
〉수신측에서ack보내주면 그때부터 다시 전송시작

동시 write() 호출에 의한 데드락

- Send-Q와 Recv-Q의 크기를 500 바이트라고 가정
- 양 종단 프로그램이 동시에 1500 바이트를 송신하고자 할 경우

서로read를 해줘야 recv-q가 비어서 들어 갈 수가 있는데 write에서 블락걸린 상태이기 때문에 안됨

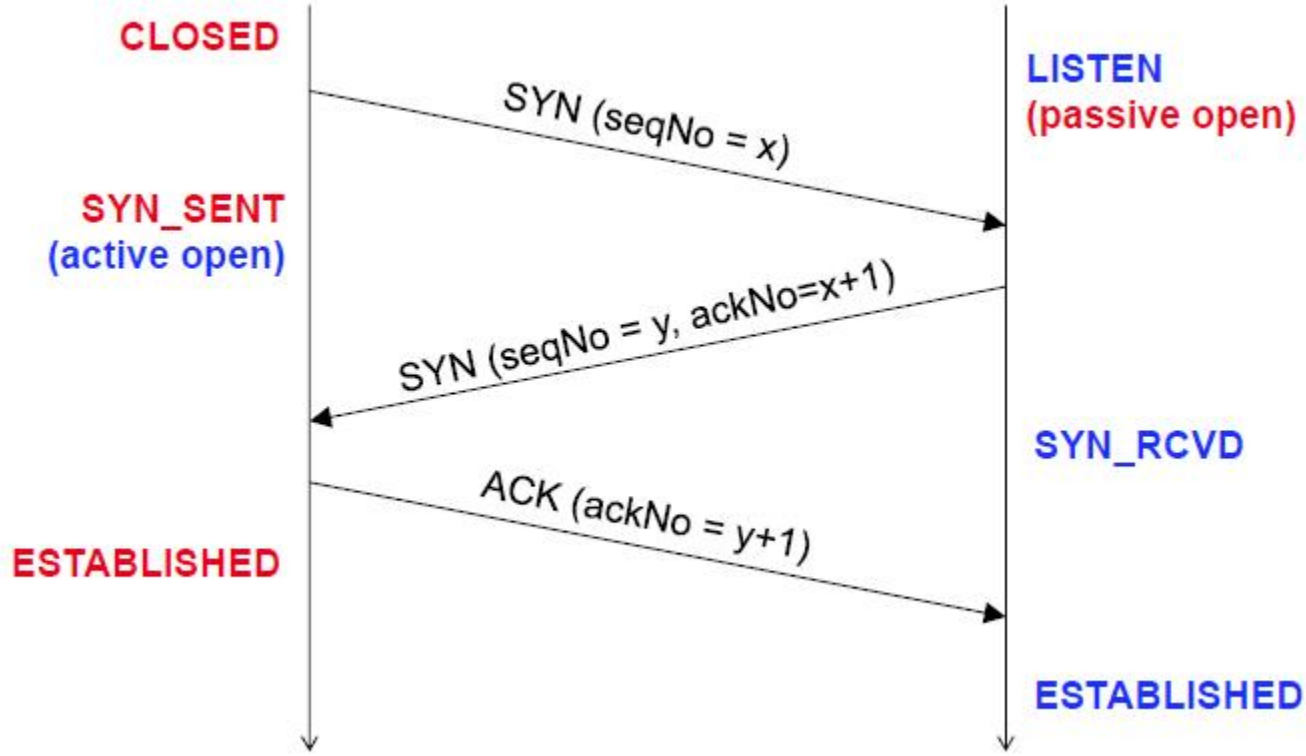


- 연결의 양 종단 프로그램이 동시에 SendQ + RecvQ보다 더 큰 버퍼를 가지고 write()를 호출하게 되면 데드락 발생
  - 이 경우, 쓰기가 완료되지 못할뿐더러 양 종단의 프로그램은 계속 블로킹된 상태로 남아있게 됨

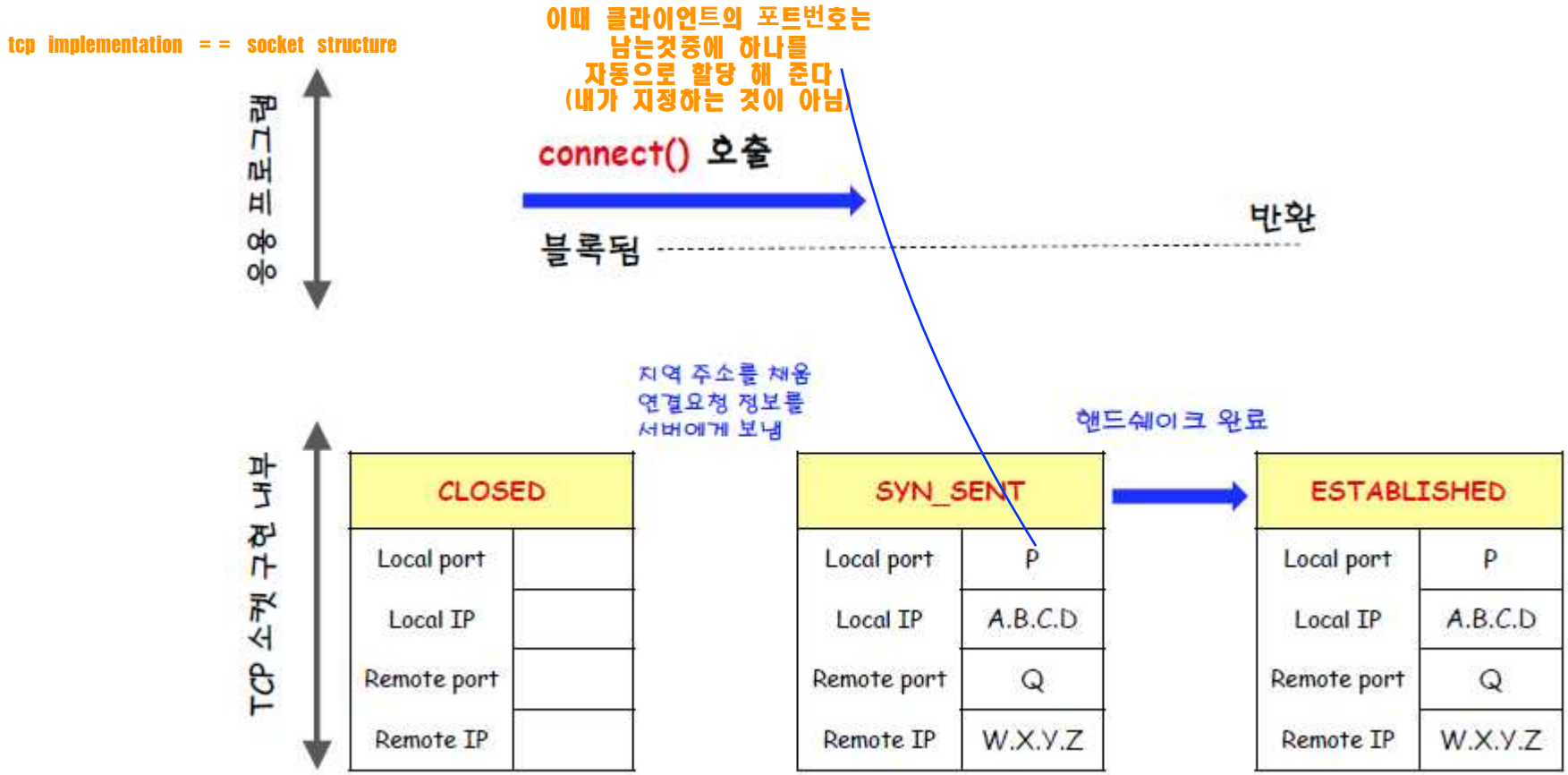
☢ 많은 양의 데이터를 서로 동시에 전송하지 않도록 유의하여 응용 프로토콜을 설계해야 함

TCP 소켓 생명 주기 1

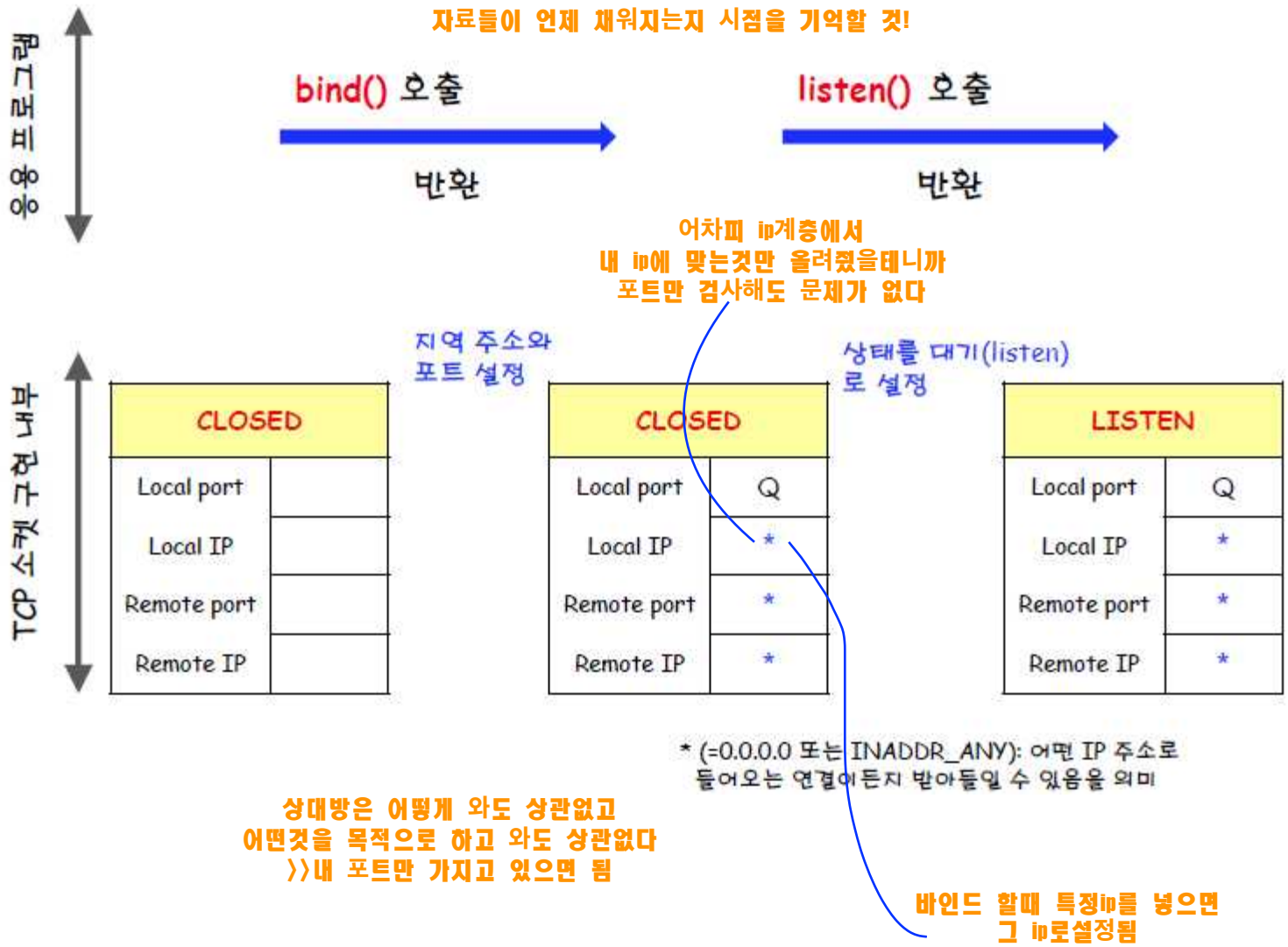
TCP 연결


CLOSED
LISTEN
(passive open)
SYN (seqNo = x)
SYN_SENT
(active open)
SYN (seqNo = y, ackNo=x+1)
SYN_RCVD
ACK (ackNo = y+1)
ESTABLISHED
ESTABLISHED

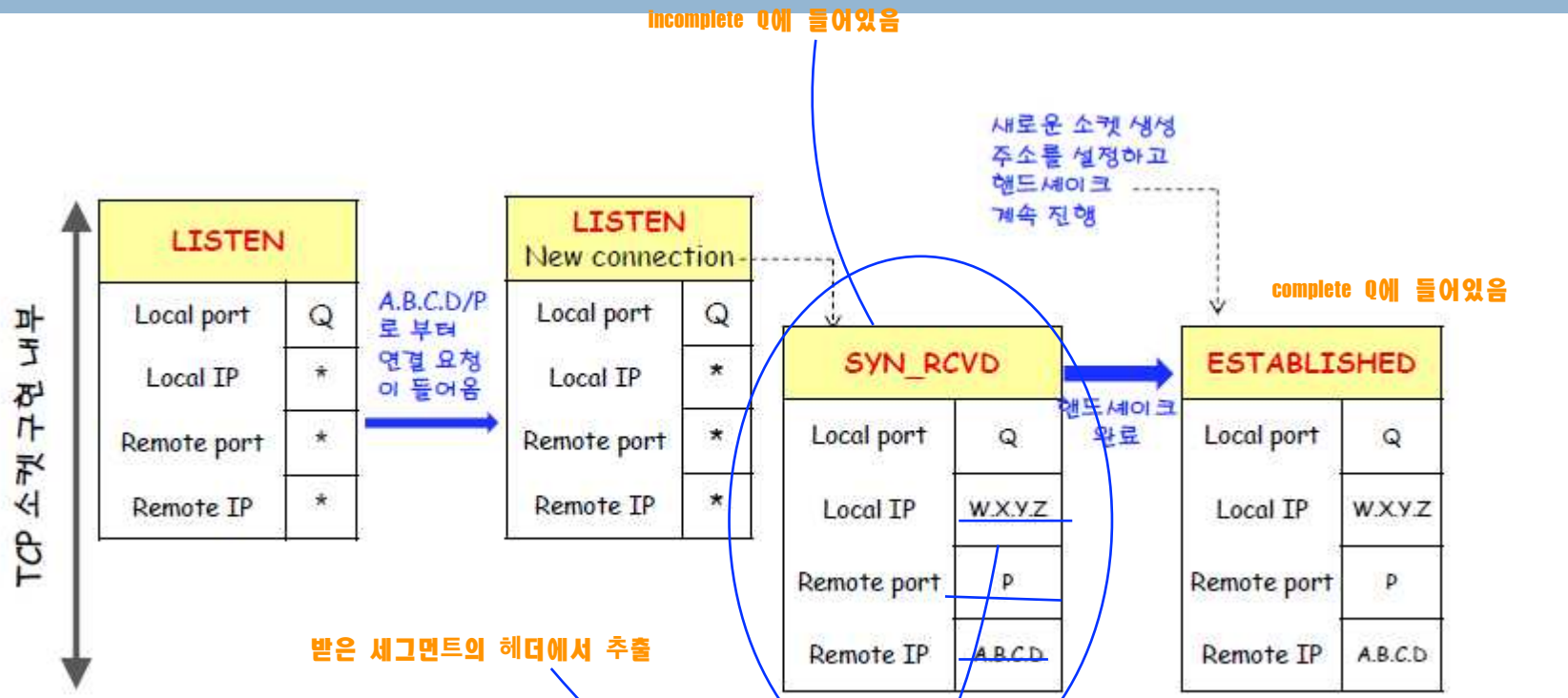
클라이언트에서 바라본 연결 설정 과정



서버에서 바라본 소켓 설정 과정



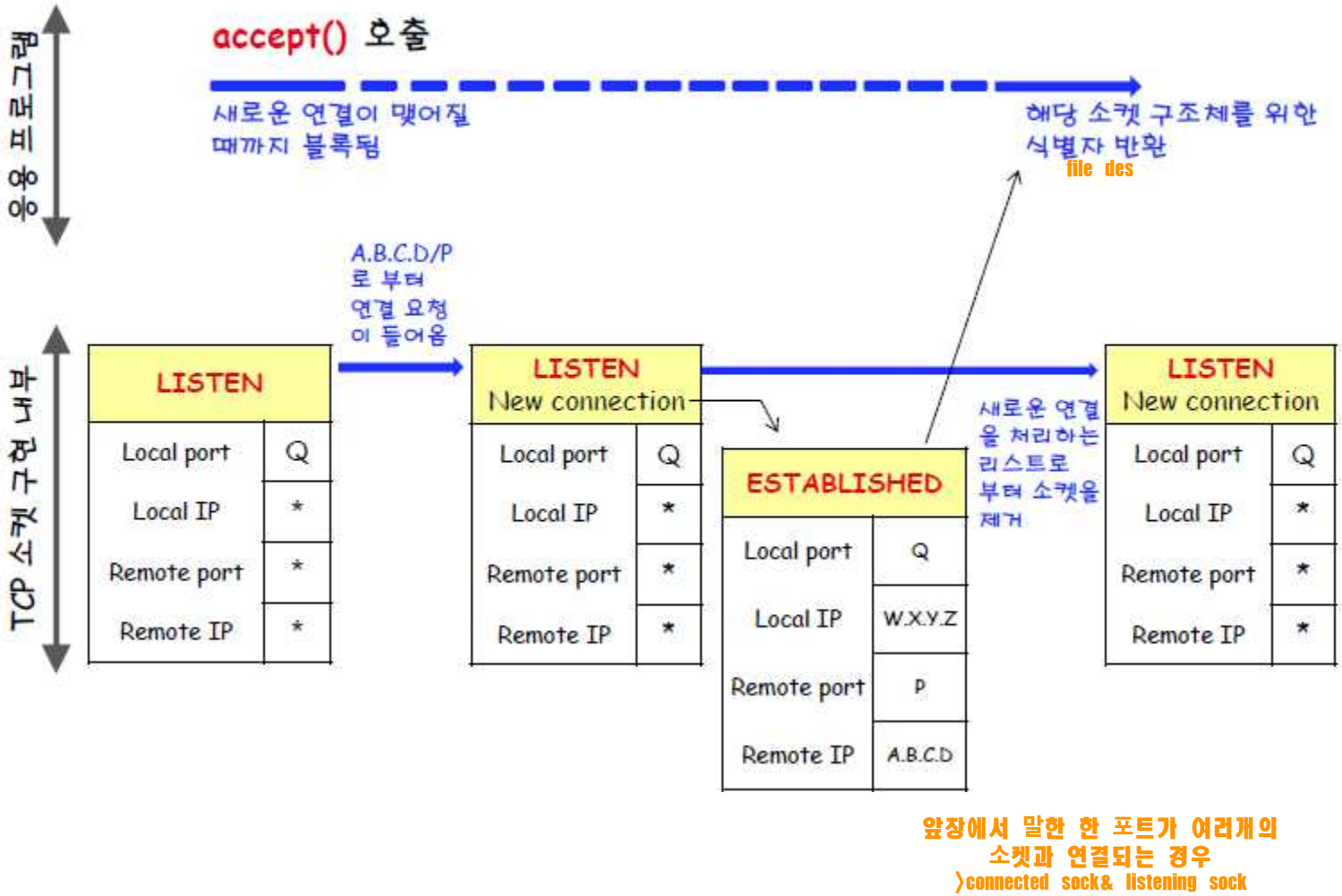
들어오는 연결 요청을 처리하는 과정




```
Active Internet connections
Proto      Recv-Q     Send-Q     Local Address          Foreign Address        State
tcp        0          0          0.0.0.0:Q              0.0.0.0:0              LISTEN
tcp        0          0          W.X.Y.Z:Q              A.B.C.D:P              SYN_RCVD
```

```
Active Internet connections
Proto      Recv-Q     Send-Q     Local Address          Foreign Address        State
tcp        0          0          0.0.0.0:Q              0.0.0.0:0              LISTEN
tcp        0          0          W.X.Y.Z:Q              A.B.C.D:P              ESTABLISHED
```

accept()를 수행하는 과정



앞장에서 말한 한 포트가 여러개의 소켓과 연결되는 경우
〉connected sock& listening sock

Demultiplexing process

Demultiplexing process

- The process of deciding to which socket an incoming packet should be delivered
  - The process of matching an incoming packet to a socket

패킷이 들어왔을때 여러개의 소켓이 있음
>어떤 소켓에 보내줘야하는지 결정하는 과정

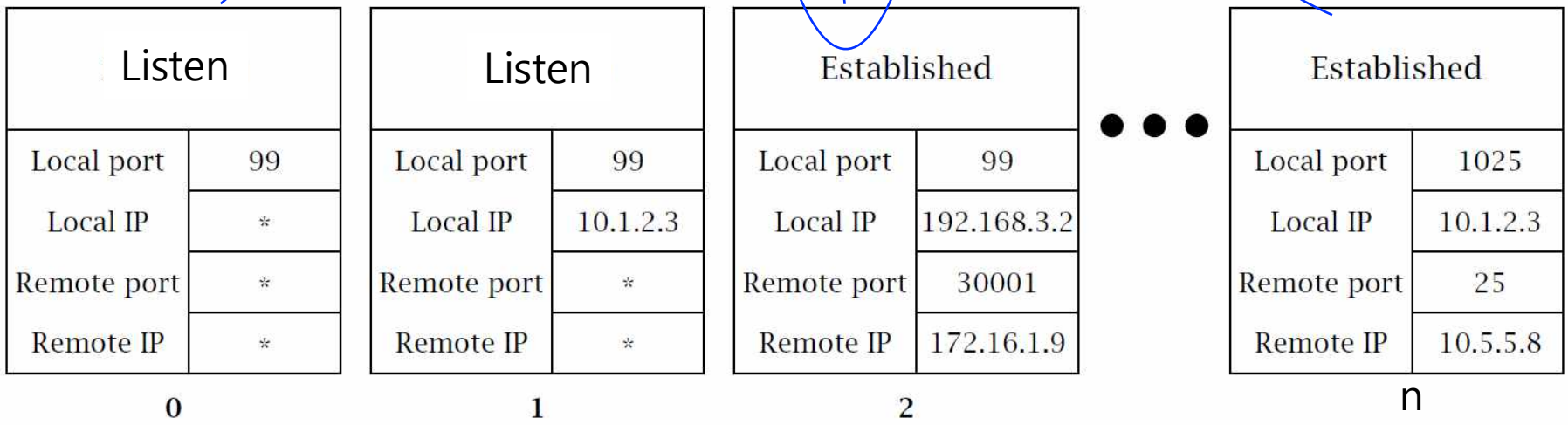
- The local port in the socket structure *must* match the destination port number in the incoming packet.
- Any address fields in the socket structure that contain the wildcard value (*) are considered to match *any* value in the corresponding field in the packet.
- If more than one socket structure matches an incoming packet for all four address fields, the one that matches using the fewest wildcards gets the packet.

받을 수 있는놈이 여러개면
여러개 중 와일드 카드가 더 적은
녀석에게 준다

Demultiplexing process

이녀석은 서버에 접속하는 클라이언트의 커넥티드 소켓이다
그냥 서버를 위한 소켓이라면 이런 형식이 나올 수 없다
〉〉리슨소켓으로부터 파생되어야 하는데
맞는 리슨소켓이 없으므로

- For example,
  - 10.1.2.3과 192.168.3.2, 두 개의 IP 주소를 사용하는 호스트



  - 송신자 IP가 172.16.1.10이고 송신 포트가 56789, 수신자 IP가 10.1.2.3이고 수신자 포트가 99인 패킷이 들어오는 경우

=〉 이 경우 1번으로 들어와서 새로운 파생소켓을 만든다

2번소켓을 할당할때
0으로부터 파생된 것이다.
local ip가 1과 다르니까

만약 local ip == 10.1.2.3으로 들어왔다면
0,1 둘다 해당되지만
와일드카드가 더 적은
1에서부터 파생되어 새로운 소켓이 생성될것이다