

## Chapter 04 TCP 서버-클라이언트

- TCP 서버-클라이언트의 기본 구조와 동작 원리를 이해한다.
- TCP 응용 프로그램 작성에 필요한 핵심 소켓 함수를 익힌다.
- IPv4와 IPv6 기반 TCP 서버-클라이언트를 작성할 수 있다.

# 목차

01 TCP 서버-클라이언트 구조

02 TCP 서버-클라이언트 분석

03 TCP 서버-클라이언트(IPv6)

UDP  
vs

# 01 TCP 서버-클라이언트 구조



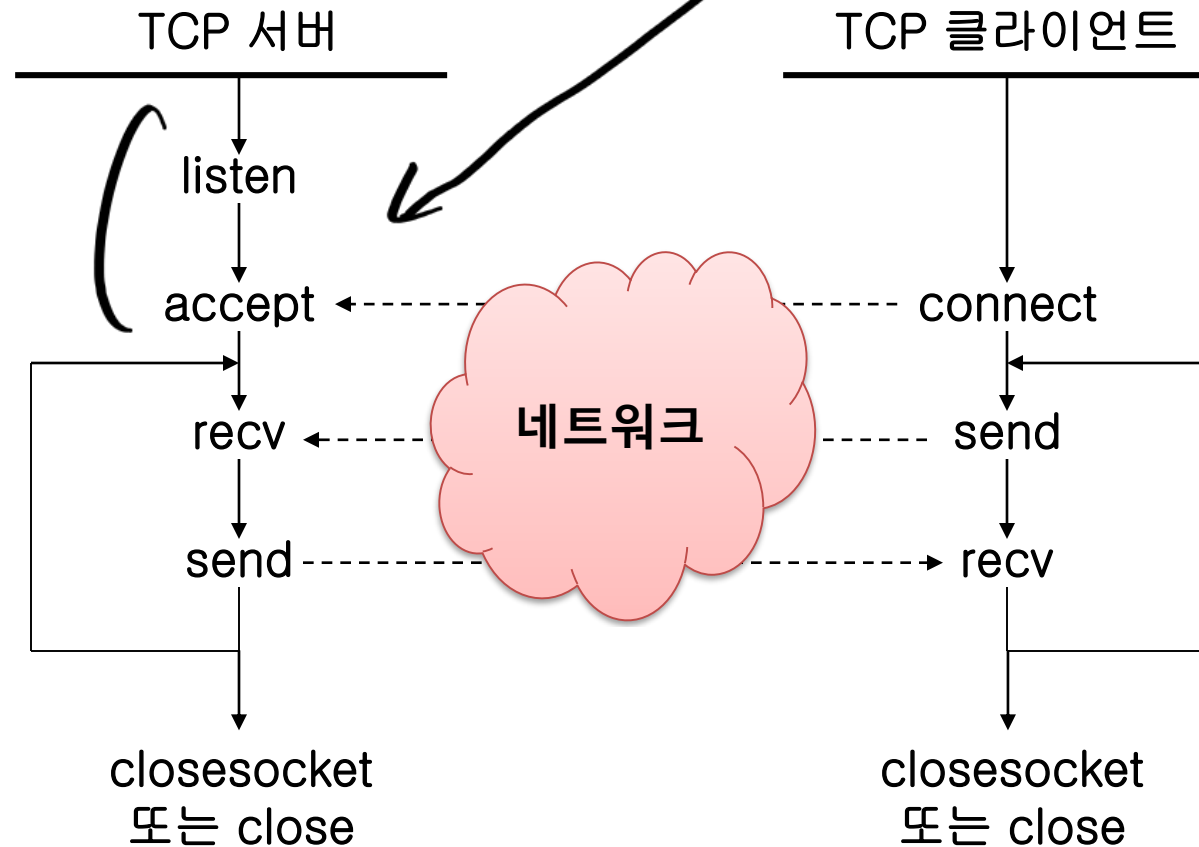
# TCP 서버-클라이언트 구조 (1)

## ■ 웹 서버-클라이언트 동작



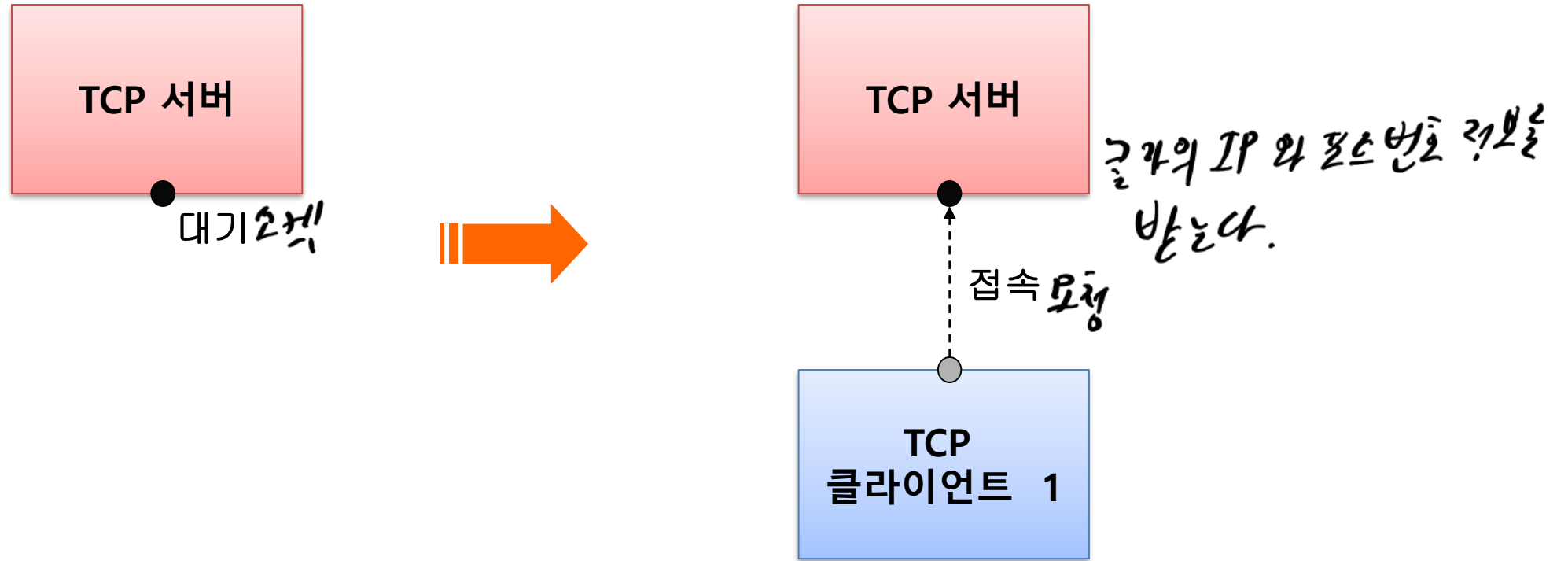
# TCP 서버-클라이언트 구조 (2)

## ■ TCP 서버-클라이언트 핵심 동작



# TCP 서버-클라이언트 구조 (3)

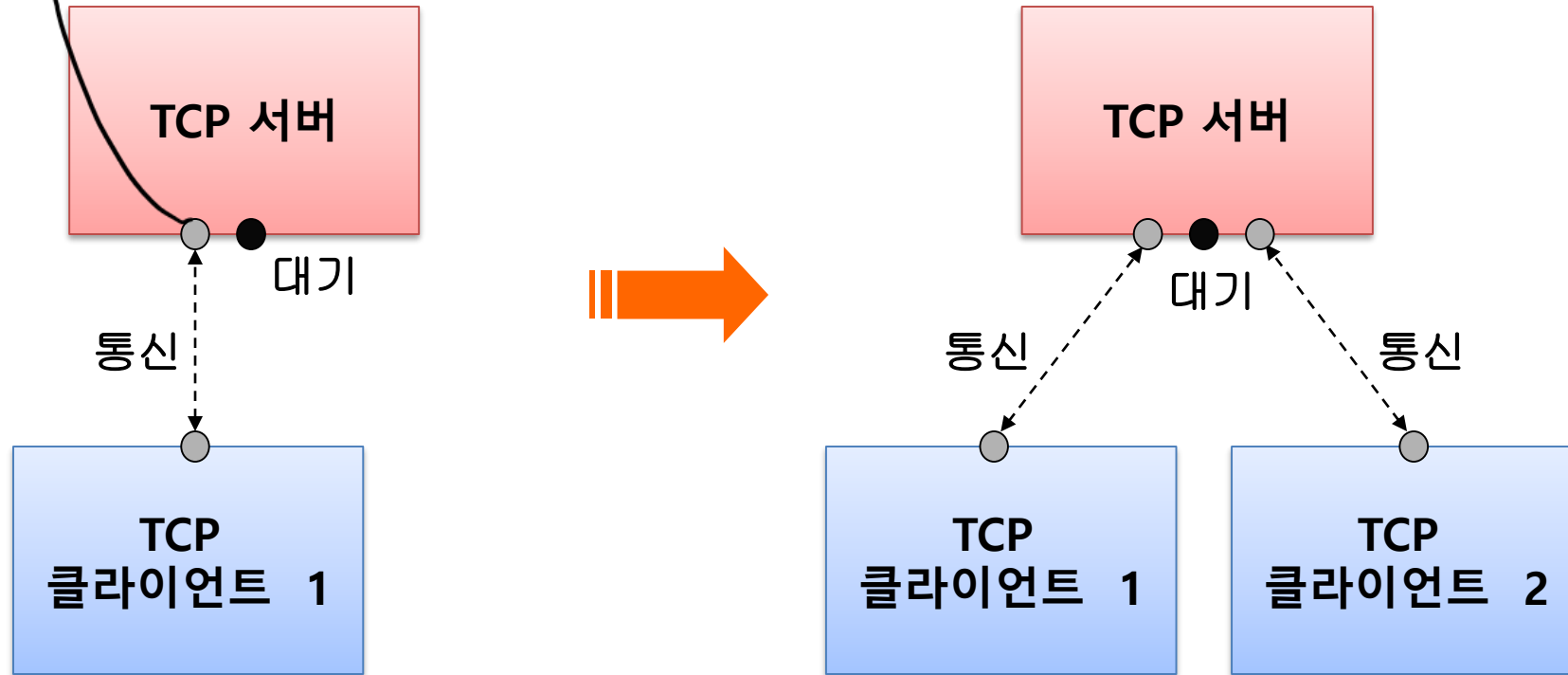
## ■ TCP 서버-클라이언트 동작 원리



# TCP 서버-클라이언트 구조 (4)

## ■ TCP 서버-클라이언트 동작 원리

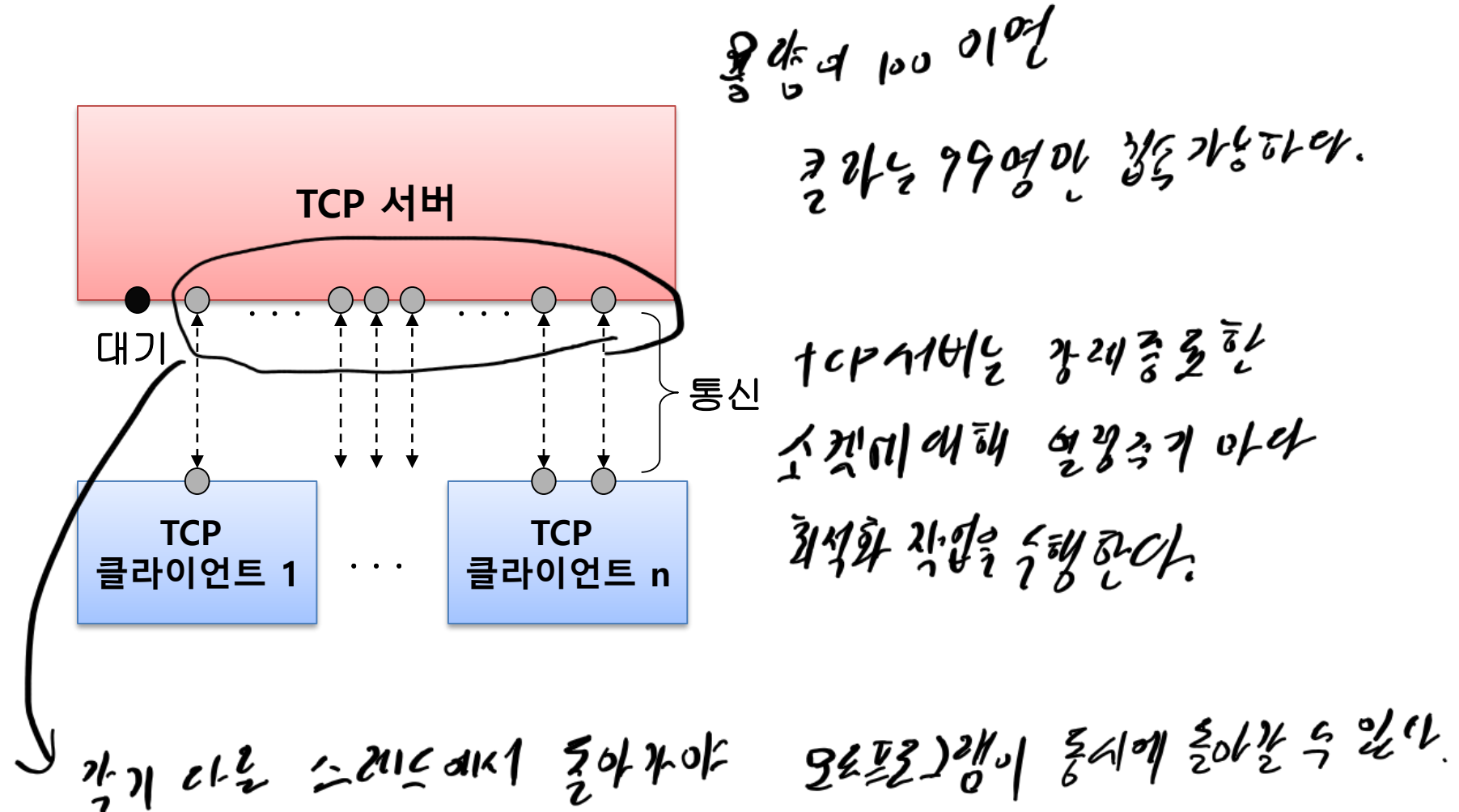
클라이언트를 전달할 소켓을 따로 생성하고 (생략된 부분)  
다시 대기소켓으로 돌아온다.





# TCP 서버-클라이언트 구조 (5)

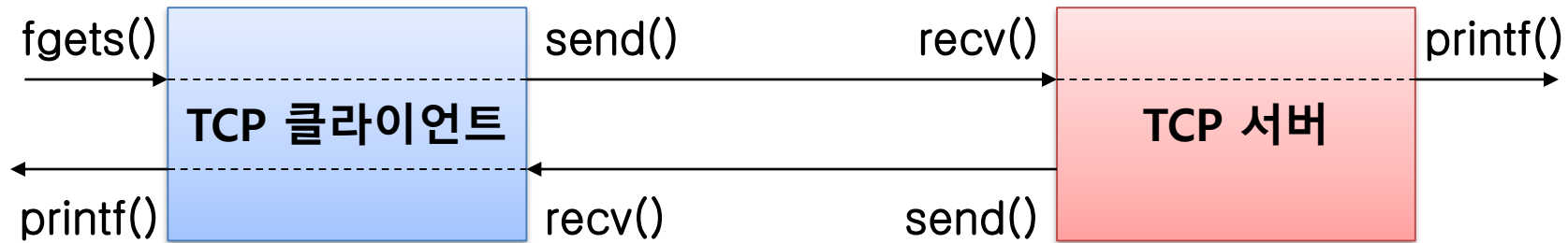
## ■ 하나의 TCP 서버와 여러 TCP 클라이언트의 통신



# TCP 서버-클라이언트 구조 (6)

## ■ TCP 서버-클라이언트 예제 동작

*echo (에코)*



# TCP 서버-클라이언트 구조 (7)

## ■ 실습 4-1 TCP 서버-클라이언트 작성

- TCPServer.cpp
  - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter04/TCPServer/TCPServer.cpp>
  - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter04/TCPServer.cpp>
- TCPClient.cpp
  - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter04/TCPClient/TCPClient.cpp>
  - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter04/TCPClient.cpp>

# TCP 서버-클라이언트 구조 (8)

## ■ 실습 4-2 TCP 서버-클라이언트 테스트

- TCP 서버를 실행 - 초기에는 아무것도 출력되지 않음



- 명령 프롬프트를 실행한 후 `netstat -a -n -p tcp | findstr 9000` 명령을 실행

# TCP 서버-클라이언트 구조 (9)

- 실습 4-2 TCP 서버-클라이언트 테스트
  - TCP 클라이언트 실행

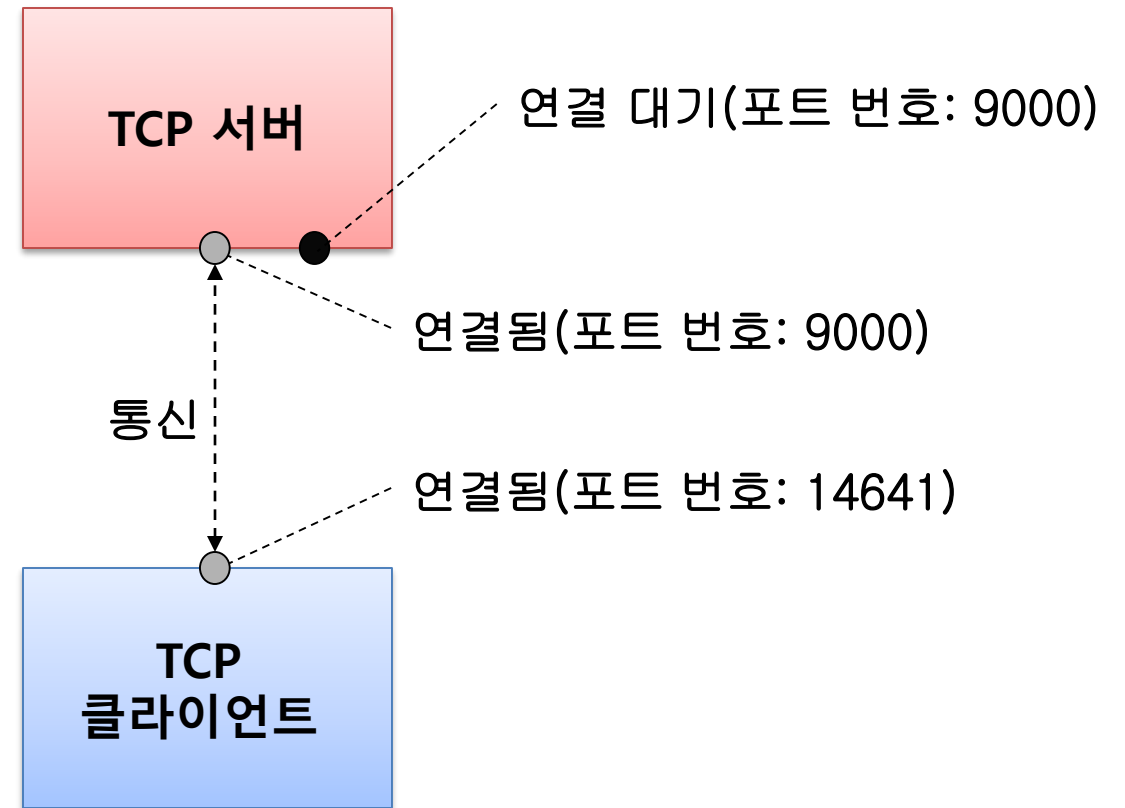


# TCP 서버-클라이언트 구조 (10)

## ■ 실습 4-2 TCP 서버-클라이언트 테스트

- 다시 `netstat -a -n -p tcp | findstr 9000` 명령을 실행

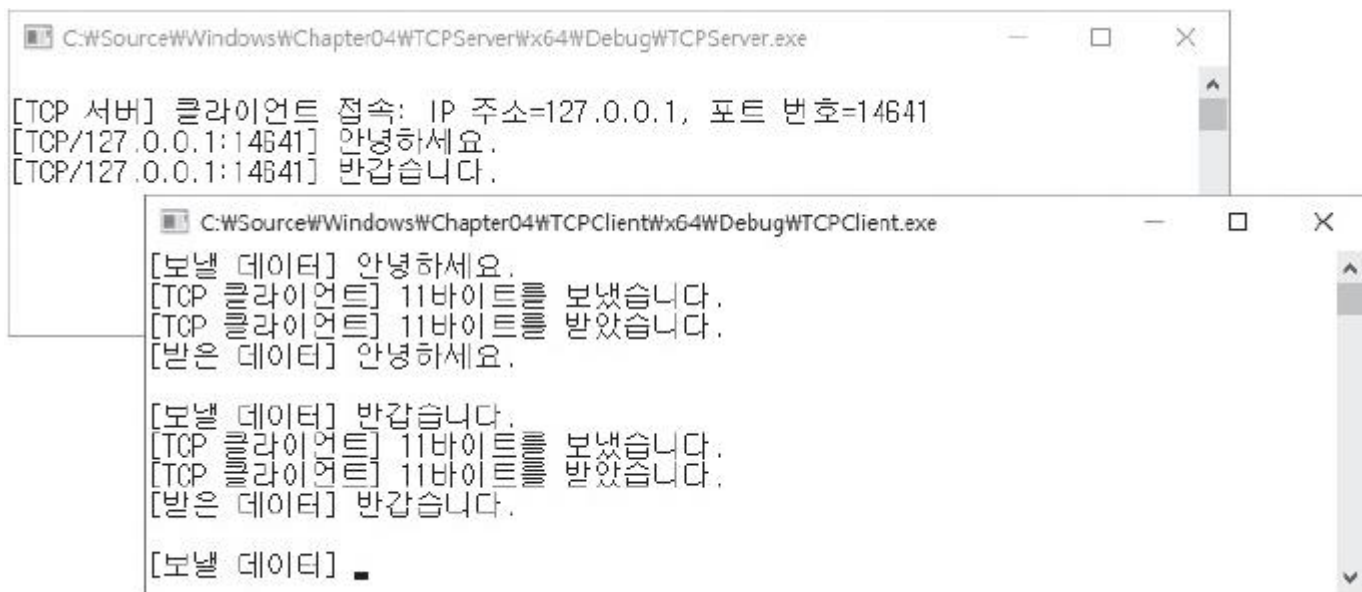
```
C:\Windows\system32\cmd.exe
C:\Users\swkin>netstat -a -n -p tcp | findstr 9000
TCP    0.0.0.0:9000      0.0.0.0:0        LISTENING
TCP    127.0.0.1:9000   127.0.0.1:14641  ESTABLISHED
TCP    127.0.0.1:14641  127.0.0.1:9000   ESTABLISHED
C:\Users\swkin>
```



# TCP 서버-클라이언트 구조 (11)

## ■ 실습 4-2 TCP 서버-클라이언트 테스트

- 클라이언트에서 글자를 입력하고 [Enter]



The screenshot shows two overlapping Windows command prompt windows. The top window, titled 'C:\Source\Windows\Chapter04\TCP\Server\Wx64\Debug\TCPServer.exe', displays the following text: '[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=14641', '[TCP/127.0.0.1:14641] 안녕하세요.', and '[TCP/127.0.0.1:14641] 반갑습니다.'. The bottom window, titled 'C:\Source\Windows\Chapter04\TCP\Client\Wx64\Debug\TCPClient.exe', displays the following text: '[보낼 데이터] 안녕하세요.', '[TCP 클라이언트] 11바이트를 보냈습니다.', '[TCP 클라이언트] 11바이트를 받았습니다.', '[받은 데이터] 안녕하세요.', '[보낼 데이터] 반갑습니다.', '[TCP 클라이언트] 11바이트를 보냈습니다.', '[TCP 클라이언트] 11바이트를 받았습니다.', '[받은 데이터] 반갑습니다.', and '[보낼 데이터] .'. The client window has a cursor at the end of the last line.

```
C:\Source\Windows\Chapter04\TCP\Server\Wx64\Debug\TCPServer.exe
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=14641
[TCP/127.0.0.1:14641] 안녕하세요.
[TCP/127.0.0.1:14641] 반갑습니다.

C:\Source\Windows\Chapter04\TCP\Client\Wx64\Debug\TCPClient.exe
[보낼 데이터] 안녕하세요.
[TCP 클라이언트] 11바이트를 보냈습니다.
[TCP 클라이언트] 11바이트를 받았습니다.
[받은 데이터] 안녕하세요.

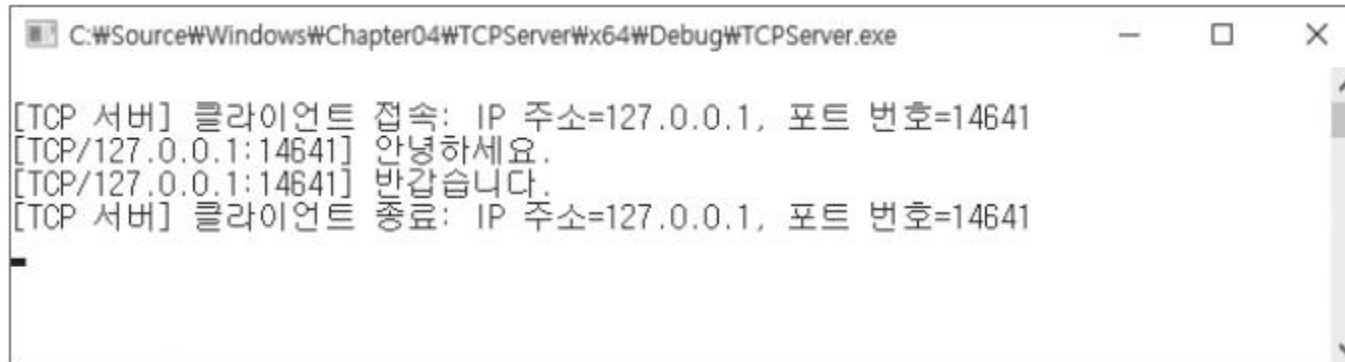
[보낼 데이터] 반갑습니다.
[TCP 클라이언트] 11바이트를 보냈습니다.
[TCP 클라이언트] 11바이트를 받았습니다.
[받은 데이터] 반갑습니다.

[보낼 데이터] .
```

# TCP 서버-클라이언트 구조 (12)

## ■ 실습 4-2 TCP 서버-클라이언트 테스트

- 글자를 입력하지 않은 상태에서 [Enter]



A screenshot of a Windows command prompt window titled "C:\Source\Windows\Chapter04\TCPServer\x64\Debug\TCPServer.exe". The window displays the following text:

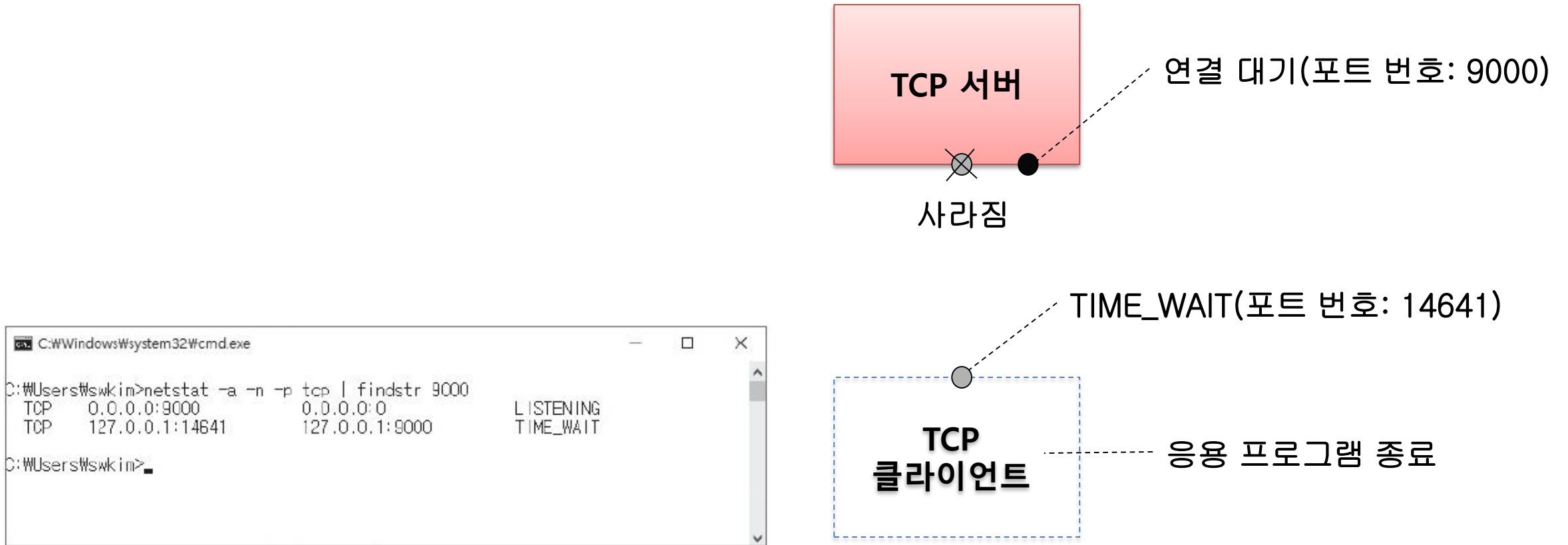
```
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=14641  
[TCP/127.0.0.1:14641] 안녕하세요.  
[TCP/127.0.0.1:14641] 반갑습니다.  
[TCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=14641
```



# TCP 서버-클라이언트 구조 (13)

## ■ 실습 4-2 TCP 서버-클라이언트 테스트

- 다시 `netstat -a -n -p tcp | findstr 9000` 명령을 실행



- 5분 후 다시 `netstat -a -n -p tcp | findstr 9000` 명령을 실행하면 2단계의 상태로 돌아옴

## 02 TCP 서버-클라이언트 분석



# TCP 서버-클라이언트 분석 (1)

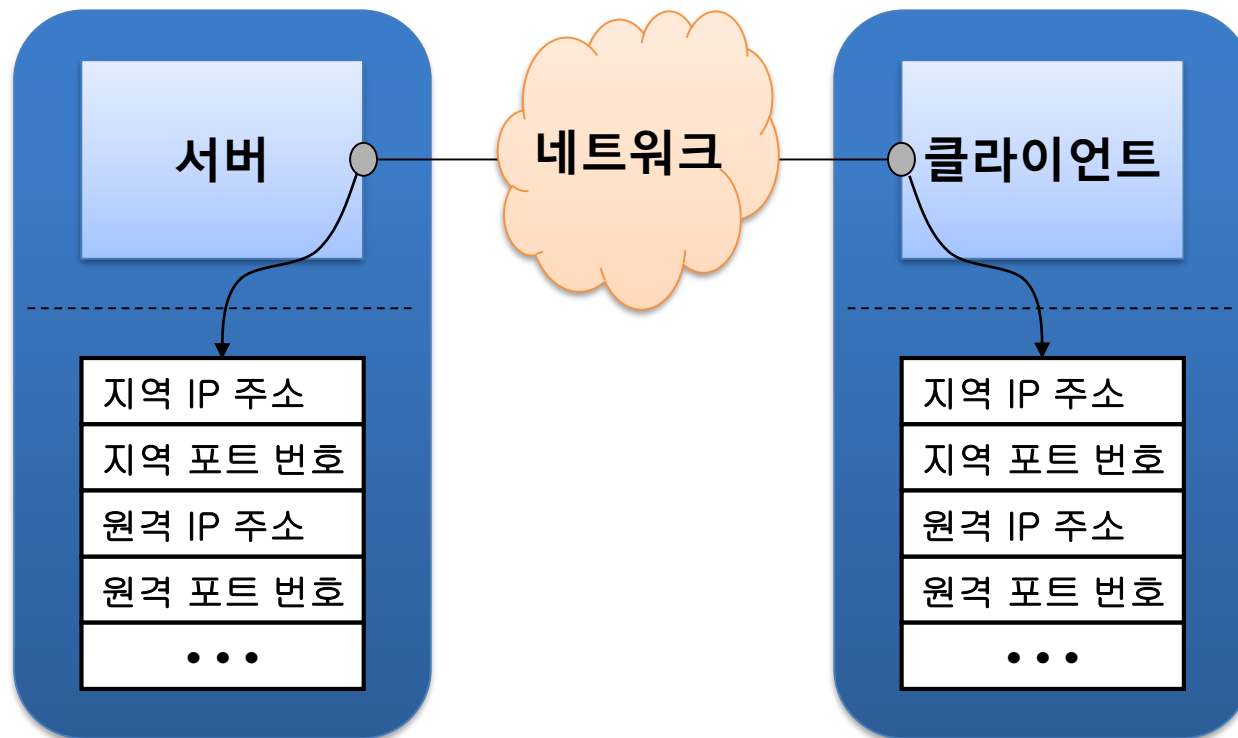
## ■ 응용 프로그램 통신을 위해 결정해야 할 요소

- 프로토콜 : 통신 규약으로, 소켓을 생성할 때 결정 *~ 사용할 프로토콜을 결정해야 함*
- 지역 IP 주소와 지역 포트 번호 : 서버 또는 클라이언트 자신의 주소
- 원격 IP 주소와 원격 포트 번호 : 서버 또는 클라이언트가 통신하는 상대의 주소 *주소 정보를 알아야 한다.*

*상대 측*

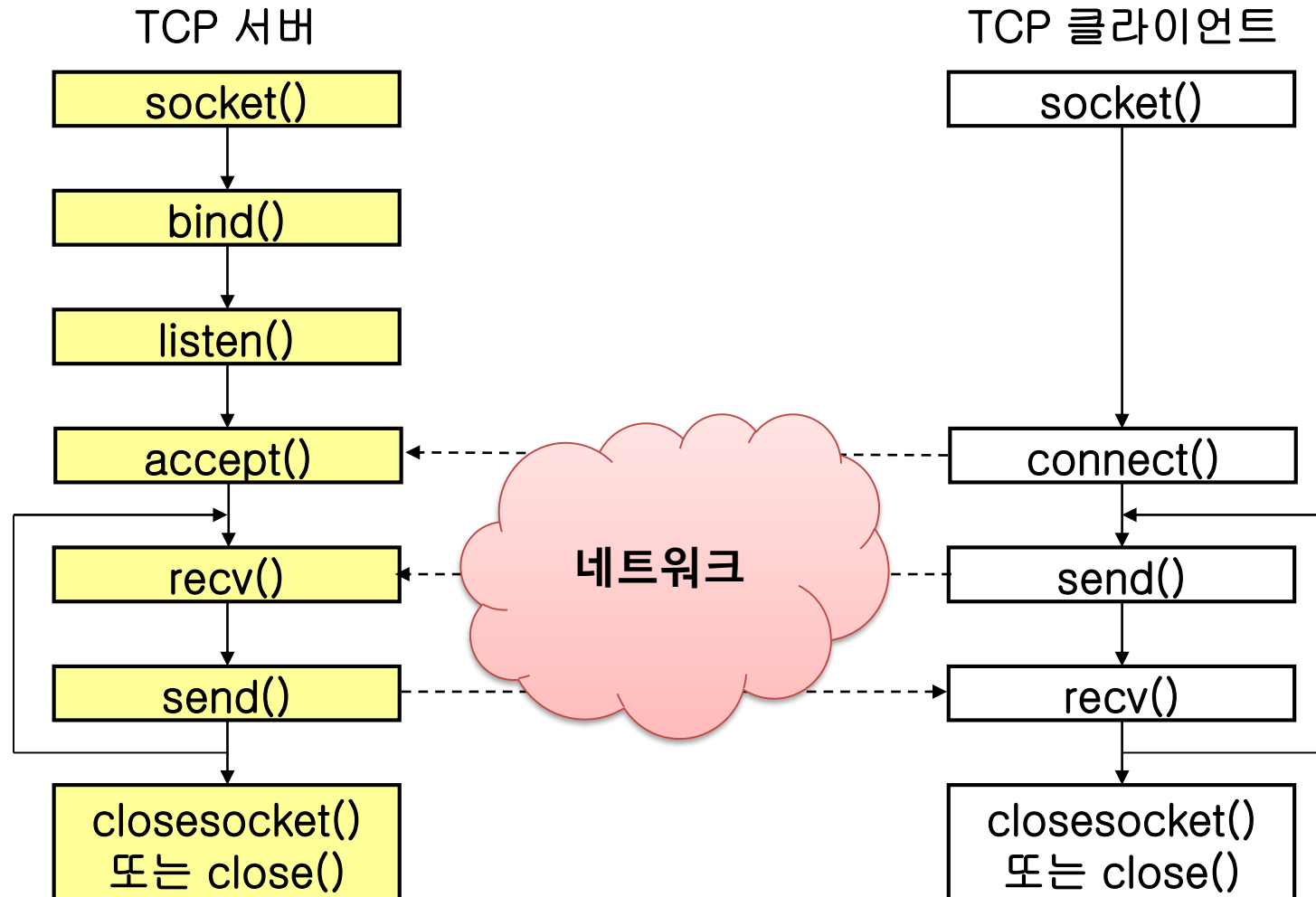
응용 프로그램

운영체제



# TCP 서버-클라이언트 분석 (2)

## ■ TCP 서버 함수



# TCP 서버-클라이언트 분석 (3)

이동자리를 알수 있게 묶어주는 함수

## ■ bind() 함수

- bind() 함수는 소켓의 지역 IP 주소와 지역 포트 번호를 결정

윈도우

```
#include <winsock2.h>
```

```
int bind(
```

① SOCKET sock, *가게소켓*

② const struct sockaddr \*addr,

③ int addrlen

```
);
```

성공: 0, 실패: SOCKET\_ERROR

리눅스

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int bind(
```

① int sock,

② const struct sockaddr \*addr,

③ socklen\_t addrlen

```
);
```

성공: 0, 실패: -1

# TCP 서버-클라이언트 분석 (4)

## ■ listen() 함수

- listen() 함수는 소켓의 TCP 상태를 LISTENING으로 변경

윈도우

```
#include <winsock2.h>
```

```
int listen(
```

```
    ① SOCKET sock,
```

```
    ② int backlog
```

```
);
```

→ 애기 소켓

→ 접속을 대기하는 큐

성공: 0, 실패: SOCKET\_ERROR

리눅스

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int listen(
```

```
    ① int sock,
```

```
    ② int backlog
```

```
);
```

성공: 0, 실패: -1

# TCP 서버-클라이언트 분석 (5)

## ■ accept() 함수

- accept() 함수는 클라이언트 접속을 수용하고, 접속한 클라이언트와 통신할 수 있는 새로운 소켓을 생성하여 리턴

윈도우

```
#include <winsock2.h>
SOCKET accept(
    ① SOCKET sock,
    ② struct sockaddr *addr,
    ③ int *addrlen
);
```

성공: 새로운 소켓, 실패: INVALID\_SOCKET

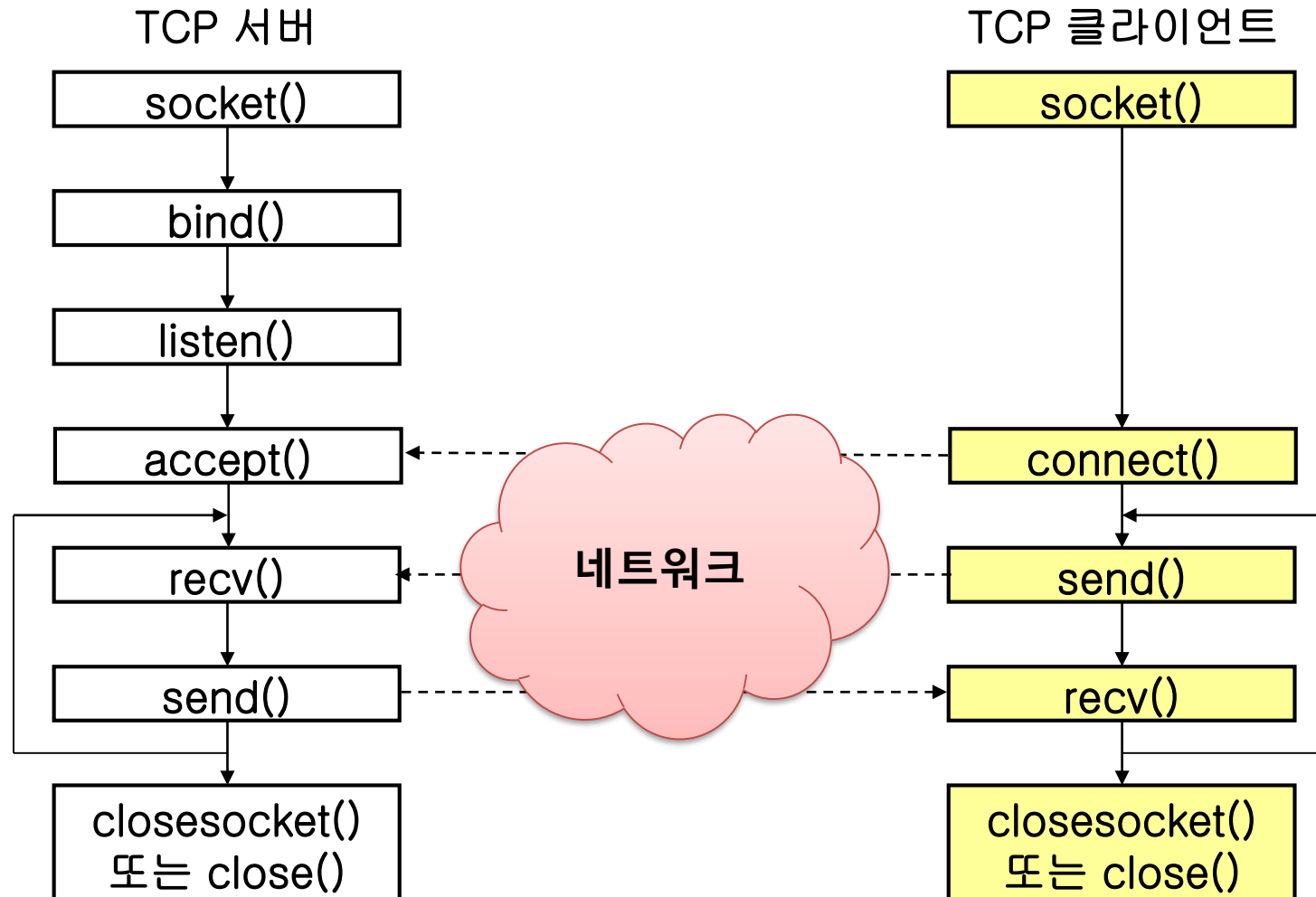
리눅스

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(
    ① int sock,
    ② struct sockaddr *addr,
    ③ socklen_t *addrlen
);
```

성공: 새로운 소켓, 실패: -1

# TCP 서버-클라이언트 분석 (6)

## ■ TCP 클라이언트 함수





# TCP 서버-클라이언트 분석 (7)

## ■ connect() 함수

- connect() 함수는 TCP 프로토콜 수준에서 서버와 논리적 연결을 설정

connect 함수가 bind() 역할을兼ね어준다 (자신의 주소와 포트 번호를 입력해준다)

윈도우

```
#include <winsock2.h>
```

```
int connect(
```

① SOCKET sock,

② const struct sockaddr \*addr,

③ int addrlen

```
);
```

성공: 0, 실패: SOCKET\_ERROR

→ 서버의 주소, 포트 번호의 정보가 들어갈 구조체

리눅스

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int connect(
```

① int sock,

② const struct sockaddr \*addr,

③ socklen\_t addrlen

```
);
```

성공: 0, 실패: -1

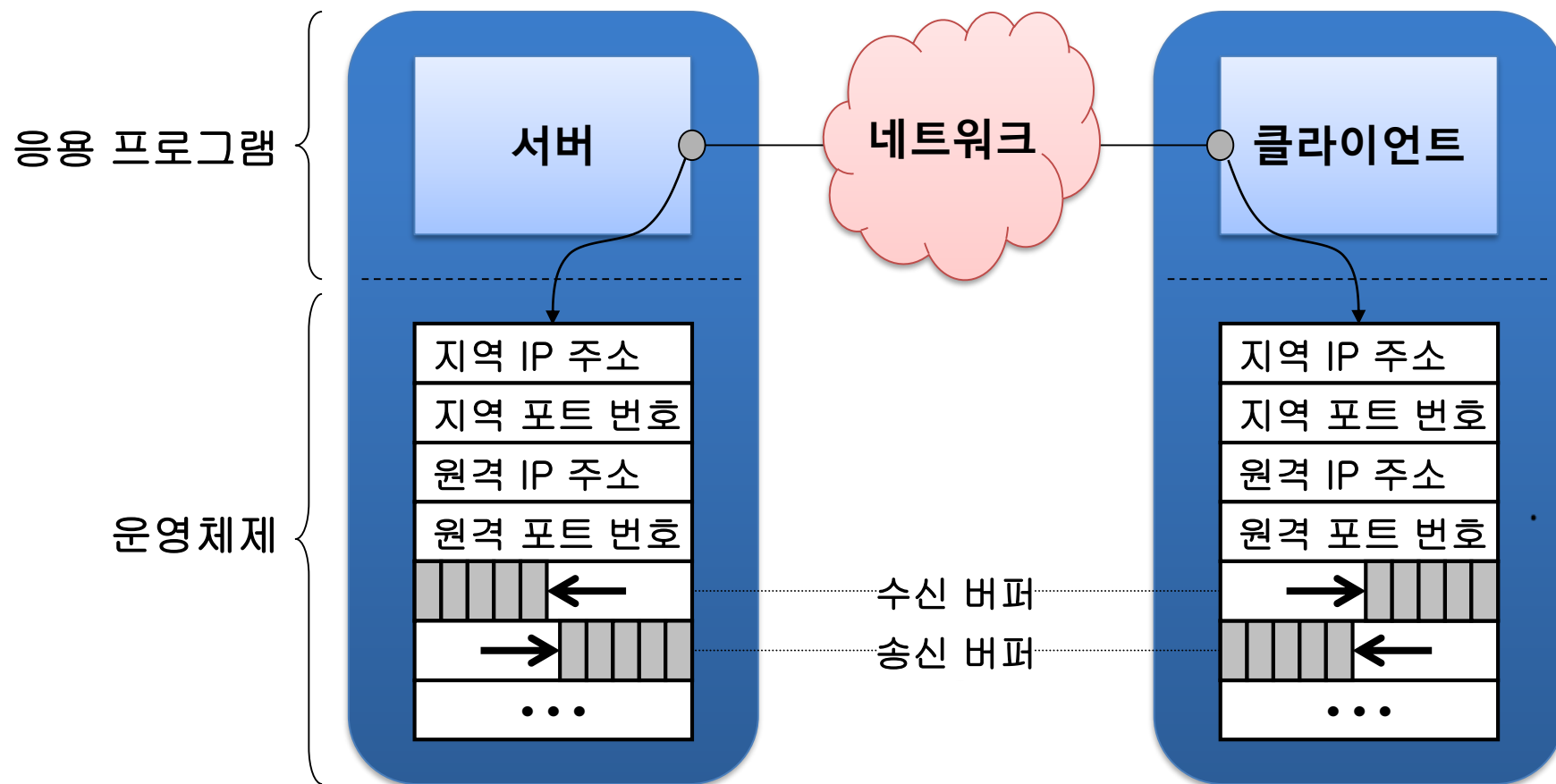
# TCP 서버-클라이언트 분석 (8)

## ■ TCP 데이터 전송 함수

- 기본이 되는 함수는 send() 함수, recv() 함수
- UDP에서 주로 사용하는 sendto() 함수, recvfrom() 함수
- 윈도우 전용 함수로 WSASend\*(), WSARecv\*() 확장 함수
- 리눅스 전용 함수로 write() 함수, read() 함수

# TCP 서버-클라이언트 분석 (9)

## ■ 소켓 데이터 구조체(2)



UDP: 잘못된 데이터는 버려버림  
 ↳ 송신 버퍼가 없다.  
 send: 운영체제 커널에  
 잠시 많이 저장되므로 나중

send() 가 제대로 return 되길 원하는 것은 송신 버퍼에 많이 들어갈수록 [서버가 데이터를  
 바깥으로 보낼 때]

# TCP 서버-클라이언트 분석 (10)

TCP가 재전송이 가능한 이점은 버퍼에

값이 남고 있기 때문 (recv를 했다고 확인이 되면  
버퍼를 삭제한다)

## ■ send() 함수

- send() 함수는 응용 프로그램의 데이터 전송을 위해 운영체제의 송신 버퍼에 데이터를 복사하고 리턴

윈도우

```
#include <winsock2.h>
```

```
int send(
```

```
    ① SOCKET sock,
```

```
    ② const char *buf,
```

```
    ③ int len,
```

```
    ④ int flags
```

```
);
```

성공: 보낸 바이트 수, 실패: SOCKET\_ERROR

→ 보내자 하듯 데이터의 크기

리눅스

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
ssize_t send(
```

```
    ① int sock,
```

```
    ② const void *buf,
```

```
    ③ size_t len,
```

```
    ④ int flags
```

```
);
```

성공: 보낸 바이트 수, 실패: -1

송신  
✓  
운영체제에 버퍼가 보낸 데이터의  
크기보다 작을 경우

방법 1) 여러번 호출해서 보낼 것.

↳ Non-Blocking 소켓

방법 2) 통신이 완료될 때까지 기다린다.

↳ Blocking 소켓을 기본 권장

# TCP 서버-클라이언트 분석 (11)

## ■ recv() 함수

- recv() 함수는 운영체제의 수신 버퍼에 도착한 데이터를 응용 프로그램 버퍼에 복사하고 리턴

윈도우

```
#include <winsock2.h>
```

```
int recv(
```

① SOCKET sock,

② char \*buf,

③ int len,

④ int flags

```
);
```

→ 여러 개의 패킷으로 나누어서 올 수도 있다. 콜때마다 recv 한다.

WAITALL: 받은 데이터의 크기를 알 경우

성공: 받은 바이트 수 또는 0(연결 종료 시), 실패: SOCKET\_ERROR

리눅스

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
ssize_t recv(
```

① int sock,

② void \*buf,

③ size\_t len,

④ int flags

```
);
```

성공: 받은 바이트 수 또는 0(연결 종료 시), 실패: -1

# TCP 서버-클라이언트 분석 (12)

## ■ recv() 함수

- recv() 함수는 두 종류의 성공적인 리턴을 할 수 있음
  - 수신 버퍼에 데이터가 도달한 경우
  - 접속이 정상 종료한 경우

## 03 TCP 서버-클라이언트(IPv6)



# TCP 서버-클라이언트(IPv6) (1)

## ■ IPv4 코드를 IPv6 코드로 변환하는 규칙

- (필요시) 새로운 헤더 파일을 포함
- 소켓 생성 시 AF\_INET 대신 AF\_INET6를 사용
- 소켓 주소 구조체로 sockaddr\_in 대신 sockaddr\_in6를 사용
- 데이터 전송 함수는 기존의 send() 함수와 recv() 함수를 변경 없이 그대로 사용

※ 이런 규칙을 따라 변경한 코드는 IPv6만을 지원



# TCP 서버-클라이언트(IPv6) (2)

## ■ 실습 4-3 TCP 서버-클라이언트(IPv6) 작성과 테스트

- TCPServer6.cpp
  - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter04/TCPServer6/TCPServer6.cpp>
  - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter04/TCPServer6.cpp>
- TCPClient6.cpp
  - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter04/TCPClient6/TCPClient6.cpp>
  - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter04/TCPClient6.cpp>