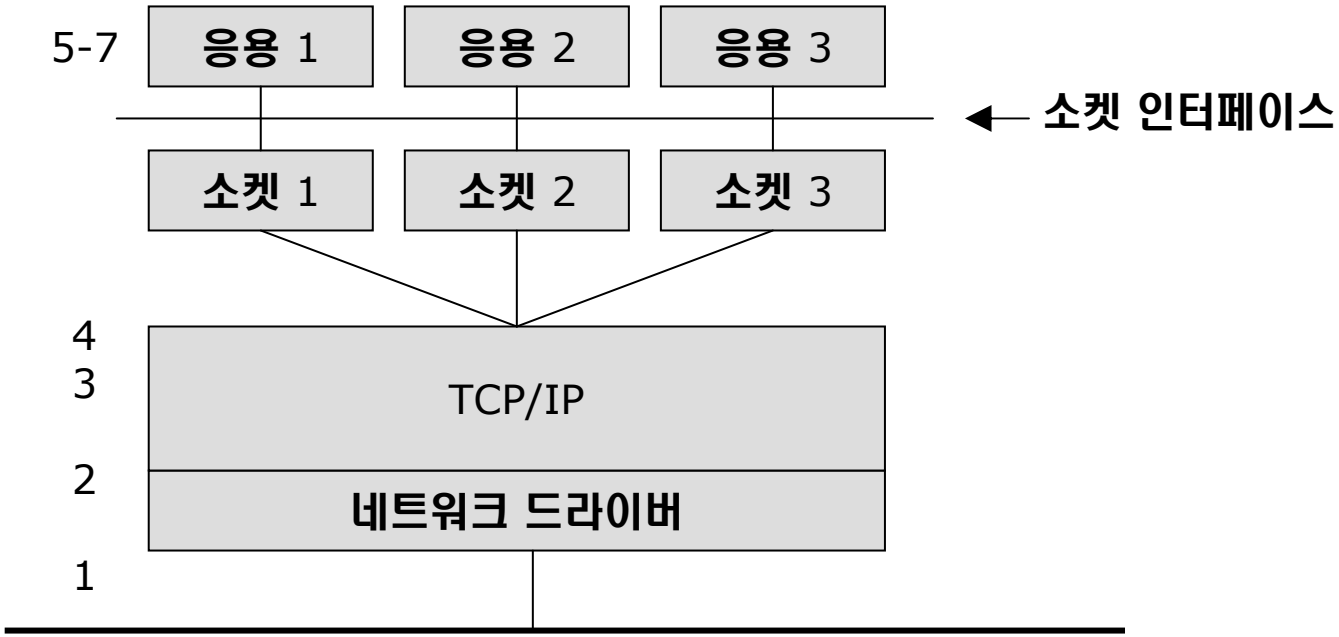
2. 소켓 프로그래밍 기초

2.1 소켓 개요

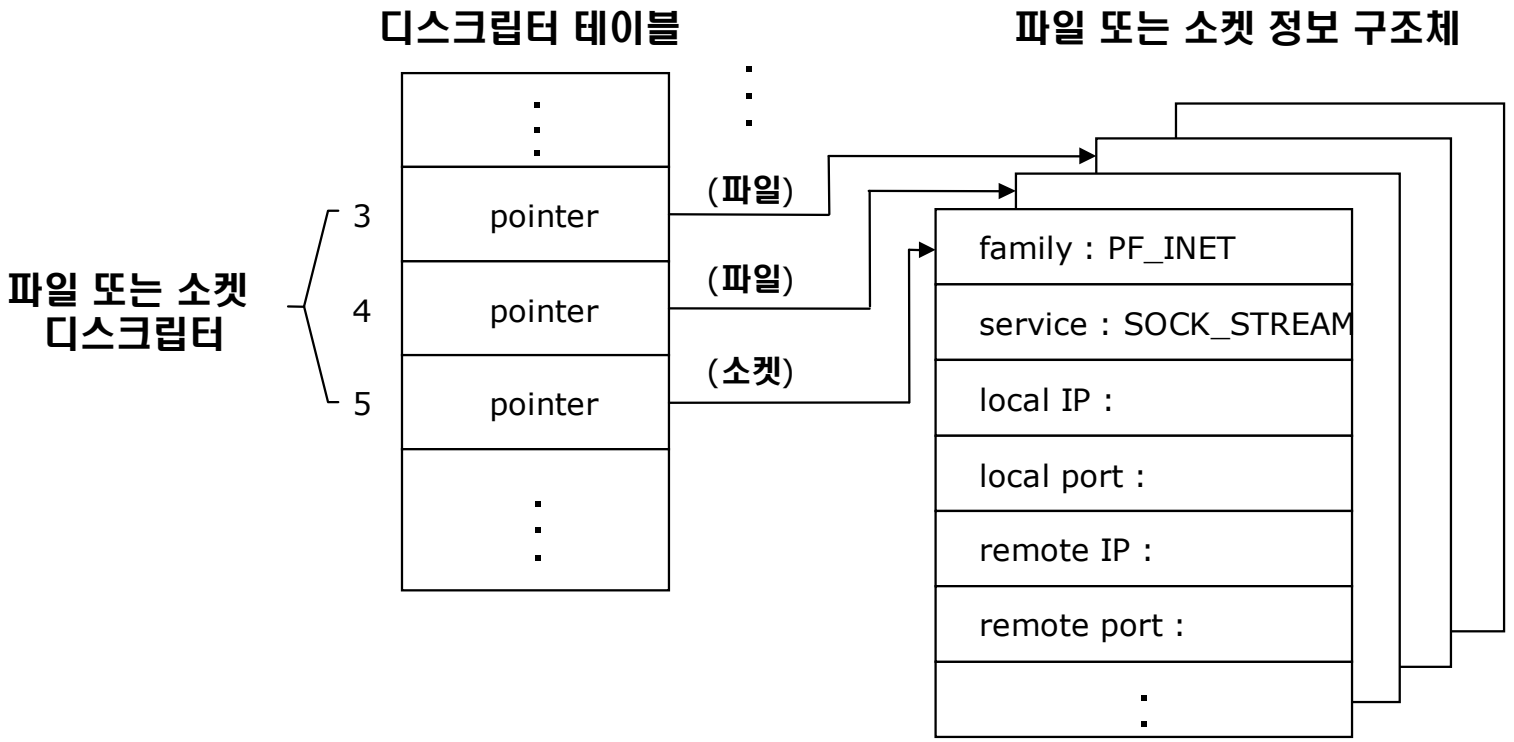
소켓 정의

- TCP나 UDP와 같은 트랜스포트 계층을 이용하는 API
  - 1982년 DBS 유닉스 4.1에서 소개
  - 모든 유닉스 운영체제에서 제공
  - Windows는 Winsock으로 제공
  - Java는 Network 관련 클래스 제공



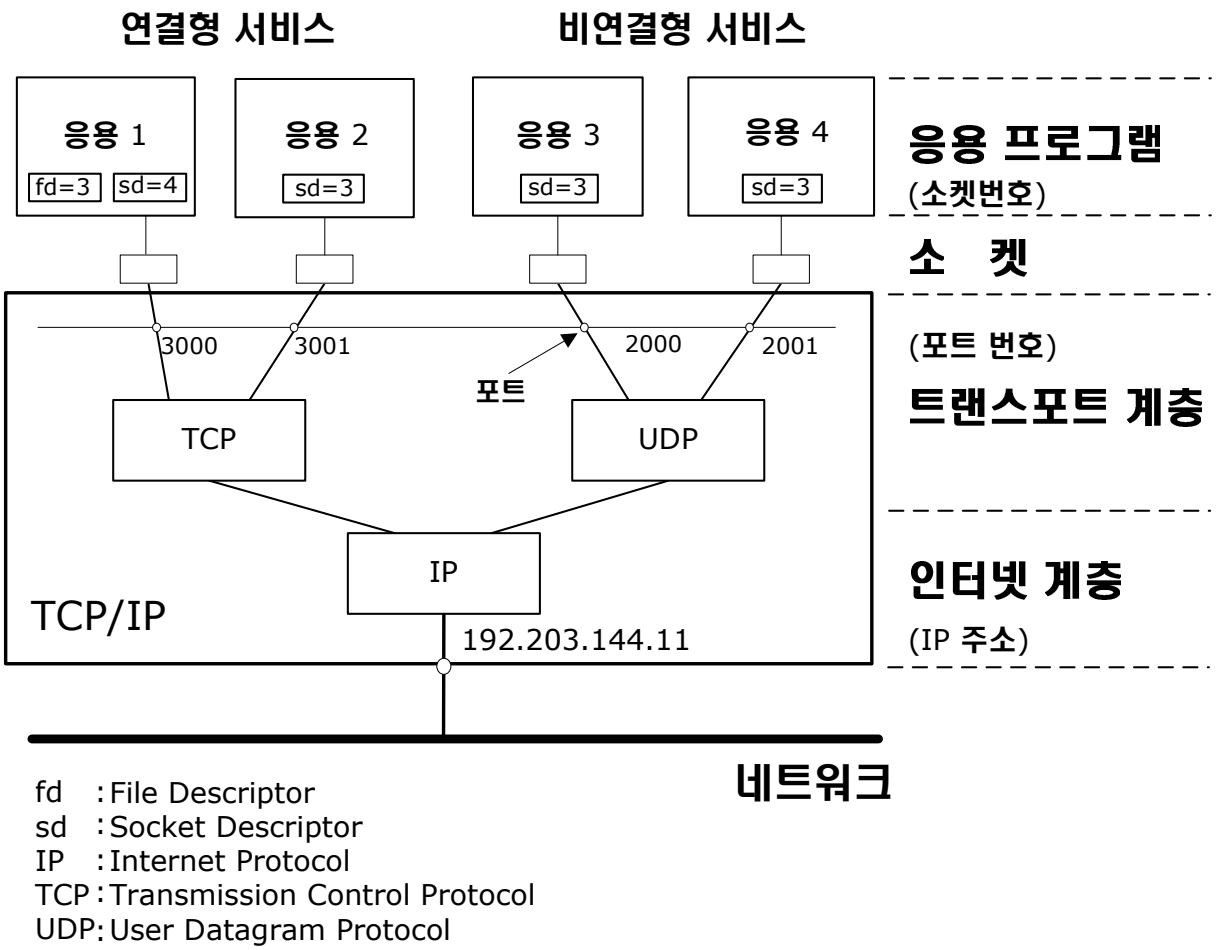
소켓 번호 (1)

- 유닉스는 모든 파일, 장치 등을 파일로 취급
  - 파일 디스크립터, 키보드, 모니터, 하드웨어 장치, 소켓 등
- 소켓 디스크립터
  - 소켓을 개설하여 얻는 파일 디스크립터
  - 데이터를 송수신할 때 사용



소켓번호 (2)

- **응용 프로그램과 소켓 그리고 TCP/IP의 관계**



fd : File Descriptor
sd : Socket Descriptor
IP : Internet Protocol
TCP : Transmission Control Protocol
UDP: User Datagram Protocol

포트번호

- IP 주소
  - IP 데이터그램을 목적지 호스트까지 전달하는 데 사용
  - 특정 호스트를 찾는 데 사용
- 포트번호
  - 16 비트로 표현
  - IP 데이터그램에 실린 데이터를 최종적으로 전달할 프로세스를 구분
  - 호스트내의 통신 접속점(소켓)을 구분하는 데 사용
  - 같은 포트번호를 TCP와 UDP가 동시에 사용 가능
- Well-known 포트
  - 1023번 이하가 배정되어 사용됨
  - 널리 사용되는 서비스를 위해 미리 지정되어 있는 포트번호
  - 예) ftp, telnet, mail, http 등
- /etc/services
  - TCP/IP가 지원하는 응용 서비스와 포트번호가 정리된 파일

소켓 사용법

- **소켓 사용을 위한 정보**
  - **통신에 사용할 프로토콜**(TCP, UDP)
  - **자신의** IP **주소**
  - **자신의 포트번호**
  - **상대방의** IP **주소**
  - **상대방의 포트번호**

소켓의 개설

- **소켓은 TCP/IP만을 위해 정의된 것은 아님**
  - TCP/IP, **유닉스 네트워크**, XEROX **네트워크 등에서 사용 가능**
  - **따라서 소켓 개설 시 프로토콜 체계를 지정해야 함**

```
#include <sys/socket.h>
int socket(
        int domain,          // 프로토콜 체계
        int type,            // 서비스 타입
        int protocol);       // 소켓에 사용할 프로토콜
```

- **지정할 수 있는 프로토콜 체계의 종류**

```
PF_INET      // 인터넷 프로토콜 체계
PF_INET6     // IPv6 프로토콜 체계
PF_UNIX      // 유닉스 방식의 프로토콜 체계
PF_NS        // XEROX 네트워크 시스템의 프로토콜 체계
PF_PACKET    // 리눅스에서 패킷 캡쳐를 위해 사용
```

- **서비스 타입**

```
SOCK_STREAM    // TCP 소켓
SOCK_DGRAM     // UDP 소켓
SOCK_RAW       // Raw 소켓
```

open_socket.c

- **동작**
  - /etc/passwd, /etc/hosts **파일을 열고 파일 디스크립터를 출력**
  - **두 개의 소켓 개설 후 소켓 번호를 확인**
- **주요 코드**

```
// passwd 파일 열기
fd1 = open("/etc/passwd", O_REONLY, 0);                    // 3
printf("/etc/passwd's file descriptor = %d\n", fd);

// 스트림형 소켓 개설
sd1 = socket(PF_INET, SOCK_STREAM, 0);                     // 4
printf(stream socket descriptor = %d\n", sd1);

// 데이터그램형 소켓 개설
sd2 = socket(PF_INET, SOCK_DGRAM, 0);                      // 5
printf(datagram socket descriptor = %d\n", sd2);

// host 파일 열기
fd2 = open("/etc/hosts", O_REONLY, 0);                     // 6
printf("/etc/host's file descriptor = %d\n", fd2);
```

소켓 개설의 한계

- **한 프로세스에서 개설할 수 있는 소켓의 한계**
  - 64 **또는** 1024 **등으로 제한**
  - UNIX**에서는** <sys/types.h>**에** FD_SETSIZE**로 정의되어 있음**
  - getdtablesize()**를 사용하여 개설 가능한 최대 소켓 수 확인 가능**

```
#include <stdio.h>
#include <unistd.h>

printf("getdtablesize(0 \ %d\n", getdtablesize());
```

C 프로그램 환경

- **네트워크 프로그램에서 주로 사용하는 헤더 파일**

```
/usr/include
/usr/include/sys
/usr/include/netinet
```

- man **페이지**
  - **특정 함수의 사용법을 확인**

```
$ man socket
Network Functions                                    socket(3N)

NAME
     socket - create an endpoint for communication

SYNOPSIS
     cc  [ flag ... ] file ... -lsocket -lnsl [ library ... ]
     #include <sys/types.h>
     #include <sys/socket.h>

     int socket(int domain, int type, int protocol);

DESCRIPTION
     socket() creates an endpoint for communication .....
```

소켓주소 구조체 - sockaddr

- **클라이언트 또는 서버의 구체적인 주소를 표현하기 위해 사용**
  - **주소 체계**
  - IP **주소**
  - **포트번호**

```
struct sockaddr {
  u_short sa_family;          // address family
  char sa_data[14];           // 주소
};
```

- socketaddr **사용이 불편함**
  - sockaddr_in **구조체를 대신 사용**

sockaddr_in 구조체

- sockaddr_in은 내부적으로 in_addr 구조체를 사용

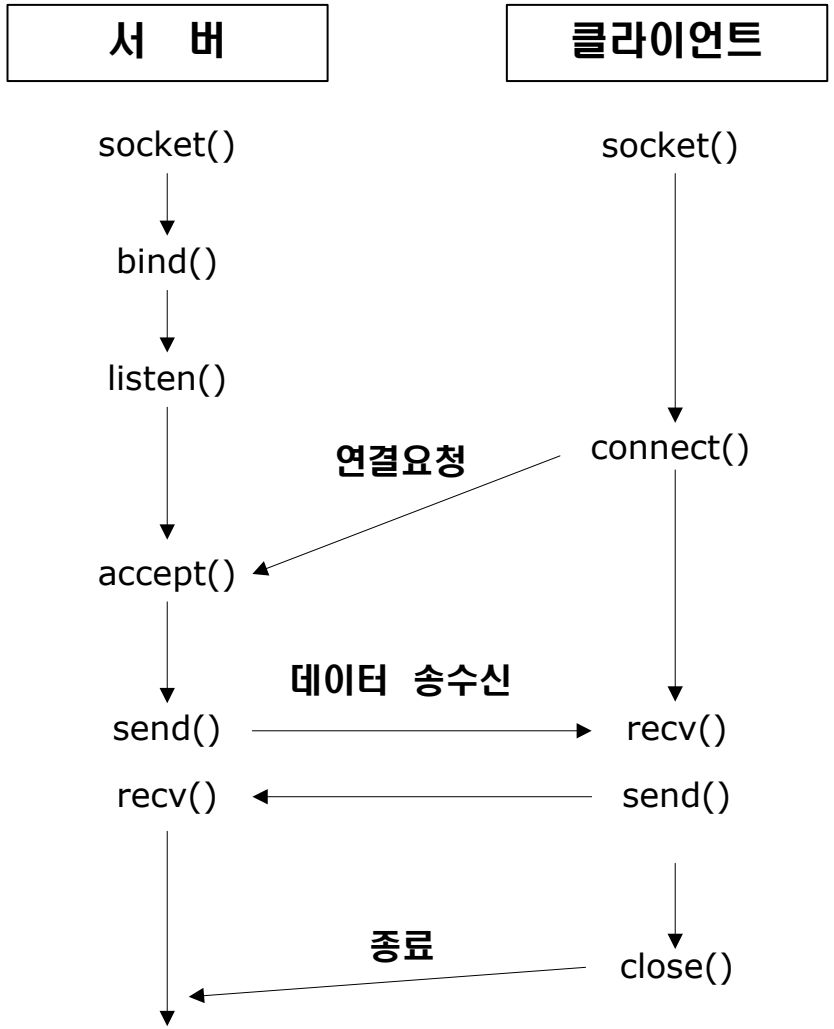
```
struct in_addr {
  u_long s_addr;                    // 32비트의 IP 주소를 저장하는 구조체
};

struct sockaddr_in {
  short sin_family;                 // 주초 체계
  u_short sin_port;                 // 16비트의 포트번호
  struct in_addr sin_addr;          // 32비트의 IP 주소
  char sin_zero[8];                 // 전체 크기를 16바이트로 맞추기 위한 dummy
};
```

- sin_family

```
AF_INET         // 인터넷 주소 체계
AF_UNIX         // 유닉스 파일 주소 체계
AF_NS           // XEROX 주소 체계
```

소켓 사용 절차

- TCP(**연결형**) **소켓 프로그래밍 절차**



2.2 인터넷 주소변환

바이트 순서

- **호스트 바이트 순서**
  - **컴퓨터가 내부 메모리에 숫자를 저장하는 순서**
  - CPU**의 종류에 따라 다름**
    - 80x86 : little-endian
    - MC68000 : big-endian
- **네트워크 바이트 순서**
  - **포트번호나** IP **주소와 같은 정보를 바이트 단위로 전송하는 순서**
  - high-order(big-endian)**로 전송**
- 80x86**과** MC68000**간의 데이터 전송**
  - **바이트 순서가 바뀜**

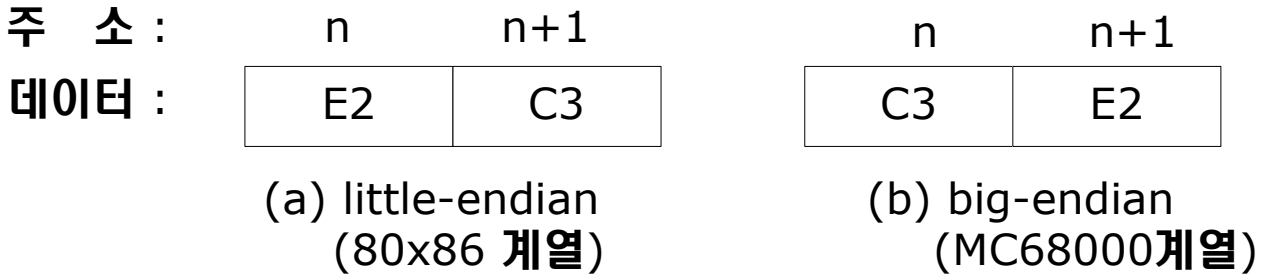
바이트 순서를 바꾸는 함수

- 2바이트와 4바이트의 구분
  - Unsigned short integer 변환

htons() : host-to-network 바이트 변환
ntohs() : network-to-hot 바이트 변환

  - Unsigned long integer 변환

htonl() : host-to-network 바이트 변환
ntohl() : network-to-host 바이트 변환



byte_order.c (1)

- **호스트 바이트 순서와 네트워크 바이트 순서를 확인하는 프로그램**
  - getservbyname() **시스템 콜 사용**

```
pmyservent = getservbyname("echo", "udp");
```

  - servent **구조체**

```
struct servent {
  char *s_name;              // 서비스 이름
  char **s_aliases;          // 별칭 목록
  int s_port;                // 포트
  char *s_proto;             // 프로토콜
};
```

byte_order.c (2)

- **주요 코드**

```
struct servent *servent;
servent = getservbyname("echo", "udp");

if (servent == NULL) {
  printf("서비스 정보를 얻을 수 없음\n\n");
  exit(0);
}

printf("UDP 에코 포트번호(네트워크 순서) : %d\n", servent->s_port);
printf("UDP 에코 포트번호(호스트 순서) : %d\n", ntohs(servent->s_port));
```

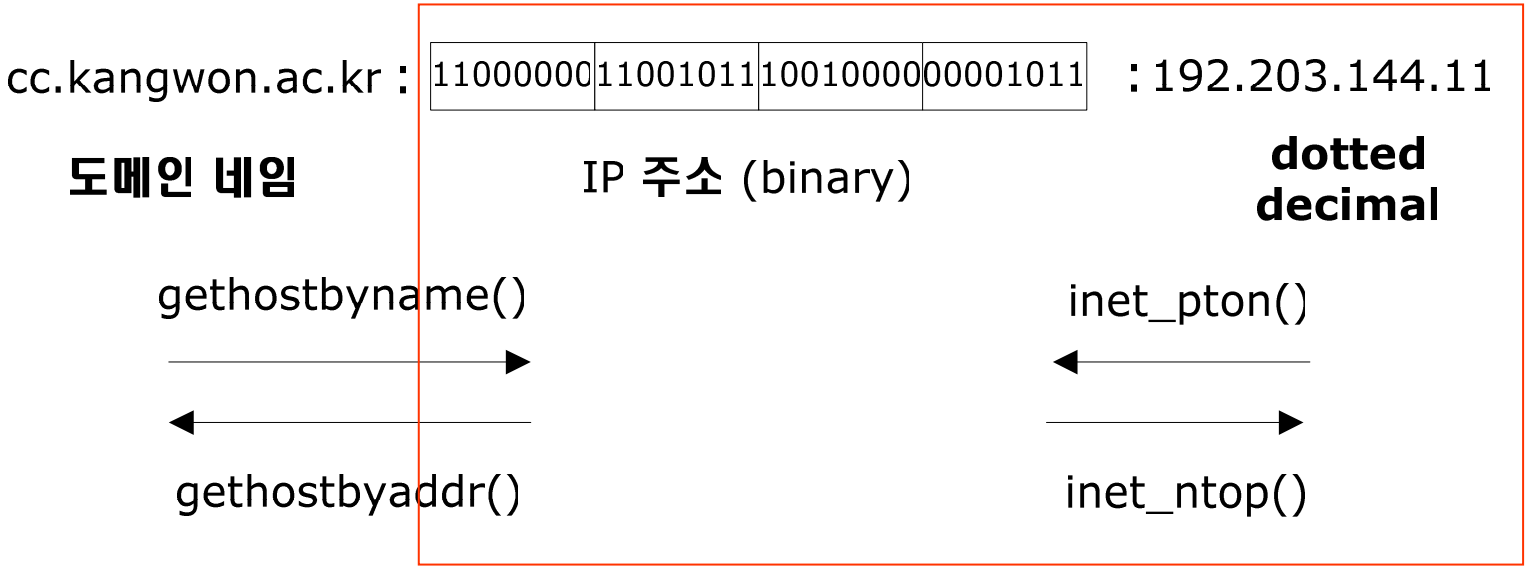
- **실행 결과**

```
// 썬 마이크로시스템즈의 sparc system
$ byte_order
UDP 에코 포트번호(네트워크 순서) : 7
UDP 에코 포트번호(호스트 순서) : 7

// 인텔 80x86 계열 PC
$ byte_order
UDP 에코 포트번호(네트워크 순서) : 1792
UDP 에코 포트번호(호스트 순서) : 7
```

IP 주소 변환

- IP **주소를 도메인 네임과** dotted deciamal **방식으로 표현**
  - dotted decimal**은 15개의 문자로 구성된 스트링 변수**
- **주소 표현법의 상호 변환 함수**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *src, char *dst, size_t cnt);
int inet_pton(int af, const char *src, void *dst);
```



ascii_ip.c

- dotted decimal **표현의 주소를** 4 **바이트의** IP **주소로 출력**
- **변환된** 4 **바이트의** IP**를** dotted decimal **표현으로 출력**
- **주요 코드**

```
inet_pton(AF_INET, argv[1], &inaddr.s_addr);
printf("inet_pton(%s) = 0x%X\n", argv[1], inaddr.s_addr);

inet_ntop(AF_INET, &inaddr.s_addr, buf, sizeof(buf));
printf("inet_ntop(0x%X) = %s\n", inaddr.s_addr, buf);
```

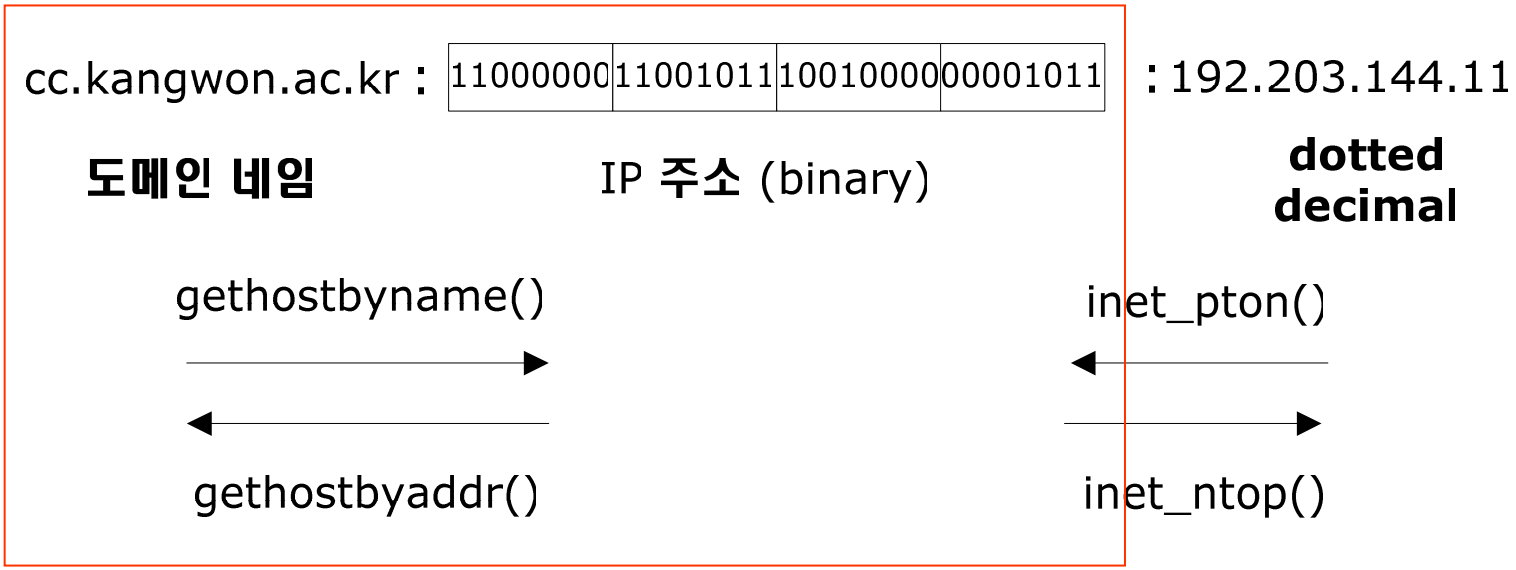
- **실행 결과**

```
$ ascii_ip 210.15.36.231
  * 입력한 dotted decimal IP 주소 : 210.15.36.231
    inet_pton(210.115.36.231) = 0xE72473D2
    inet_ntop(0xE72473D2) = 210.115.36.231
```

도메인 주소변환 (1)

- DNS(Domain Name Service)
  - **도메인 네임으로부터 IP 주소를 반환**
  - **IP 주소로부터 도메인 네임을 반환**
- **도메인 주소 변환 함수**

```
#include <netdb.h>
struct hostent *gethostbyname(const char *hname);
struct hostent *gethostbyaddr(const char *in_addr, int len, int family);
```



도메인 주소변환 (2)

- gethostbyname()
  - hname에 해당하는 호스트의 정보를 hostent 구조체 포인터를 반환
- gethostbyaddr()
  - in_addr, 길이, 주소 타입으로부터 hostent 구조체 포인터 반환

- hostent 구조체

```
struct hostent {
  char* h_name;                    // 호스트 이름
  char** h_aliases;                // 호스트 별칭
  int h_addrtype;                  // 호스트 주소의 종류
  int h_length;                    // 주소의 크기
  char** h_addr_list;              // IP 주소 리스트
};

#define h_addr h_addr_list[0]      // 첫 번째(대표) 주소
```

get_hostent.c

- **주요 코드**

```
hp = gethostbyname(argv[1]);
if (hp == NULL) {
  printf("gethostbyname fail\n");
  exit(0);
}

printf("호스트 이름 : %s\n", hp->hname);
printf("호스트 주소타입 번호 : %d\n", hp->h_addrtype);
printf("호스트 주소의 길이 : %d\n", hp->h_length);

for(i=0; hp->h_addr_list[i]; i++) {
  memcpy(&in.s_addr, hp->h_addr_list[i], sizeof(in.s_addr));
  inet_ntop(AF_INET, &in, buf, sizeof(buf));
  printf("IP주소(%d번째) : %s\n", i+1, buff);
}

for (i=0; hp->h_aliases[i], i++)
  printf("호스트 별명(%d 번째) : %s", i+1, hp->h_aliases[i]);
```

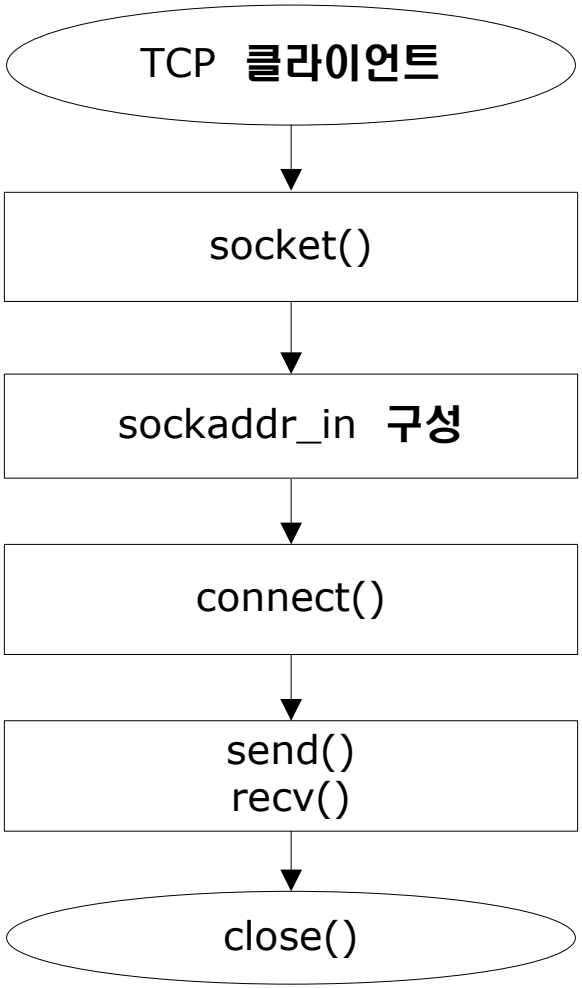
get_host_byaddr.c

- **주요 코드**

```
inet_pton(AF_INET, argv[1], &in.s_addr);
myhost = gethostbyaddr((char *)&(in.s_addr), sizeof(in.s_addr), AF_INET);

if (myhost == NULL) {
  printf("Error at gethostbyaddr()\n");
  exit(0);
}

printf("호스트 이름 : %s\n", myhost->h_name);
```

2.3 TCP **클라이언트 프로그램**

TCP 클라이언트 프로그램 작성 절차



소켓 개설

서버의 IP 주소와 포트 번호 지정

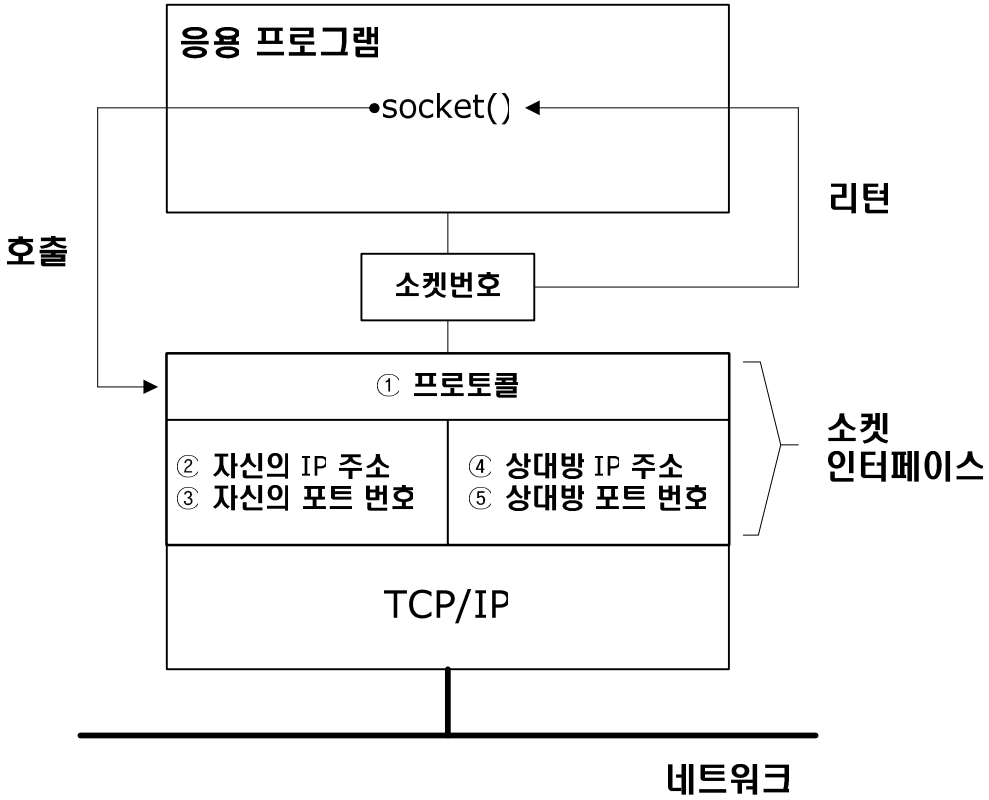
서버로 연결 요구

데이터 송수신

socket(), 소켓 개설

- TCP, UDP **소켓을 선택**
  - TCP : SOCK_STREAM
  - UDP : SOCK_DGRAM

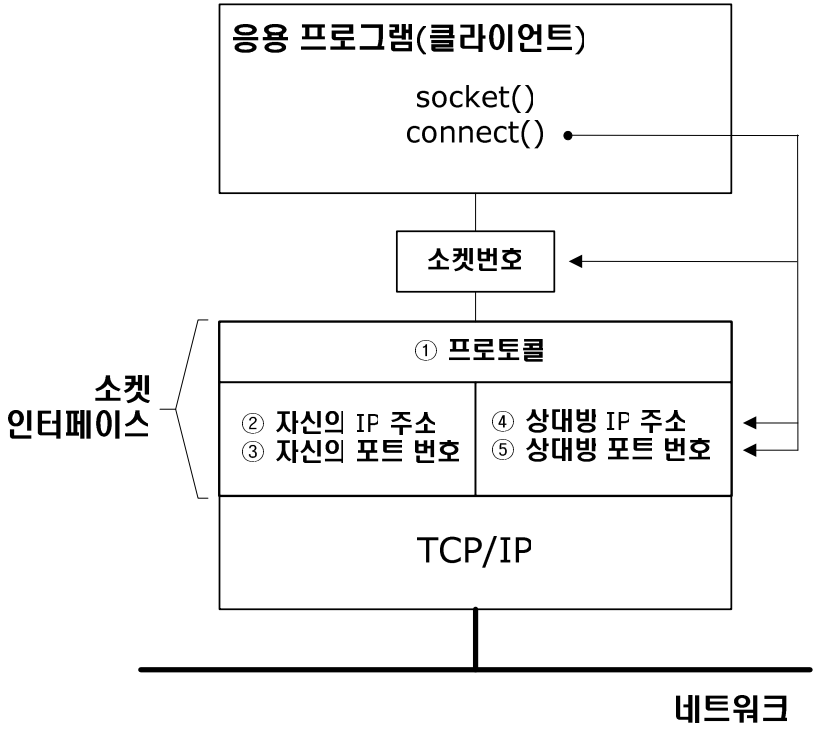
- socket() **호출 시 소켓번호 리턴 과정**



connect(), 서버에 연결 요청

- connect()

```
int connect (
  int s,                              // 서버와 연결시킬 소켓번호
  const struct sockaddr *addr,        // 상대방 서버의 소켓주소 구조체
  int addrlen);                       // 구조체 *addr의 크기
```



send(), recv(), 데이터 송수신

- TCP **소켓의 데이터 송수신 함수**

문 법	인 자	
int send(int s, char *buf, int length, int flags);	s	**소켓번호**
	buf	**전송할 데이터가 저장된 버퍼**
	length	buf**의 크기**
	flags	**보통** 0
int write(int s, const void* buf, int length);	s	**소켓번호**
	buf	**전송할 데이터가 저장된 버퍼**
	length	buf**의 길이**
int recv(int s, char* buf, int length, ing flags);	s	**소켓번호**
	buf	**수신 데이터를 저장할 버퍼**
	length	buf**의 길이**
	flags	**보통** 0
int read(int s, void* buf, int length);	s	**소켓번호**
	buf	**수신 데이터를 저장할 버퍼**
	length	bug**의 길이**

close(), 소켓 닫기

- **소켓의 사용을 종료**
- close()**는 서버, 클라이언트에 관계없이 호출 가능**
- close()**를 호출한 시점에서의 송신버퍼**
  - **아직 전송되지 못한 데이터는 모두 전달된 후 연결 종료**
  - **전달중인 데이터는 모두 전달된 후 연결 종료**
  - **소켓 옵션을 이용하여 미전송 데이터를 모두 버리고 종료**

mydaytime.c

- **주요 코드**

```
s = socket(PF_INET, SOCK_STREAM, 0);                    // 소켓 개설

bzero((char *)servaddr, sizeof(servaddr))               // 서버 소켓 구조체 초기화

struct sockaddr_in servaddr;                            // 서버 소켓 구조체
servaddr.sin_family = AF_INET;                          // 주소 체계
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);        // 32비트 IP주소로 변환

servaddr.sin_port = htons(13);                          // daytime 서비스 port

connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr));

n = read(s, buf, sizeof(buf));                          // 메시지 수신
```

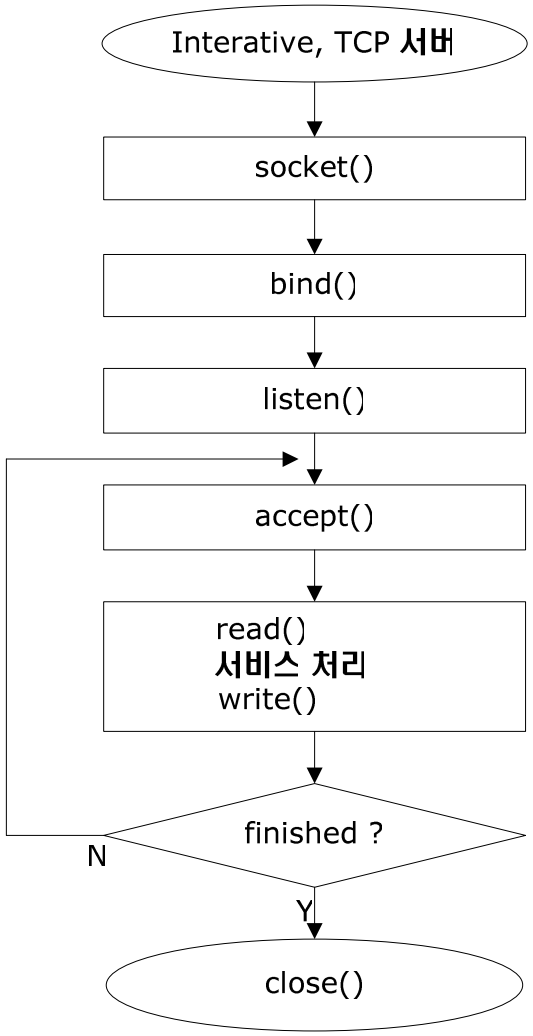
tcp_echocli.c

- **주요 코드**

```
servaddr.sin_port = htons(7);                    // echo 서비스 port

fgets(buf, sizeof(buf), stdin)                   // echo 메시지 입력
```

2.4 TCP 서버 프로그램

TCP 서버 프로그램 작성 절차



소켓 개설

소켓번호와
소켓주소의 결합

연결을 기다림

연결 수락

클라이언트의 요구를
처리한 후 결과 전송

socket(), 소켓 생성

- **클라이언트와 통신하기 위해 소켓을 생성해야 함**
  - **연결형 소켓** : SOCK_STREAM
  - **비연결형 소켓** : SOCK_DGRAM

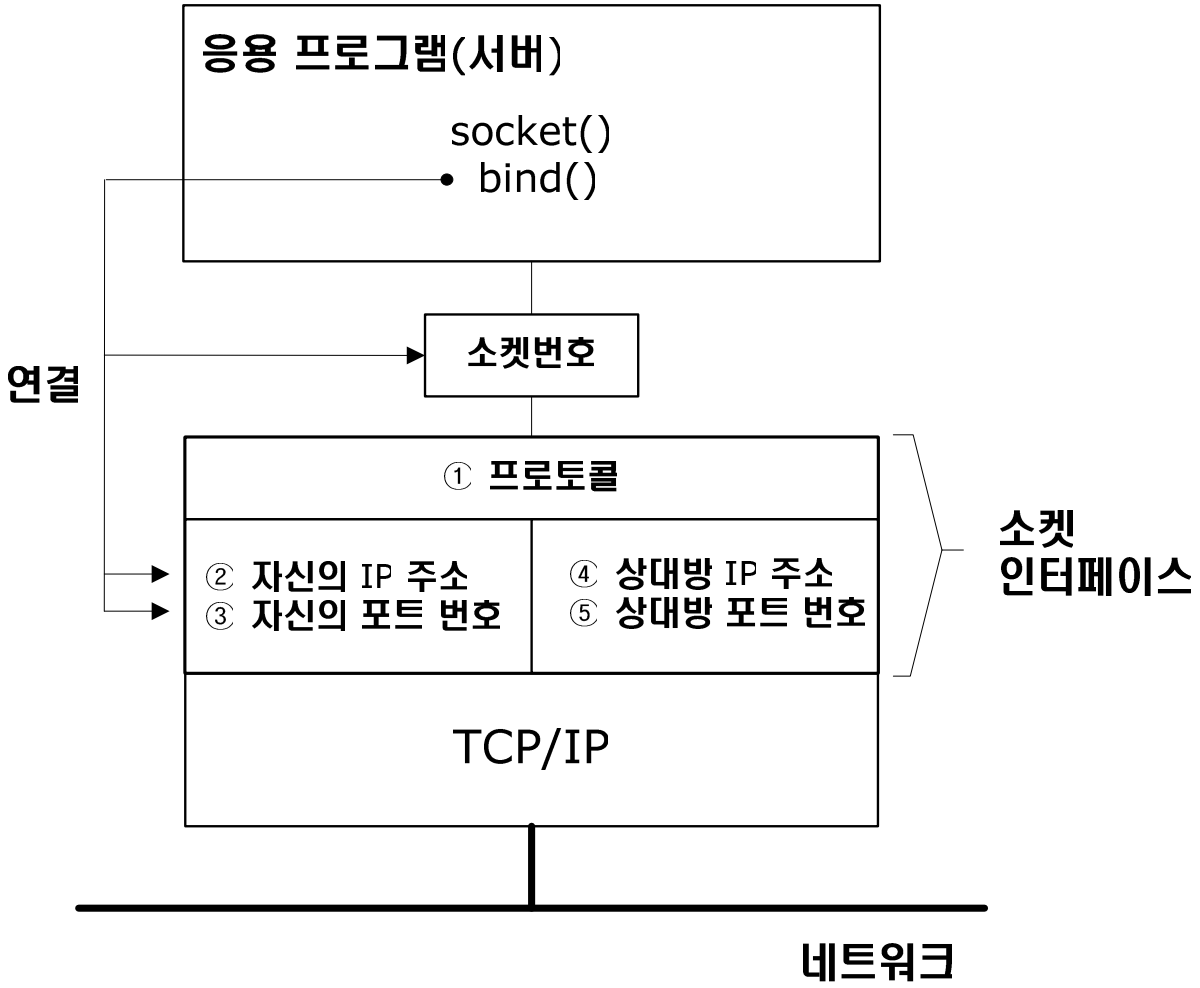
bind()

- **생성된 소켓은 응용 프로그램 내에서 유일한 소켓번호를 배정받음**
- **소켓 번호**
  - **응용 프로그램만 알고 있는 번호**
  - **외부와 통신하기 위해서 IP 주소와 포트번호를 연결해야 함**
- bind()
  - **IP 주소와 포트번호를 연결하기 위해 사용**

```
int bind(
  int s,                        // 소켓 번호
  struct sockaddr *addr,        // 서버 자신의 소켓주소 고조체 포인터
  int len);                     // *addr 구조체의 크기
```

```
s = socket(PF_INET, SOCK_SREAM, 0);
struct sockaddr_in servaddr;

servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
inet_pton(AF_INET, "203.252.65.3", &servaddr.sin_addr);
servaddr.sin_port = htons(SERV_PORT);

bind(s, (struct sockaddr *)&servaddr, sizeof(servaddr));
```

bind() 호출 시 소켓번호와 소켓주소와의 관계



listen()

- **클라이언트의 연결 요청을 받아들이기 위해 사용**
  - **능동적 소켓 : 요청을 보내는 클라이언트 소켓**
  - **수동적 소켓 : 연결 요청을 받아들이는 서버의 소켓**
- listen()**을 이용하여 수동적 소켓으로 변경**
  - socket()**에 의해 생성되는 소켓은 기본적으로 능동적 소켓임**

```
int listen(
  int s,                // 소켓번호
  int backlog);         // 연결을 기다리는 클라이언트의 최대 수
```

accept()

- **클라이언트와 설정된 연결을 실제로 받아들이기 위해 사용**

```
int accept (
  int s,                          // 소켓번호
  struct sockaddr *addr,          // 연결요청을 한 클라이언트의 소켓주소 구조체
  int *addrlen);                  // *addr 구조체 크기의 포인터
```

- **반환 값**
  - **성공 : 접속된 클라이언트와의 통신에 사용할 새로운 소켓 번호를 반환**
  - **실패 : -1 반환**

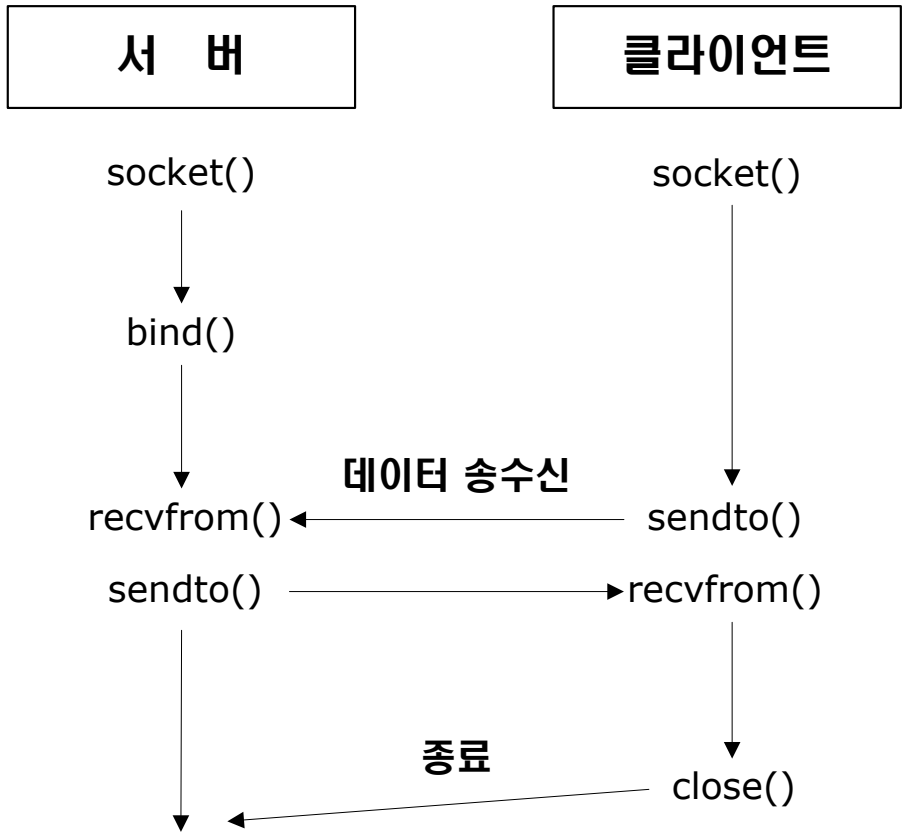
tcp_echoserv.c

- **주요 코드**

```
if ((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
  perror("socket fail");  exit(0);
}

bzero((char *)&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr(htonl(INADDR_ANY);
servaddr.sin_port = htons(atoi(argv[i]));

if (bind(listen_sock, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
  perror("bind fail");  exit(0);
}

listen(listen_sock, 5);

while(1) {
  accp_sock = accept(listen_sock, (struct sockaddr *)&cliaddr, &addrlent);
  if (accp_sock < 0) { perror("accept fail"); exit(0); }
  ...
}
```

UDP 프로그램

- TCP와 달리 일 대 일 통신에만 사용되지 않음
- **비연결형 소켓**
  - connect() **시스템 콜을 사용할 필요가 없음**
  - **소켓 개설 후 바로 상대방과 데이터를 송수신**



데이터 송수신 함수

- **데이터를 송수신**
  - **각 데이터그램마다 목적지의 IP주소와 포트번호가 주어져야 함**

문 법	인 자	
int sendto(int s, char* buf, int length, int flags, sockaddr* to, int tolen)	s	**소켓번호**
	buf	**전송할 데이터가 저장된 버퍼**
	length	buf **버퍼의 크기**
	flags	**보통** 0
	to	**목적지의 소켓주소 구조체**
	tolen	to **버퍼의 크기**
int recvfrom(int s, char* buf, int length, int flags, sockaddr* from, int* fromlen)	s	**소켓번호**
	buf	**수신 데이터를 저장할 버퍼**
	length	buf **버퍼의 길이**
	flags	**보통** 0
	from	**발신자의 소켓주소 구조체**
	fromlen	from **버퍼의 길이**

udp_echocli.c

- **주요 코드**
  - tcp_echocli.c**에서 변경되는 부분**

```
socket(PF_INET, SOCK_DGRAM, 0);

...

sendto(s, buf, strlen(buf), 0, (struct sockaddr *)&servaddr, addrlen);
recvfrom(s, buf, MAXLINE, 0, (struct sockaddr *)&servaddr, &addrlen);
```

udp_echoserv.c

- **한 쪽이** sendto()**를 호출했으면 상대방은 반드시** recvfrom()**을 호출하고 있어야 함**
- **주요코드**

```
if ((s = socket(PF_INET< SOCK_DGRAM, 0)) < 0) {
  perror("socket fail"); exit(0); }

...

while(1) {
  nbyte = recvfrom(s, buf, MAXLINE, 0, (struct sockaddr *)&cliaddr, &addrlen);

  ...

  if (sendto(s, buf, nbyte, 0, (struct sockaddr *)&cliaddr, addrlen) < 0) {
    perror("sendto fail"); exit(1); }
}
```

Connected UDP

- UDP 소켓
  - sendto()로 데이터를 전송하는 순간 소켓과 커널이 내부적으로 연결
- UDP 소켓을 통해서 처리속도 향상을 위해 사용 가능
  - 커널은 인자로 받은 소켓주소와 UDP 소켓을 내부적으로 연결해 둠
  - TCP와 같이 3-way 핸드쉐이크 연결설정이 이루어지지 않음
  - connect() 함수를 호출
  - send()나 recv() 함수를 사용해야 함
- 유의사항
  - 잘못된 IP 주소를 인자로 주어도 connect()에서 에러가 발생하지 않음
    - 3-way 핸드쉐이크가 수행되지 않기 때문
  - read(), recv() 실행시에 에러가 발생
  - 연결된 UDP의 고정된 통신 상대를 connect()로 변경 가능
- 연결된 UDP 사용 종료
  - sin_family = AF_UNSPEC로 설정하고 connect() 호출