

Global KPU!

글로벌 경쟁력을 갖춘 산업기술 명문대학
세계를 향해 더 큰 미래를 펼쳐갑니다

4 TCP서버 – 클라이언트

네트워크 게임 프로그래밍

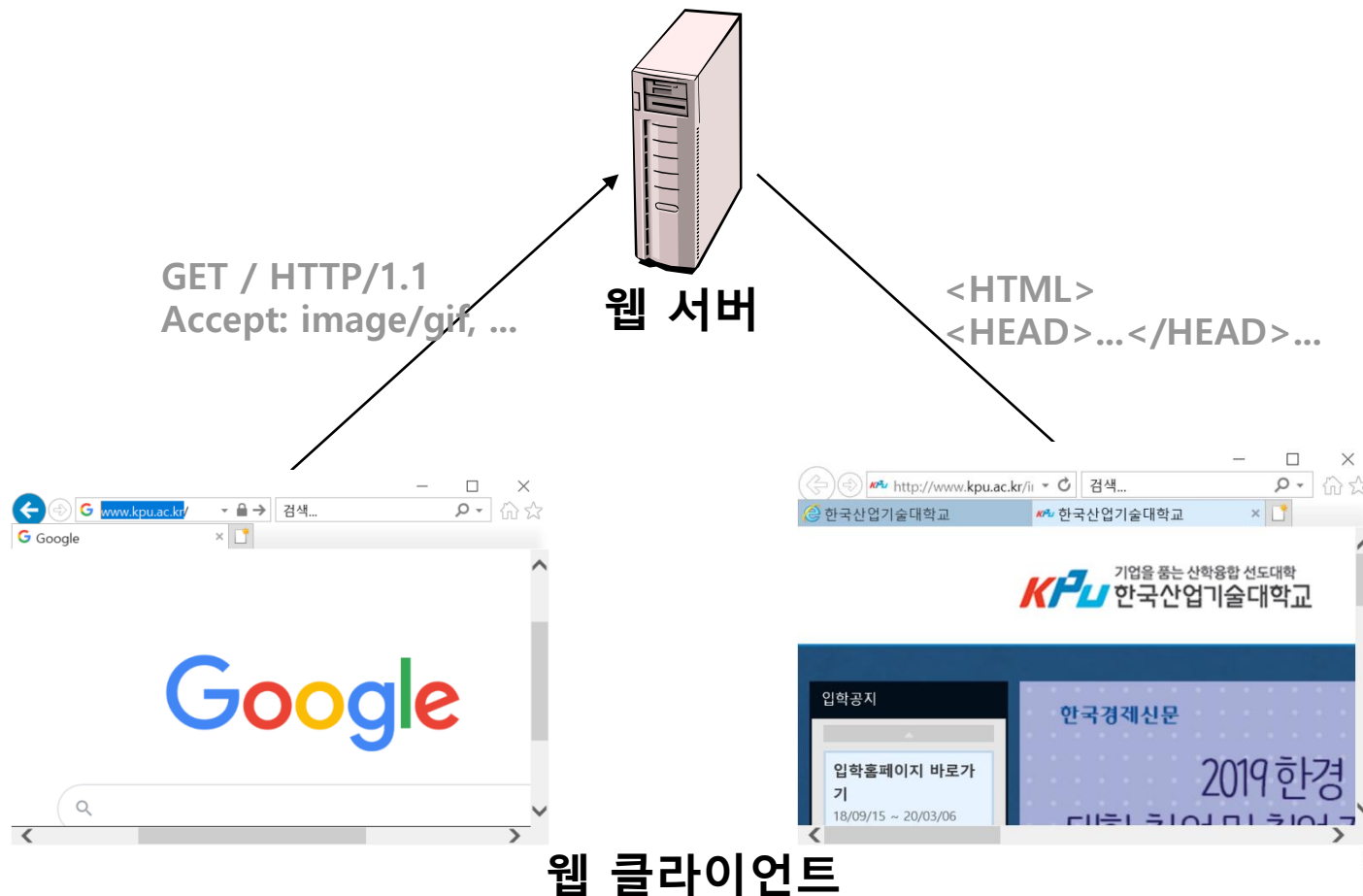
- ❖ **TCP 서버-클라이언트**의 기본 구조와 동작 원리를 이해한다.
- ❖ TCP 응용 프로그램 작성에 필요한 핵심 소켓 함수를 익힌다.
- ❖ IPv4와 IPv6 기반 TCP 서버-클라이언트를 작성할 수 있다.



TCP 서버-클라이언트 개념 (1)

❖ TCP 서버-클라이언트 동작

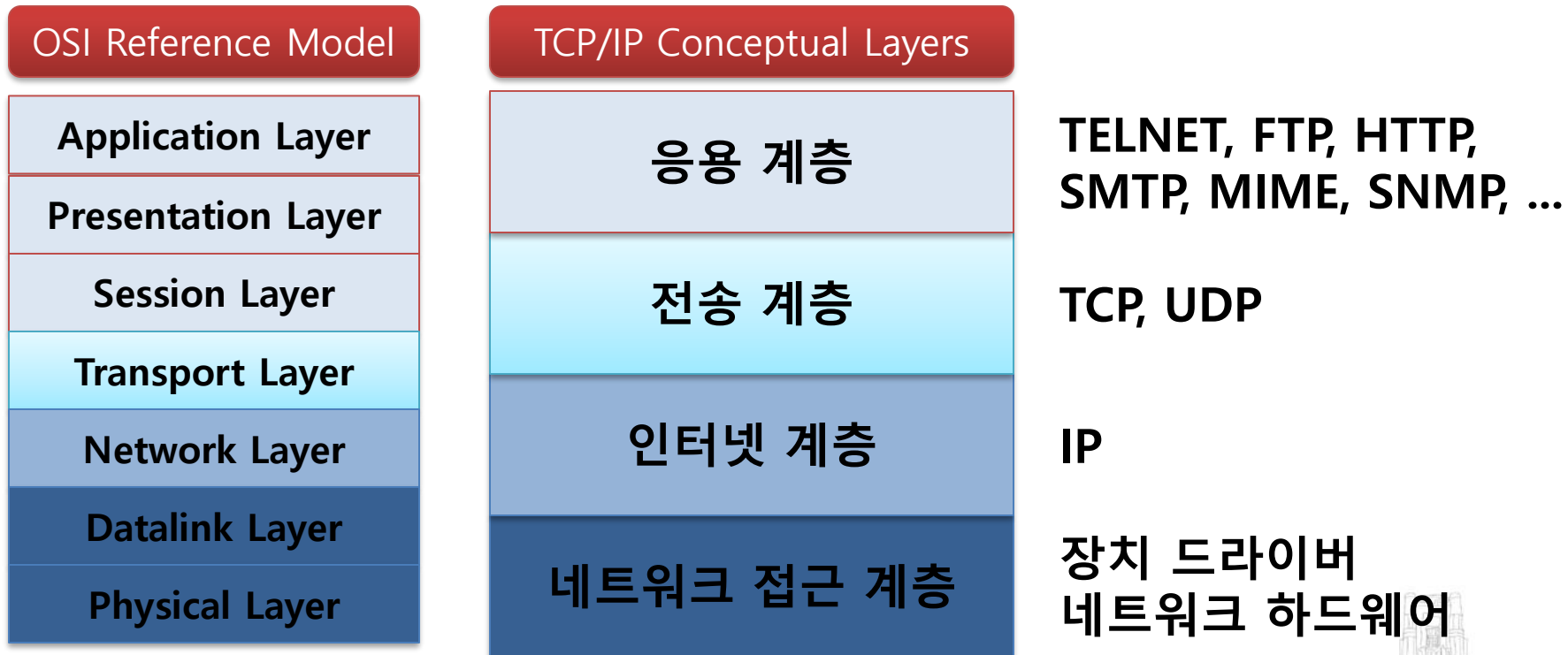
- HTTP는 TCP에 기반한 프로토콜이므로 웹서버-클라이언트는 대표적인 TCP 서버-클라이언트 응용 프로그램이라 할수 있다.



TCP 서버-클라이언트 개념 (1)

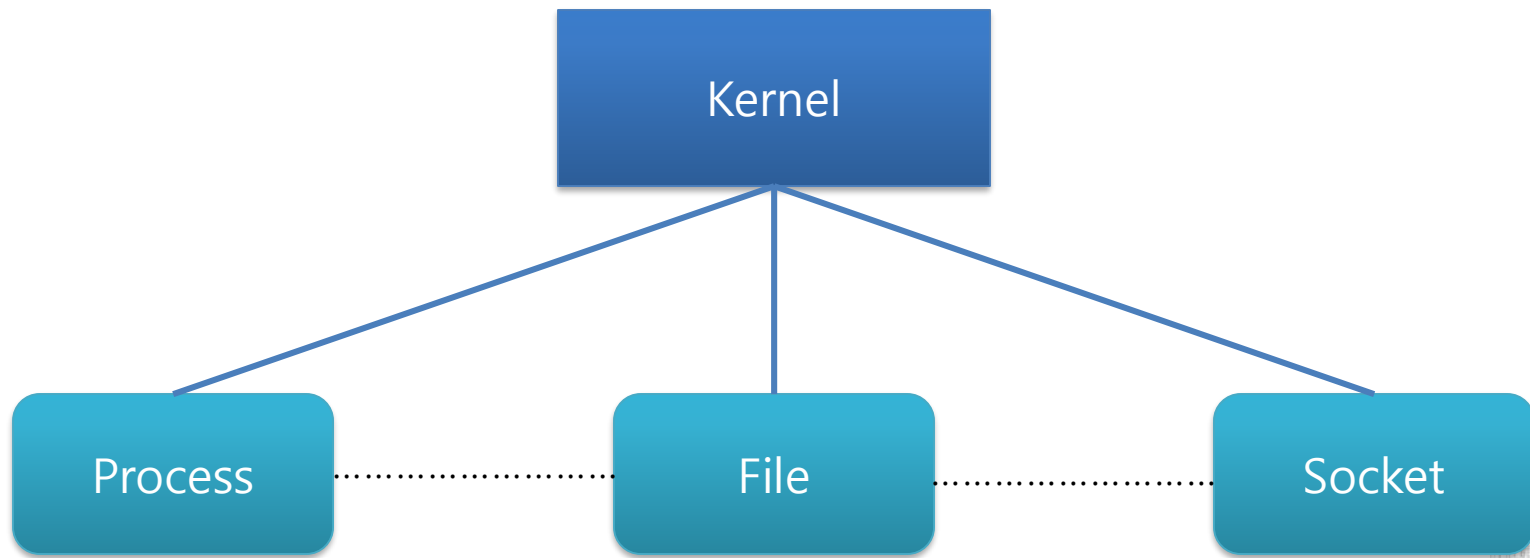
❖ TCP/IP 프로토콜 구조

- 계층적 구조



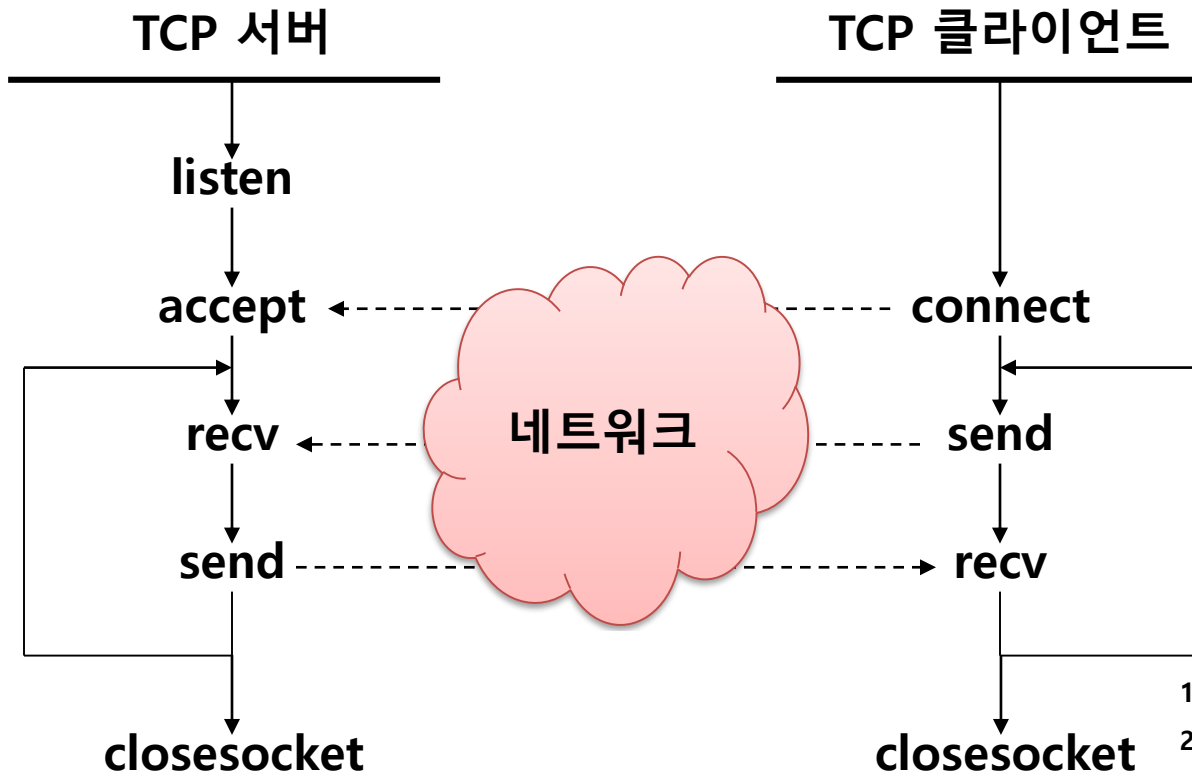
❖ WinSock

- 유닉스/리눅스는 소켓을 파일로 구성
- Windows는 소켓을 객체로 인식



TCP 서버-클라이언트 개념 (2)

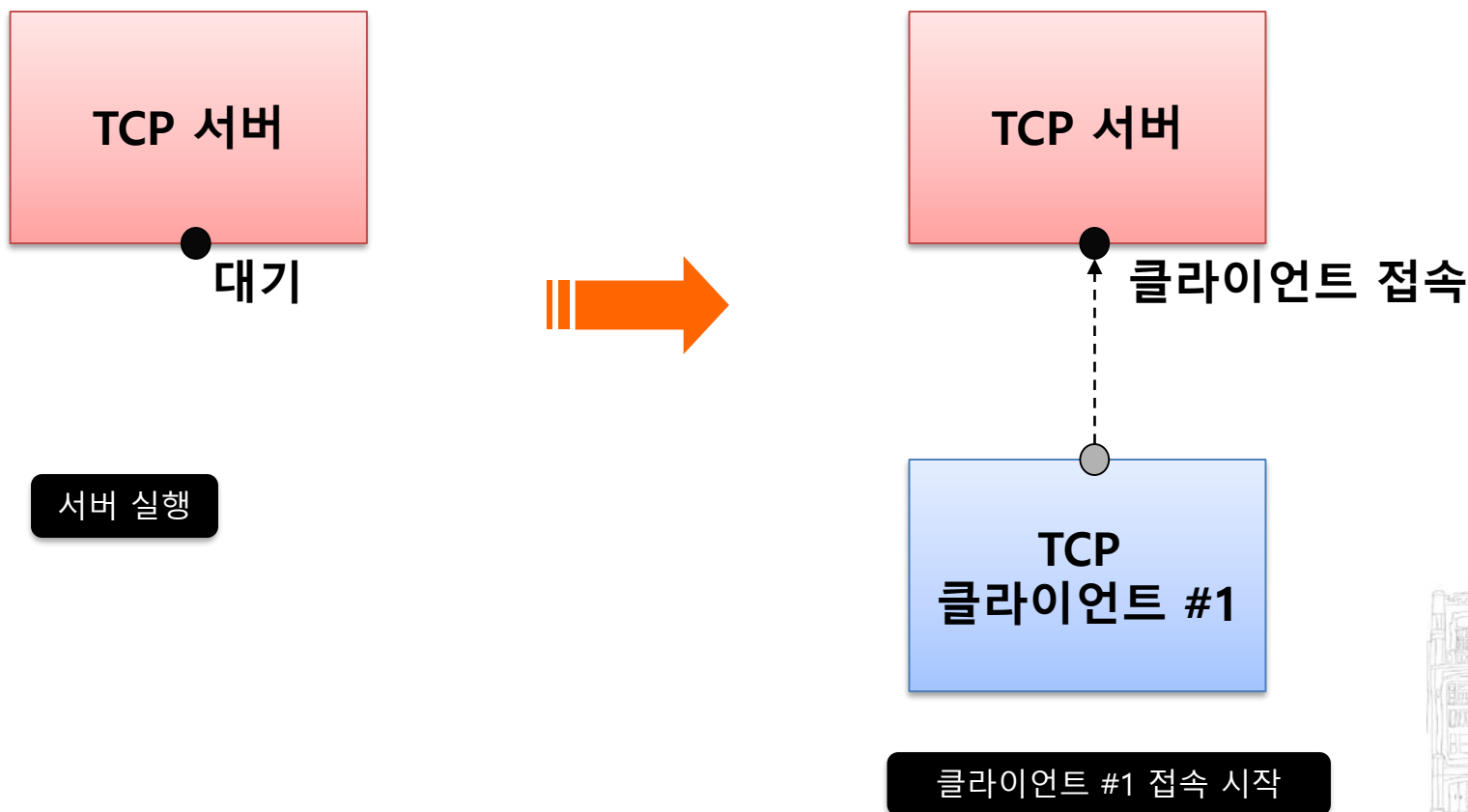
❖ TCP 서버-클라이언트 핵심 동작



1. 서버는 클라이언트가 접속하기를 대기(listen)
2. 클라이언트는 서버에 접속(Connect)
3. 클라이언트는 데이터를 서버로 전송(send)
4. 서버는 클라이언트 접속을 수용(accept)
5. 클라이언트가 보낸 데이터를 수신(recv) 후 처리
6. 서버는 처리한 데이터를 클라이언트에 전송(send)
7. 클라이언트는 서버가 보낸 데이터를 수신(recv)
8. 접속 종료(closesocket)

❖ TCP 서버-클라이언트 동작 원리

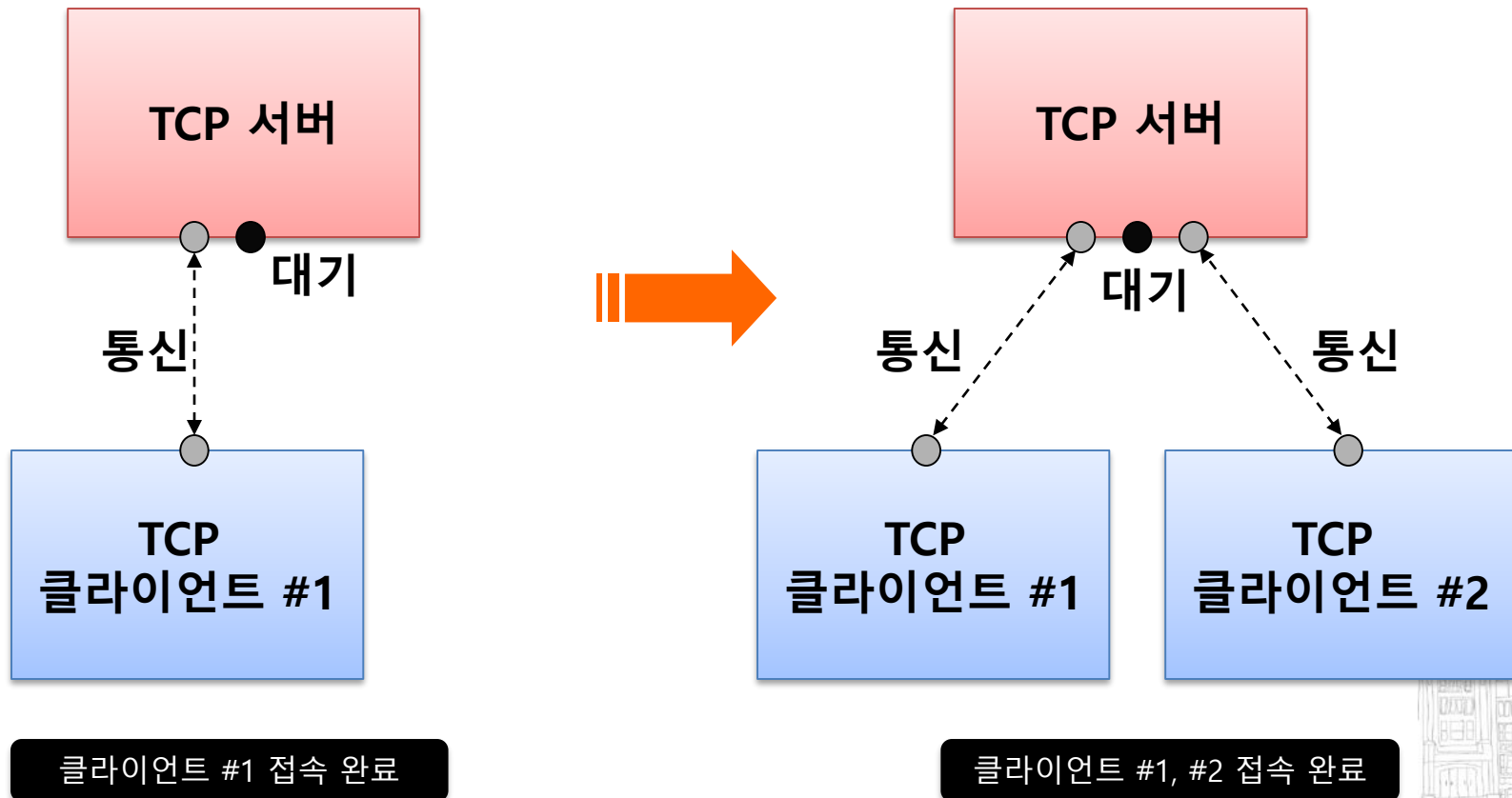
- 서버는 소켓을 생성한 후 클라이언트가 접속하기를 대기 (특정 포트 번호로 접속 가능)
- 클라이언트가 서버에 접속. TCP 프로토콜 수준에서 연결 설정을 위한 패킷 교환이 일어남



TCP 서버-클라이언트 동작 원리 (2)

❖ TCP 서버-클라이언트 동작 원리

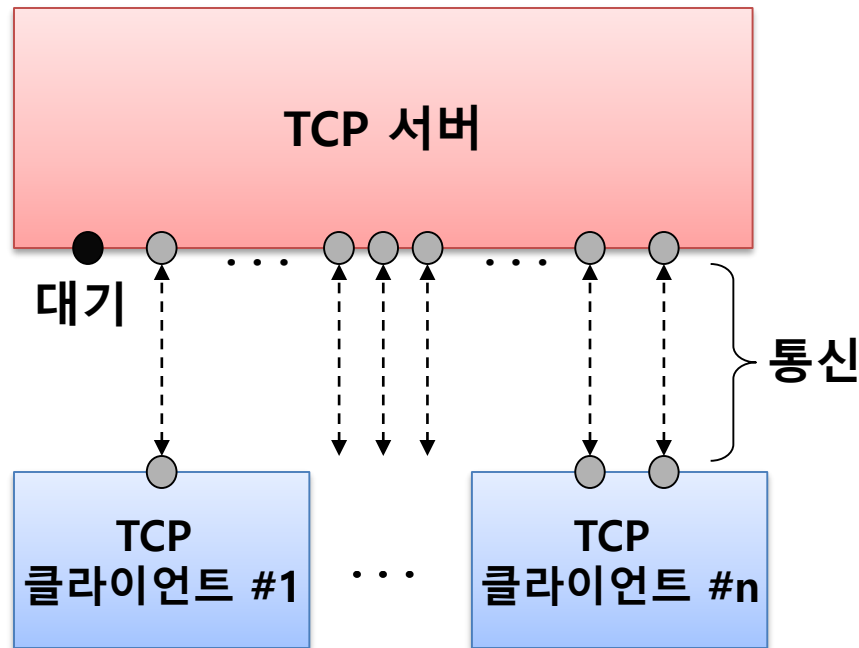
- TCP 연결 절차가 끝나면 새로운 소켓 생성. 기존 소켓은 새로운 클라이언트 접속을 수용하는 용도로 계속 사용
- 서버에는 소켓이 총 세 개 존재하며, 이 중 두 소켓을 접속한 클라이언트와 통신하는 용도로 사용



TCP 서버-클라이언트 동작 원리 (3)

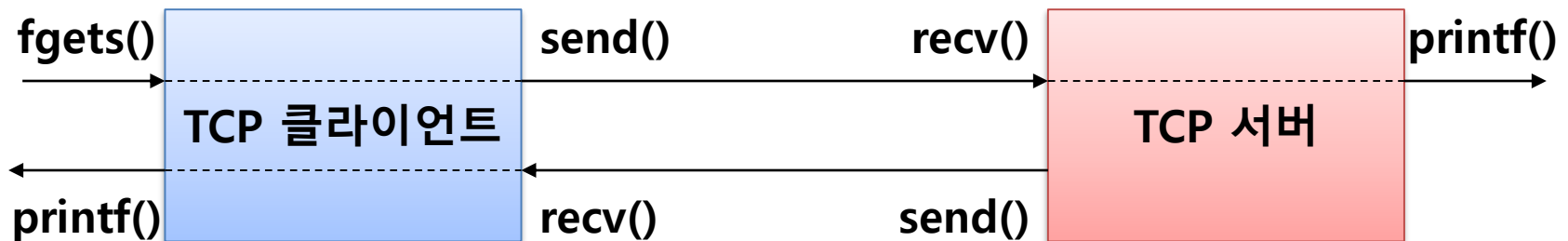
❖ TCP 서버 하나와 여러 TCP 클라이언트의 통신

- 서버측 소켓과 클라이언트 측 소켓이 일대일로 대응
- 클라이언트 한 개가 소켓을 두개 이상 사용해 서버에 접속 가능



❖ TCP 서버-클라이언트 예제 동작

- **서버:** 클라이언트가 보낸 데이터를 받아서(recv) 이를 문자열로 간주해 무조건 화면에 출력.printf) 후, 받은 데이터를 변경 없이 다시 클라이언트에 보냄(send)
- **클라이언트:** 사용자가 입력한 문자열(fgets)을 서버에 전송(send) 후, 서버가 받은 데이터를 그대로 송신하고, 클라이언트는 이를 받아서(recv) 화면에 출력.printf)



실습 4-1 TCP 서버 - 클라이언트 p94~

```
C:\Windows\system32\cmd.exe

[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=51188
[TCP/127.0.0.1:51188] 한국산업기술대학교
[TCP/127.0.0.1:51188] 게임공학과
[recv<>] 현재 연결은 원격 호스트에 의해 강제로 끊겼습니다.
[TCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=51188
```

```
C:\Windows\system32\cmd.exe

[보낼 데이터] 한국산업기술대학교
[TCP 클라이언트] 18바이트를 보냈습니다.
[TCP 클라이언트] 18바이트를 받았습니다.
[받은 데이터] 한국산업기술대학교

[보낼 데이터] 게임공학과
[TCP 클라이언트] 10바이트를 보냈습니다.
[TCP 클라이언트] 10바이트를 받았습니다.
[받은 데이터] 게임공학과

[보낼 데이터] _
```



netstat -na

The image contains two screenshots of a Windows command prompt window, titled "C:\Windows\system32\cmd.exe".

The top screenshot displays the output of the `netstat` command, showing network connections. The output is as follows:

Protocol	Local Address	Foreign Address	State
TCP	0.0.0.0:61616	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5354	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5354	127.0.0.1:49155	ESTABLISHED
TCP	127.0.0.1:5432	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5432	127.0.0.1:51201	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51202	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51203	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51204	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51205	ESTABLISHED
TCP	127.0.0.1:5939	0.0.0.0:0	LISTENING
TCP	127.0.0.1:9000	127.0.0.1:51188	ESTABLISHED
TCP	127.0.0.1:27015	0.0.0.0:0	LISTENING
TCP	127.0.0.1:27015	127.0.0.1:50198	ESTABLISHED
TCP	127.0.0.1:31000	127.0.0.1:32000	ESTABLISHED
TCP	127.0.0.1:32000	0.0.0.0:0	LISTENING
TCP	127.0.0.1:32000	127.0.0.1:31000	ESTABLISHED
TCP	127.0.0.1:49155	127.0.0.1:5354	ESTABLISHED
TCP	127.0.0.1:50198	127.0.0.1:27015	ESTABLISHED
TCP	127.0.0.1:51188	127.0.0.1:9000	ESTABLISHED
TCP	127.0.0.1:51201	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51202	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51203	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51204	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51205	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51214	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51216	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51217	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51218	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51219	127.0.0.1:9000	TIME WAIT
TCP	127.0.0.1:51220	127.0.0.1:5432	ESTABLISHED

The bottom screenshot shows the output of the `netstat` command, showing network connections. The output is as follows:

Protocol	Local Address	Foreign Address	State
TCP	0.0.0.0:61616	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5354	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5354	127.0.0.1:49155	ESTABLISHED
TCP	127.0.0.1:5432	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5432	127.0.0.1:51201	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51202	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51203	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51204	ESTABLISHED
TCP	127.0.0.1:5432	127.0.0.1:51205	ESTABLISHED
TCP	127.0.0.1:5939	0.0.0.0:0	LISTENING
TCP	127.0.0.1:9000	127.0.0.1:51188	ESTABLISHED
TCP	127.0.0.1:27015	0.0.0.0:0	LISTENING
TCP	127.0.0.1:27015	127.0.0.1:50198	ESTABLISHED
TCP	127.0.0.1:31000	127.0.0.1:32000	ESTABLISHED
TCP	127.0.0.1:32000	0.0.0.0:0	LISTENING
TCP	127.0.0.1:32000	127.0.0.1:31000	ESTABLISHED
TCP	127.0.0.1:49155	127.0.0.1:5354	ESTABLISHED
TCP	127.0.0.1:50198	127.0.0.1:27015	ESTABLISHED
TCP	127.0.0.1:51188	127.0.0.1:9000	ESTABLISHED
TCP	127.0.0.1:51201	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51202	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51203	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51204	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51205	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51214	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51216	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51217	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51218	127.0.0.1:5432	ESTABLISHED
TCP	127.0.0.1:51219	127.0.0.1:9000	TIME WAIT
TCP	127.0.0.1:51220	127.0.0.1:5432	ESTABLISHED



❖ 소켓 통신을 위해 결정해야 할 요소

- ① 프로토콜
 - 통신 규약. 소켓을 생성할 때 결정
- ② 지역 IP 주소와 지역 포트 번호
 - 서버 또는 클라이언트 자신의 주소
- ③ 원격 IP 주소와 원격 포트 번호
 - 서버 또는 클라이언트가 통신하는 상대의 주소

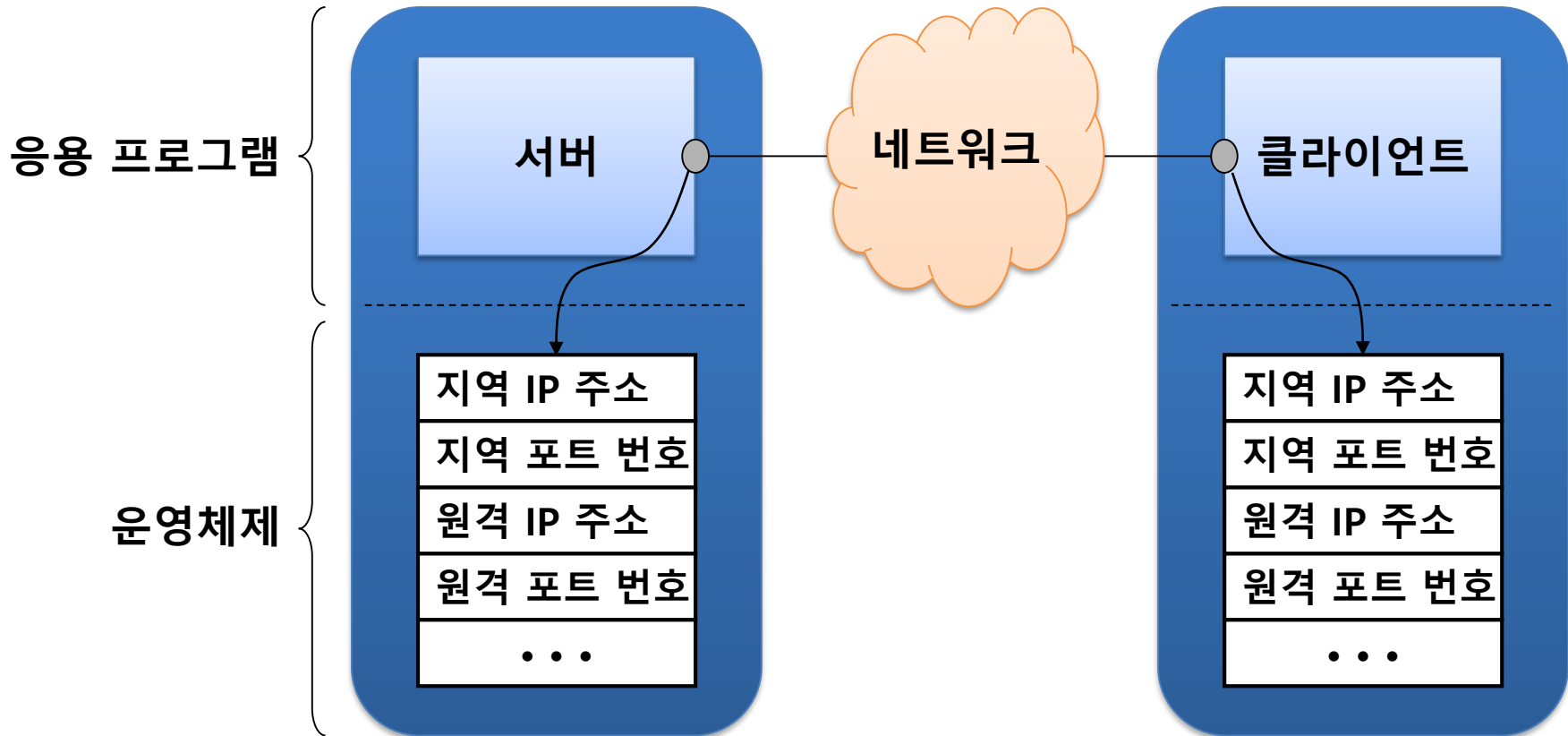
❖ 쉽게,

- ① 통신 프로토콜 (TCP/UDP등)
- ② 클라이언트 IP 주소 및 포트번호 (운영체제에서 할당)
- ③ 서버 IP 주소 및 포트번호



TCP 서버-클라이언트 분석 (2)

❖ 소켓 데이터 구조체

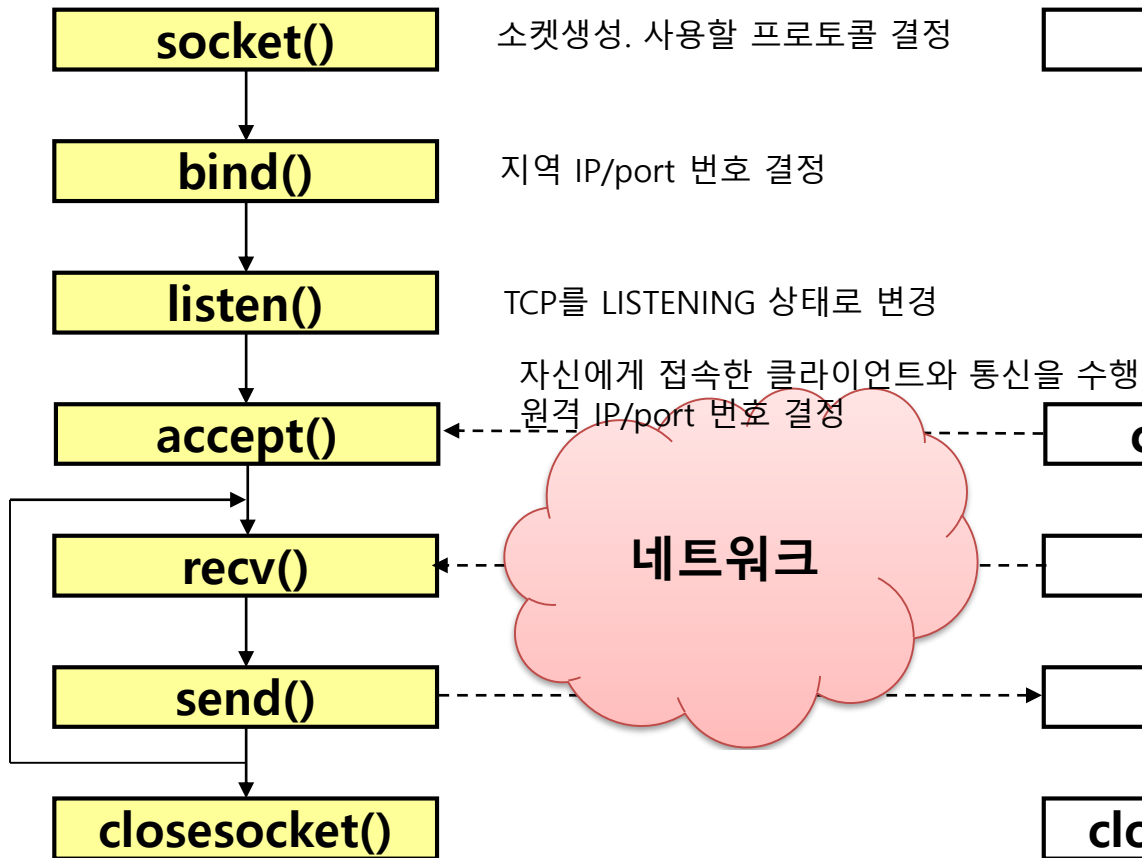


- 소켓 함수는 지역 주소와 원격 주소를 결정하고 TCP 상태를 변경하기 위한 일련의 절차
- 예제에서는 서버 함수, 클라이언트 함수, 데이터 전송 함수로 구분
- 서버 함수는 서버에서만, 클라이언트 함수는 클라이언트에서만, 데이터 전송 함수는 양 쪽에서 사용

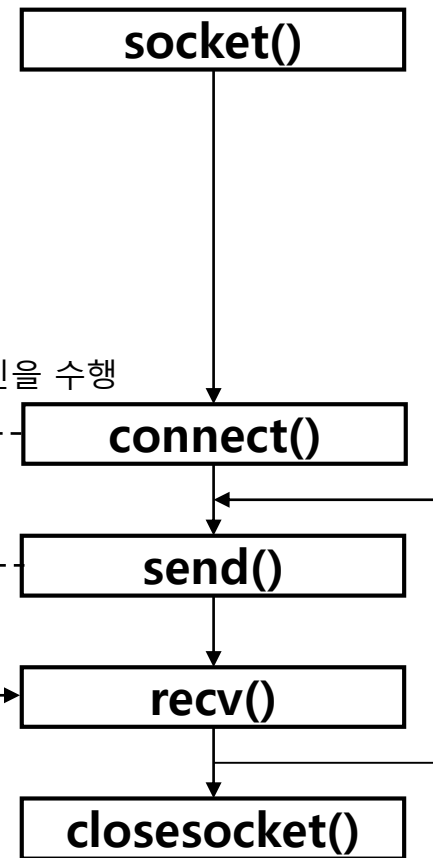


❖ TCP 서버 함수

TCP 서버



TCP 클라이언트



- send(),recv() : 데이터 전송함수로 클라이언트와 통신을 수행한 후, closesocket() 함수로 소켓을 닫음
- 유닉스에서는 read()/write() 사용 가능



❖ bind() 함수

- 소켓의 지역 IP 주소와 지역 포트 번호를 결정
- 직접적인 소켓 연결

```
int bind (  
    SOCKET s,  
    const struct sockaddr *name,  
    int namelen  
);
```

성공: 0, 실패: SOCKET_ERROR

- s: 클라이언트 접속을 수용할 목적으로 만든 소켓
- name: 소켓 주소 구조체를 지역 IP 주소와 지역 포트 번호로 초기화하여 전달
- namelen: 소켓 주소 구조체의 길이



❖ bind() 함수 사용 예

```
050  SOCKADDR_IN serveraddr;  
051  ZeroMemory(&serveraddr, sizeof(serveraddr));  
052  serveraddr.sin_family = AF_INET;  
053  serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);  
054  serveraddr.sin_port = htons(SERVERPORT);  
055  retval = bind(listen_sock, (SOCKADDR *)&serveraddr, sizeof(serveraddr));  
056  if(retval == SOCKET_ERROR) err_quit("bind()");
```

- 50-51행: 소켓 주소 구조체 변수를 선언하고 0으로 초기화
- 52행: 인터넷 주소 체계를 사용. AF_INET(IPv4)
- 53행: 서버의 IP 주소 설정
 - INADDR_ANY는 모든 주소를 기다림을 의미
- 54행: 서버의 지역 포트 번호 설정
- 55-56행: bind() 함수를 호출하고 오류를 처리



❖ listen() 함수

- 소켓의 TCP 포트 상태를 LISTENING으로 변경
- 수신 대기열 생성
- accept()를 통해 연결

```
int listen (  
    SOCKET s,  
    int backlog  
);
```

성공: 0, 실패: SOCKET_ERROR

- s: 클라이언트 접속을 수용할 목적으로 만든 소켓(연결소켓)
- backlog: 서버가 당장 처리하지 않더라도 접속 가능한 클라이언트 개수
- 전화를 기다리는 과정(함수)
 - bind()/listen()/accept()



❖ listen() 함수 사용 예

```
059  retval = listen(listen_sock, SOMAXCONN);  
060  if(retval == SOCKET_ERROR) err_quit("listen()");
```

- 59-60행: backlog 를 최대값으로 하여 listen() 함수를 호출하고 오류를 처리



❖ accept() 함수

- 접속한 클라이언트와 통신할 수 있도록 새로운 소켓을 생성해서 리턴 (연결 접수)
- 접속한 클라이언트의 IP 주소와 포트 번호를 알려줌

```
SOCKET accept (  
    SOCKET s,  
    struct sockaddr *addr,  
    int *addrlen  
);
```

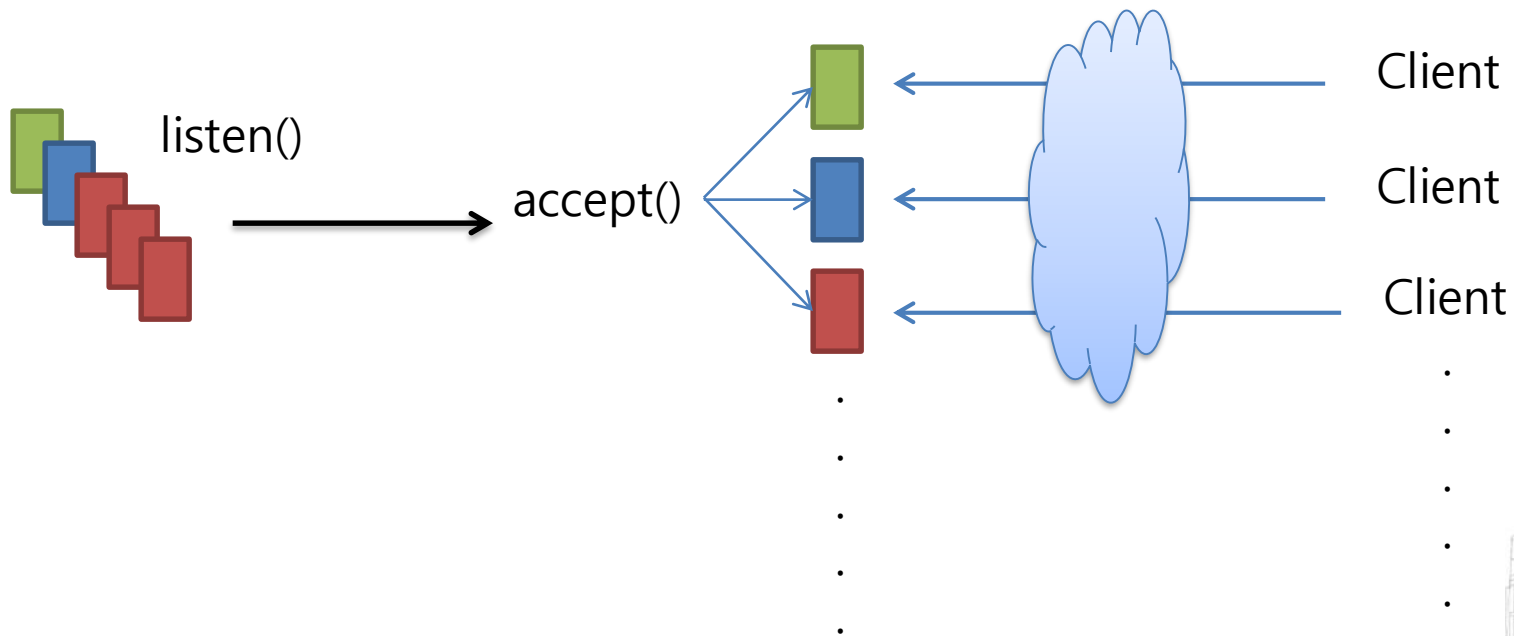
성공: 새로운 소켓, 실패: INVALID_SOCKET

- s: 클라이언트 접속을 수용할 목적으로 만든 소켓
- addr: 소켓 주소 구조체를 전달하면 접속한 클라이언트의 주소 정보로 채워짐
- addrlen: 정수형 변수를 addr이 가르키는 소켓 주소 구조체의 크기로 초기화한 후 전달



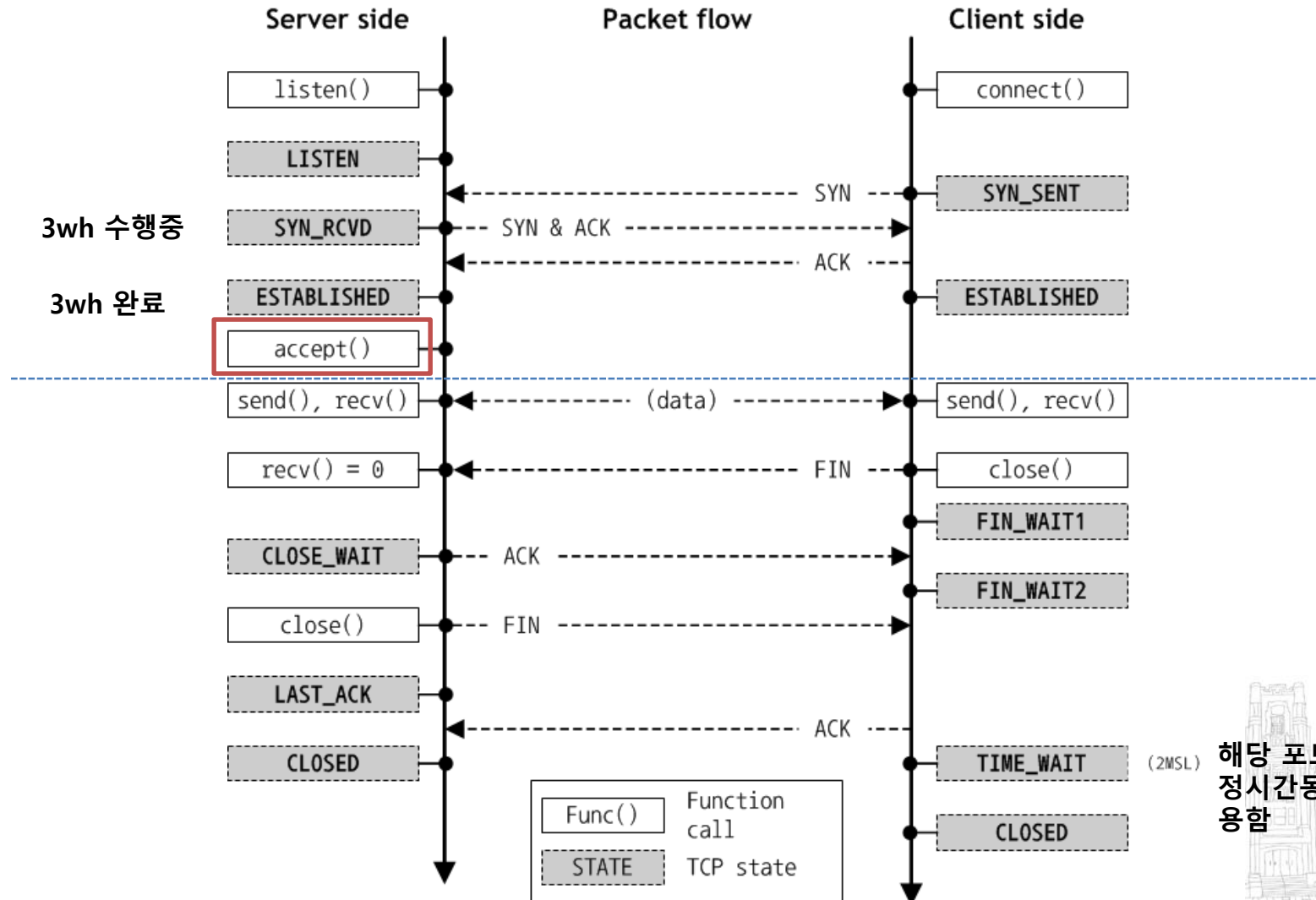
❖ accept() 함수

- 접속한 클라이언트와 통신할 수 있도록 새로운 소켓을 생성해서 리턴
- 접속한 클라이언트의 IP 주소와 포트 번호를 알려줌



TCP 서버 함수 (6)

❖ accept() 함수



해당 포트를 일
정시간동안 사
용함

❖ accept() 함수 사용 예

```
063 SOCKET client_sock;
064 SOCKADDR_IN clientaddr;
065 int addrlen;
...
068 while(1){
069     // accept()
070     addrlen = sizeof(clientaddr);
071     client_sock = accept(listen_sock, (SOCKADDR *)&clientaddr, &addrlen);
072     if(client_sock == INVALID_SOCKET){
073         err_display("accept()");
074         break;
075     }
076
077     // 접속한 클라이언트 정보 출력
078     printf("\n[TCP 서버] 클라이언트 접속: IP 주소=%s, 포트 번호=%d\n",
079         inet_ntoa(clientaddr.sin_addr), ntohs(clientaddr.sin_port));
```



❖ accept() 함수 사용 예

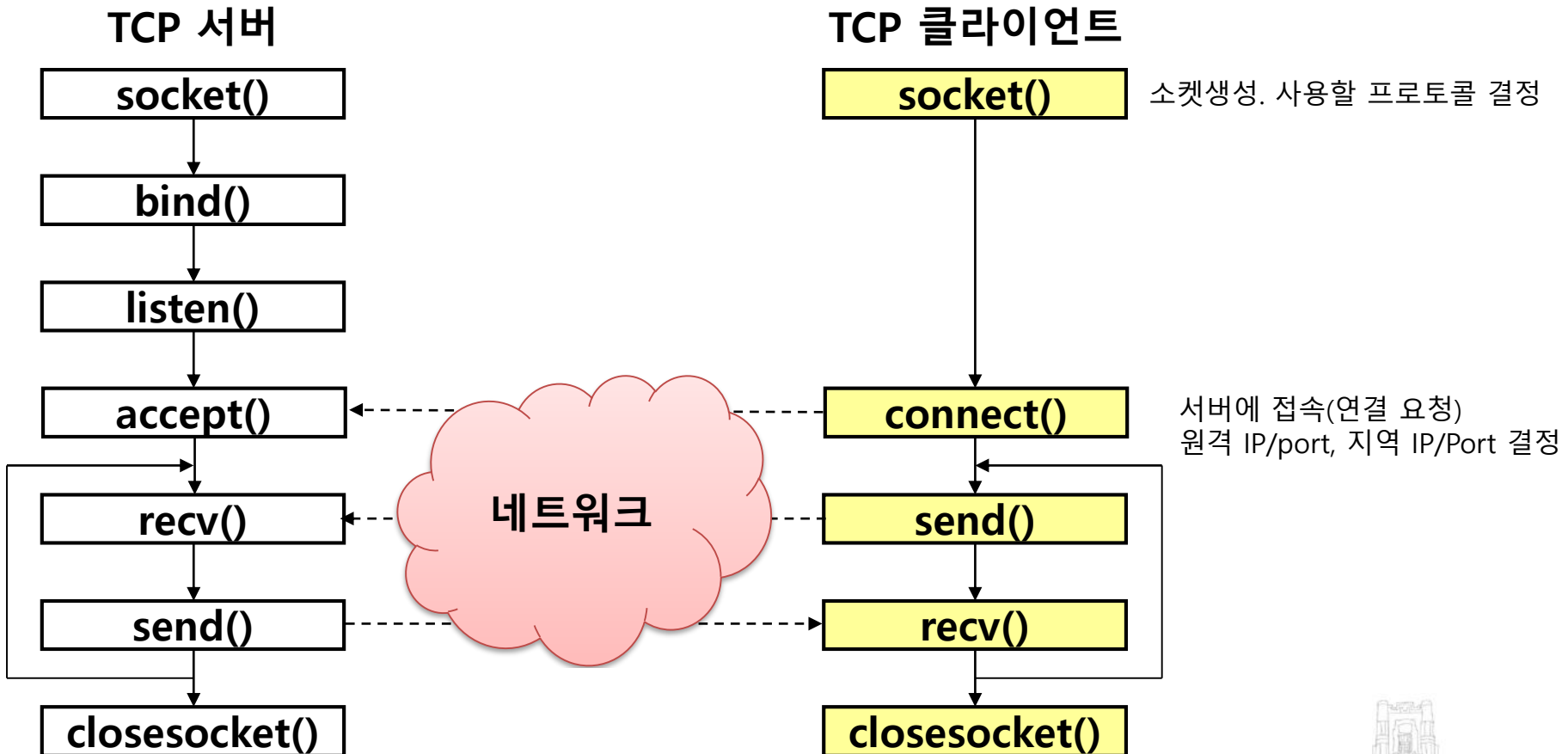
```
080
081 // 클라이언트와 데이터 통신
082 while(1){
...
103 }
104
105 // closesocket()
106 closesocket(client_sock);
107 printf("[TCP 서버] 클라이언트 종료: IP 주소=%s, 포트 번호=%d\n",
108        inet_ntoa(clientaddr.sin_addr), ntohs(clientaddr.sin_port));
109 }
```

- 82-103행: accept() 함수가 리턴한 소켓을 이용해 클라이언트와 통신
- 106-107행: 클라이언트와 통신을 마치면 소켓을 닫고, 접속을 종료한 클라이언트 IP 주소와 포트 번호를 화면에 출력



TCP 클라이언트 함수 (1)

❖ TCP 클라이언트 함수



- send(),recv() : 데이터 전송함수로 클라이언트와 통신을 수행한 후, closesocket() 함수로 소켓을 닫음



❖ connect() 함수

- TCP 프로토콜 수준에서 서버와 논리적 연결을 설정

```
int connect (  
    SOCKET s,  
    const struct sockaddr *name,  
    int namelen  
);
```

성공: 0, 실패: SOCKET_ERROR

- s: 서버와 통신할 목적으로 만든 소켓
- name: 소켓 주소 구조체를 서버 주소로 초기화하여 전달
 - 다양한 형태의 소켓 유형이 있으므로 값들이 이상함
- namelen: (name의) 소켓 주소 구조체의 길이



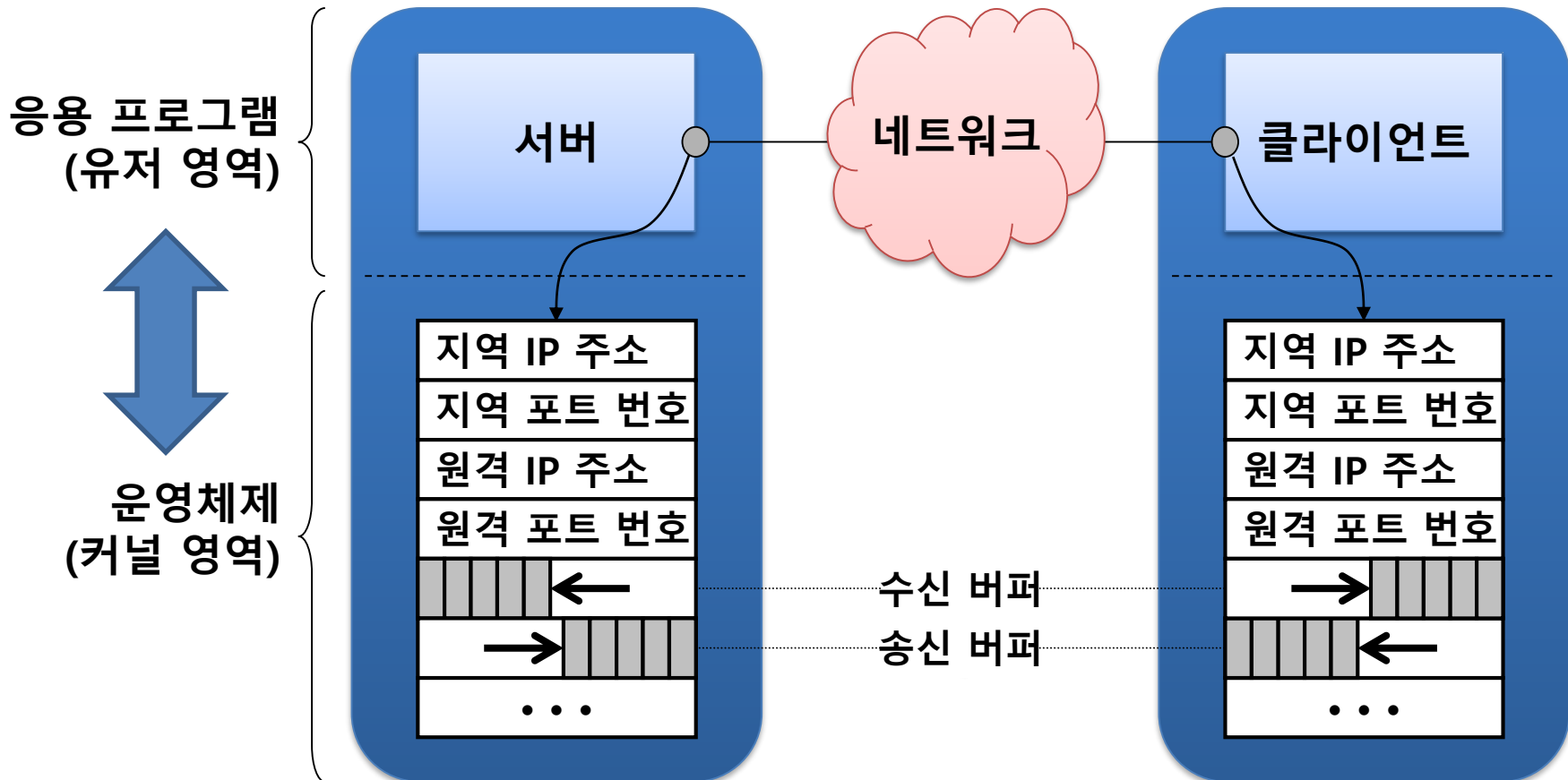
❖ connect() 함수 사용 예

```
071  SOCKADDR_IN serveraddr;  
072  ZeroMemory(&serveraddr, sizeof(serveraddr));  
073  serveraddr.sin_family = AF_INET;  
074  serveraddr.sin_addr.s_addr = inet_addr(SERVERIP);  
075  serveraddr.sin_port = htons(SERVERPORT);  
076  retval = connect(sock, (SOCKADDR *)&serveraddr, sizeof(serveraddr));  
077  if(retval == SOCKET_ERROR) err_quit("connect()");
```

- 71-75행: 소켓 주소 구조체 변수를 0으로 초기화하고, IP 주소와 포트 번호를 대입
- 76-77행: connect() 함수를 호출하고 오류를 처리
 - 비어있는 임의의 포트를 할당하고 연결 시도



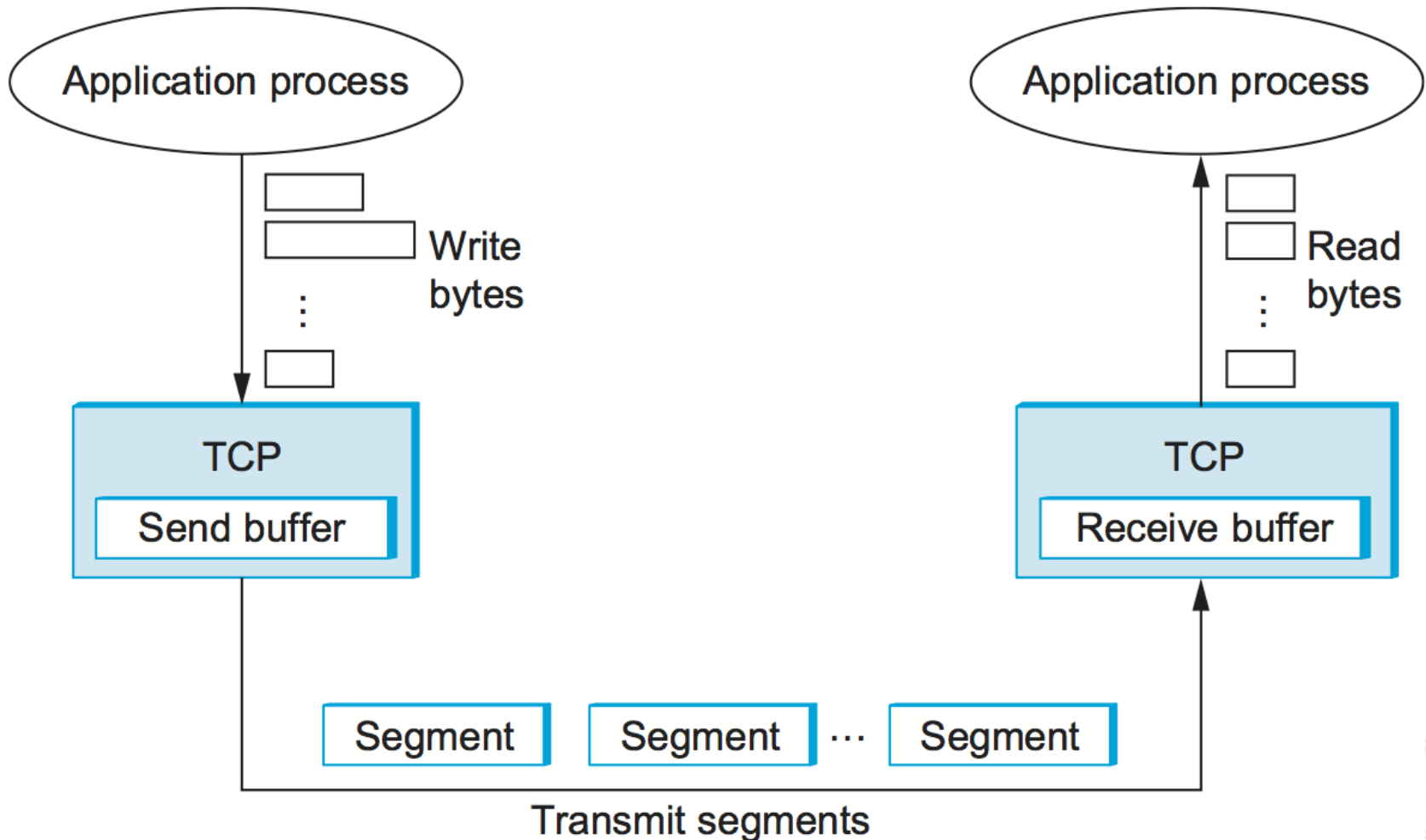
❖ 소켓 데이터 구조체



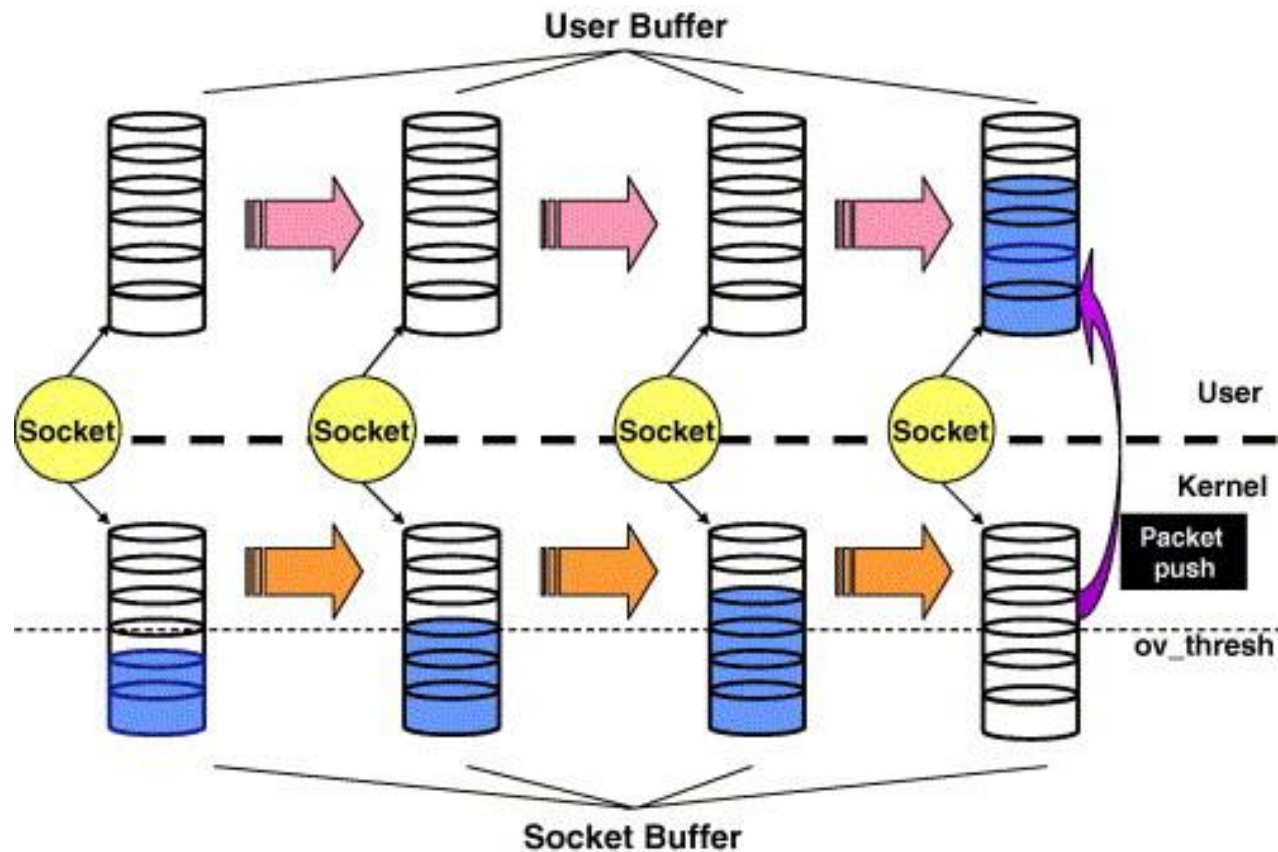
- 데이터 전송함수는 데이터를 보내는 함수와 데이터를 받는 함수로 구분
- `send()` 와 `recv()` 외에 `sendto()/recvfrom()` 과 `WSASend*()/WSARecv*()` 형태의 확장 함수가 존재. (윈도우는 소켓 버퍼를 따로 두지 않고 바로 운영체제의 버퍼로 전송하는 기능 지원)

❖ read() / write() – Unix

- Block / non-Block



❖ User buffer / Socket Buffer (UNIX)



❖ send() 함수

- 응용 프로그램 데이터를 운영체제의 송신 버퍼에 복사함으로써 데이터를 전송

```
int send(  
    SOCKET s,  
    const char *buf,  
    int len,  
    int flags  
);
```

성공: 보낸 바이트 수, 실패: SOCKET_ERROR

- s: 통신할 대상과 연결된 소켓 (소켓 지정번호)
- buf: 보낼 데이터를 담고 있는 응용 프로그램의 버퍼 주소
- len: 보낼 데이터 크기
- flags: send() 함수의 동작을 바꾸는 옵션. 대부분 0을 사용
- 참고
 - 소켓 함수: send, recv, sendto, recvfrom
 - 파일 함수: read, write



❖ recv() 함수

- 운영체제의 수신 버퍼에 도착한 데이터를 응용 프로그램 버퍼에 복사

```
int recv (  
    SOCKET s,  
    char *buf,  
    int len,  
    int flags  
);
```

성공: 받은 바이트 수 또는 0(연결 종료시)
실패: SOCKET_ERROR

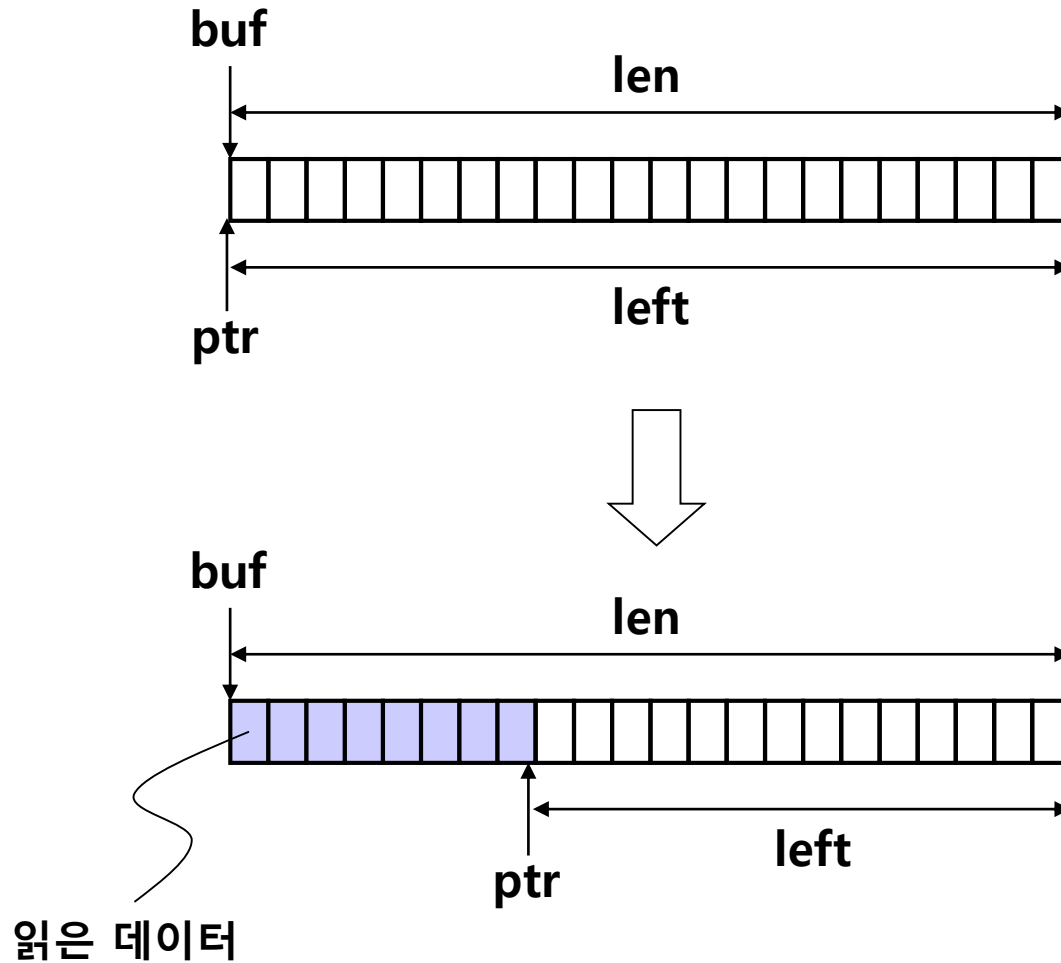
- s: 통신할 대상과 연결된 소켓
- buf: 받은 데이터를 저장할 응용 프로그램 버퍼 주소
- len: 운영체제의 수신 버퍼로부터 복사할 최대 데이터 크기
- flags: recv() 함수의 동작을 바꾸는 옵션. 대부분 0을 사용



❖ recvn() 함수 정의 : 사용자 정의 함수

```
038 int recvn(SOCKET s, char *buf, int len, int flags)
039 {
040     int received;
041     char *ptr = buf;
042     int left = len;
043
044     while(left > 0){
045         received = recv(s, ptr, left, flags);
046         if(received == SOCKET_ERROR)
047             return SOCKET_ERROR;
048         else if(received == 0)
049             break;
050         left -= received;
051         ptr += received;
052     }
053
054     return (len - left);
055 }
```

❖ recvn() 함수 동작 원리



❖ 데이터 전송 함수 사용 예 - TCP 클라이언트

```
079 // 데이터 통신에 사용할 변수
080 char buf[BUFSIZE+1];
081 int len;
082
083 // 서버와 데이터 통신
084 while(1){
085     // 데이터 입력
086     printf("\n[보낼 데이터] ");
087     if(fgets(buf, BUFSIZE+1, stdin) == NULL)
088         break;
089
090     // '\n' 문자 제거
091     len = strlen(buf);
092     if(buf[len-1] == '\n')
093         buf[len-1] = '\0';
094     if(strlen(buf) == 0)
095         break;
```



❖ 데이터 전송 함수 사용 예 - TCP 클라이언트

```
096
097 // 데이터 보내기
098 retval = send(sock, buf, strlen(buf), 0);
099 if(retval == SOCKET_ERROR){
100     err_display("send()");
101     break;
102 }
103 printf("[TCP 클라이언트] %d바이트를 보냈습니다.\n", retval);
104
105 // 데이터 받기
106 retval = recv(sock, buf, retval, 0);
107 if(retval == SOCKET_ERROR){
108     err_display("recv()");
109     break;
110 }
111 else if(retval == 0)
112     break;
```

❖ 데이터 전송 함수 사용 예 - TCP 클라이언트

```
113
114 // 받은 데이터 출력
115 buf[retval] = '\0';
116 printf("[TCP 클라이언트] %d바이트를 받았습니다.\n", retval);
117 printf("[받은 데이터] %s\n", buf);
118 }
```

- 80행: 보낼 데이터 또는 받은 데이터를 저장할 버퍼
- 81행: 사용자가 입력한 문자열 데이터의 길이를 계산할때 사용
- 86-88행: fgets() 함수를 사용해 사용자로부터 문자열을 입력 받음
- 91-93행: 'Wn' 문자를 제거
- 94-95행: 'Wn' 문자를 제거한 후 문자열 길이가 0이면 사용자가 글자를 입력하지 않고 곧바로 엔터키를 눌렀다는 의미. 이 경우 루프를 빠져나가고 closesocket() 함수를 호출해 접속을 정상 종료
- 98-103행: send() 함수를 호출하고 오류를 처리
- 106-112행: recvn() 함수를 호출하고 오류를 처리
- 115-117행: 받은 데이터 끝에 'W0'을 추가하여 화면에 출력



❖ 데이터 전송 함수 사용 예 - TCP 서버

```
066 char buf[BUFSIZE+1];
067
068 while(1){
...
081 // 클라이언트와 데이터 통신
082 while(1){
083     // 데이터 받기
084     retval = recv(client_sock, buf, BUFSIZE, 0);
085     if(retval == SOCKET_ERROR){
086         err_display("recv()");
087         break;
088     }
089     else if(retval == 0)
090         break;
091
```

- 66행: 받은 데이터를 저장할 응용 프로그램 버퍼
- 82행: recv() 함수의 리턴 값이 0(정상종료) 또는 SOCKET_ERROR(오류발생)가 될때까지 계속 루프를 돌며 데이터를 수신
- 84-90행: recv() 함수를 호출하고 오류를 처리



❖ 데이터 전송 함수 사용 예 - TCP 서버

```
092    // 받은 데이터 출력
093    buf[retval] = '\0';
094    printf("[TCP/%s:%d] %s\n", inet_ntoa(clientaddr.sin_addr),
095           ntohs(clientaddr.sin_port), buf);
096
097    // 데이터 보내기
098    retval = send(client_sock, buf, retval, 0);
099    if(retval == SOCKET_ERROR){
100        err_display("send()");
101        break;
102    }
103    } ← 안쪽 while 루프의 끝
...
109    } ← 바깥쪽 while 루프의 끝
```

- 93-95행: 받은 데이터 끝에 '\0'을 추가해 화면에 출력
- 96-102행: send() 함수를 호출하고 오류를 처리

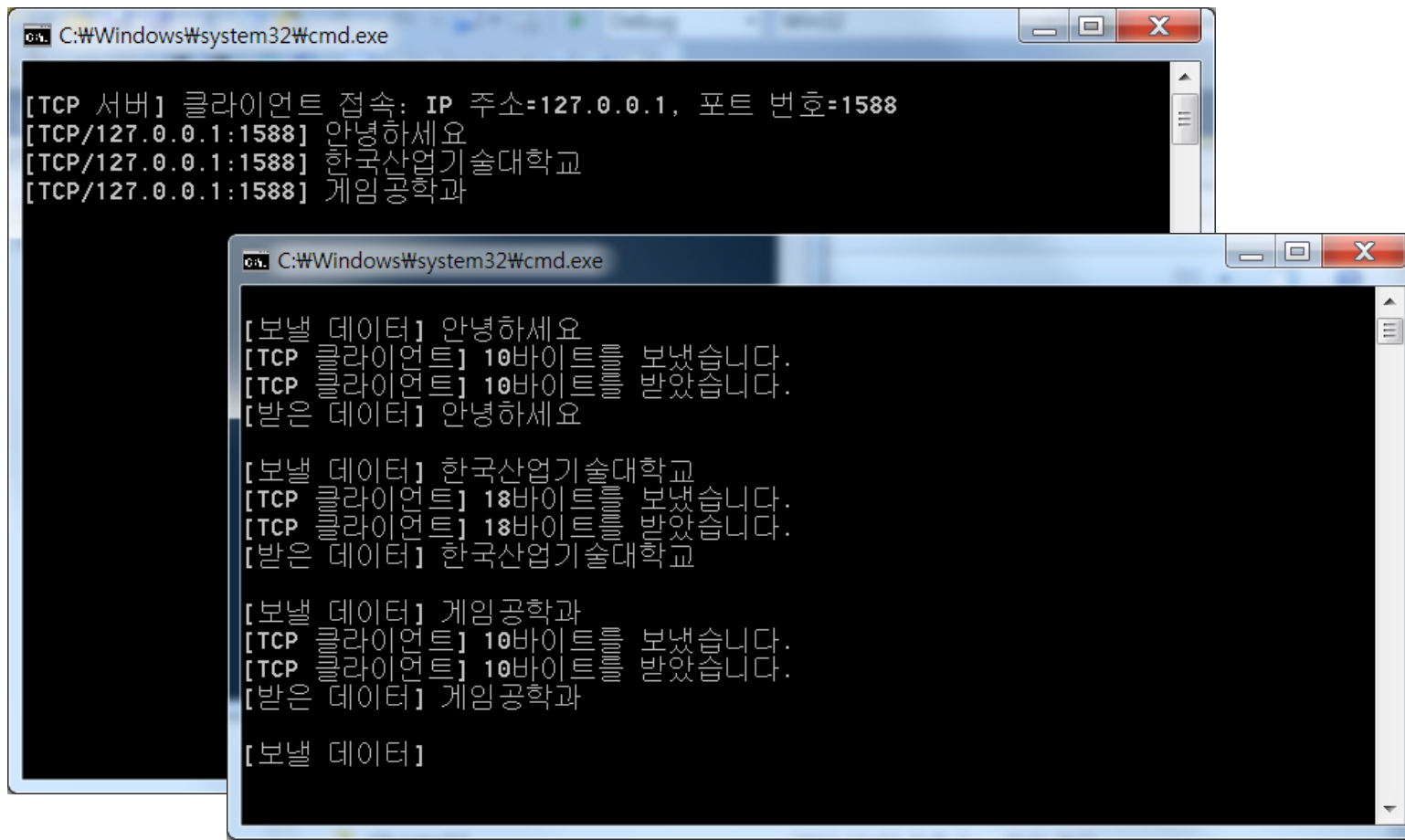


❖ IPv4 코드 ⇒ IPv6 코드

- ws2tcpip.h 헤더 파일을 포함
- 소켓 생성 시 AF_INET 대신 AF_INET6를 사용
- 소켓 주소 구조체로 SOCKADDR_IN 대신 SOCKADDR_IN6를 사용
 - 구조체를 변경하면 구조체 필드명도 그에 따라 변경
 - 서버에서 주로 사용하는 INADDR_ANY(0으로 정의됨) 값은 in6addr_any(0으로 정의됨)로 변경
- IPv4만을 지원하는 주소 변환 함수를 IPv4/IPv6 지원 함수로 대체
- 데이터 전송 함수는 기존의 send()/recv() 함수를 변경 없이 그대로 사용



실습 4-2 TCP 서버 – 클라이언트 예제 p121~



The image shows two overlapping Windows command prompt windows. The top window, titled 'C:\Windows\system32\cmd.exe', displays the following text:

```
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=1588  
[TCP/127.0.0.1:1588] 안녕하세요  
[TCP/127.0.0.1:1588] 한국산업기술대학교  
[TCP/127.0.0.1:1588] 게임공학과
```

The bottom window, also titled 'C:\Windows\system32\cmd.exe', displays the following text:

```
[보낼 데이터] 안녕하세요  
[TCP 클라이언트] 10바이트를 보냈습니다.  
[TCP 클라이언트] 10바이트를 받았습니다.  
[받은 데이터] 안녕하세요  
  
[보낼 데이터] 한국산업기술대학교  
[TCP 클라이언트] 18바이트를 보냈습니다.  
[TCP 클라이언트] 18바이트를 받았습니다.  
[받은 데이터] 한국산업기술대학교  
  
[보낼 데이터] 게임공학과  
[TCP 클라이언트] 10바이트를 보냈습니다.  
[TCP 클라이언트] 10바이트를 받았습니다.  
[받은 데이터] 게임공학과  
  
[보낼 데이터]
```



연습문제

클라이언트 모듈에 bind()를 이용하여 특정 포트로 접속하면?

```
// bind()
SOCKADDR_IN localaddr;
ZeroMemory(&localaddr, sizeof(localaddr));
localaddr.sin_family = AF_INET;
localaddr.sin_addr.s_addr = htonl(INADDR_ANY);
localaddr.sin_port = htons(50000);           // 특정 포트 설정
retval = bind(sock, (SOCKADDR *)&localaddr, sizeof(localaddr));
if(retval == SOCKET_ERROR) err_quit("bind()");
```





Thank You !

oasis01@gmail.com / rhqudtn75@nate.com