2. 소켓 프로그래밍 기초

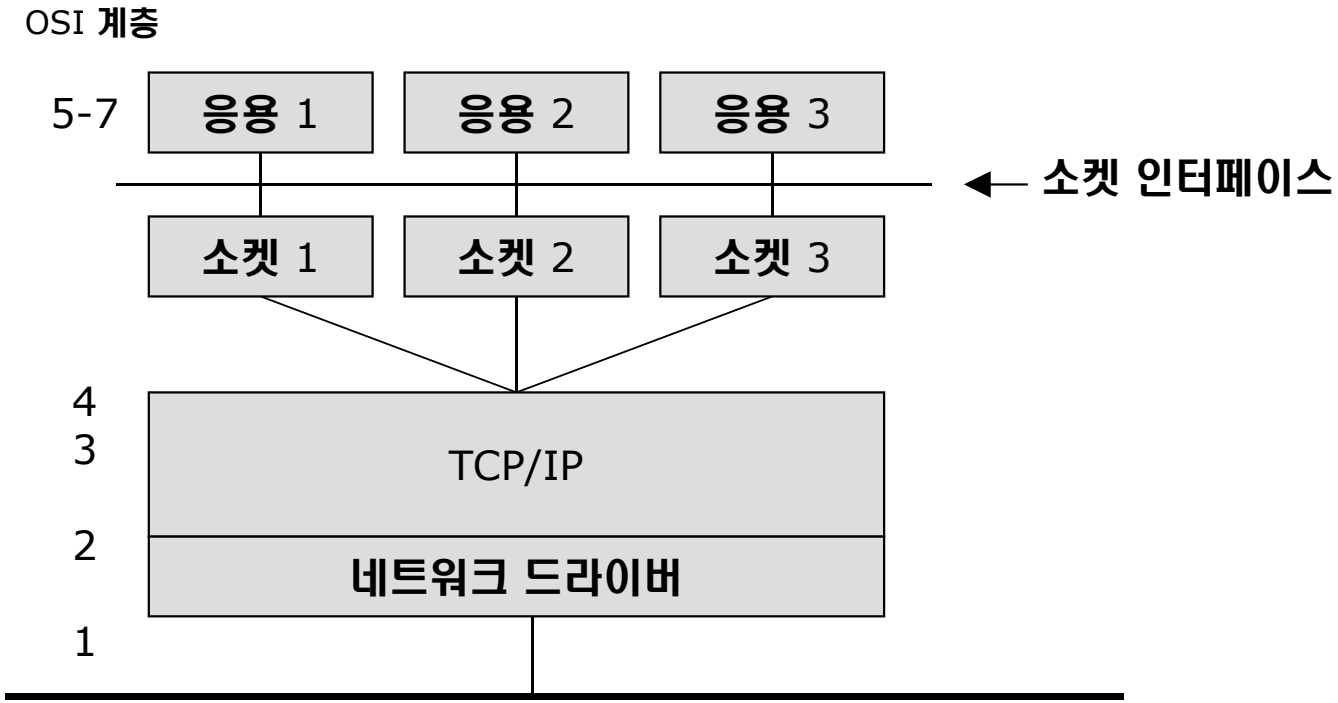
- **소켓 개요**
- **주소변환**
- TCP **프로그램**
- UDP **프로그램**

2.1 소켓 개요

- 소켓 정의
- 소켓 사용법

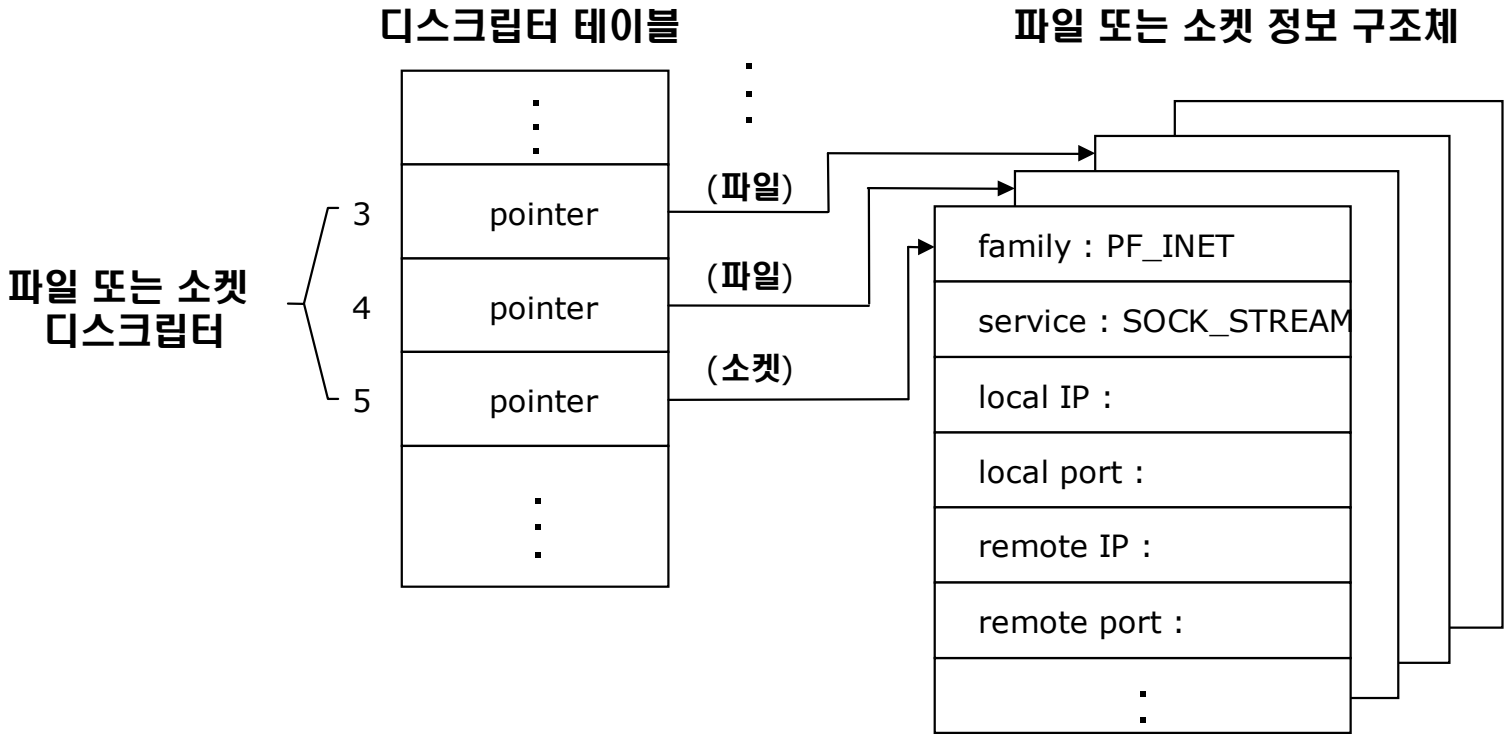
소켓 정의

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



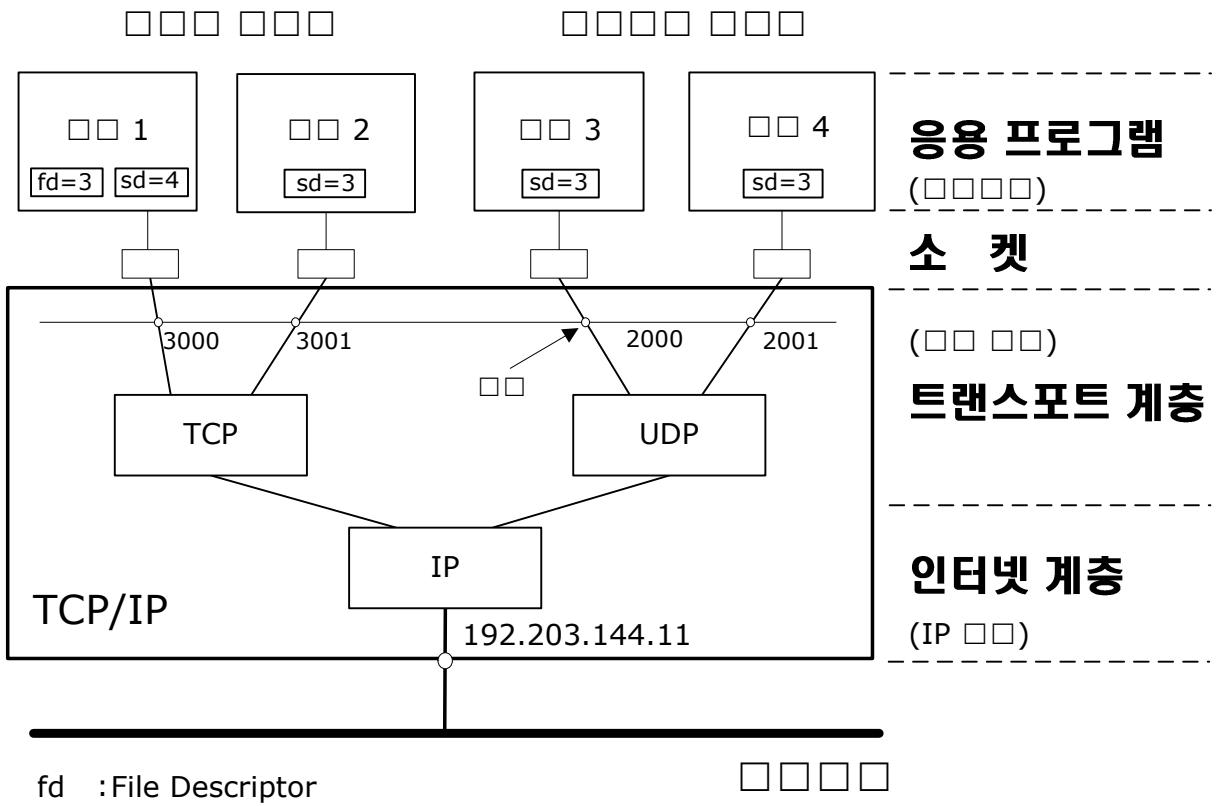
소켓 번호 (1)

- 유닉스는 모든 파일, 장치 등을 파일로 취급
  - 파일, 키보드, 모니터, 하드웨어 장치, 소켓 등
- 소켓 디스크립터
  - 소켓을 개설하여 얻는 파일 디스크립터
  - 데이터를 송수신할 때 사용
- 파일 디스크립터 : 표준입력(0), 표준출력(1), 표준에러(2)



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 사용
  - 널리 사용되는 서비스를 위해 미리 지정되어 있는 포트번호
  - 예) ftp, telnet, mail, http 등
- /etc/services
  - TCP/IP가 지원하는 응용 서비스와 포트번호가 정리된 파일

/etc/services

```
# service-name  port/protocol  [aliases ...]   [# comment]

tcpmux          1/tcp                   # TCP port service multiplexer
tcpmux          1/udp                   # TCP port service multiplexer
rje             5/tcp                   # Remote Job Entry
rje             5/udp                   # Remote Job Entry
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
systat          11/udp          users
daytime         13/tcp
daytime         13/udp
                . . .
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
```

소켓 사용법

- **소켓 사용을 위해 필요한 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(0)
```

- **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, 연결형
SOCK_DGRAM     // UDP 소켓, 비연결형
SOCK_RAW       // Raw 소켓, TCP, UDP 계층을 거치지않고 IP 계층을 이용하는 프로그램
```

open_socket.c

- **파일과 소켓을 열고 파일과 소켓 디스크립터를 알아보는 프로그램**
  - /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", fd1);

// 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**
  - **최대수가** 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() = %d\n", getdtablesize());
```

C 프로그램 환경

- **네트워크 프로그램에서 주로 사용하는 헤더 파일**

```
/usr/include            // stdio.h, stdlib.h … 기본적으로 사용하는 헤더 파일
/usr/include/sys        // socket.h, types.h, ipc.h … 시스템 관련 헤더 파일
/usr/include/netinet    // in.h, arp.h, igmp.h, ip.h, tcp.h … 인터넷 관련 헤더 파일
```

  - socket.h**에 함수** socket, bind, connect, send, recv**와 구조체** socketaddr **를 정의**
  - man**으로 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
};
```

- sockaddr **사용이 불편함**
  - 14 **바이트에** IP**주소 와 포트번호를 구분하여 쓰거나 읽기가 불편**
  - 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;                 // 주소 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
};
```

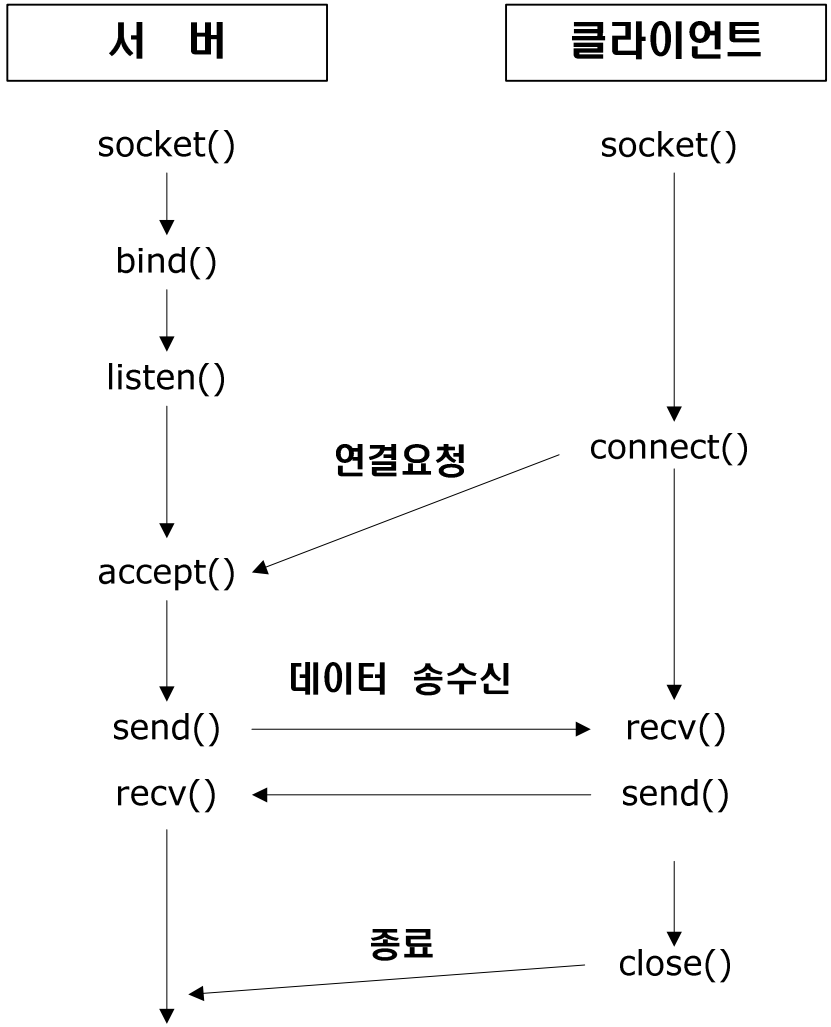
- sin_family

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

- PF_INET과 AF_INET은 모두 2로 혼용하여 사용

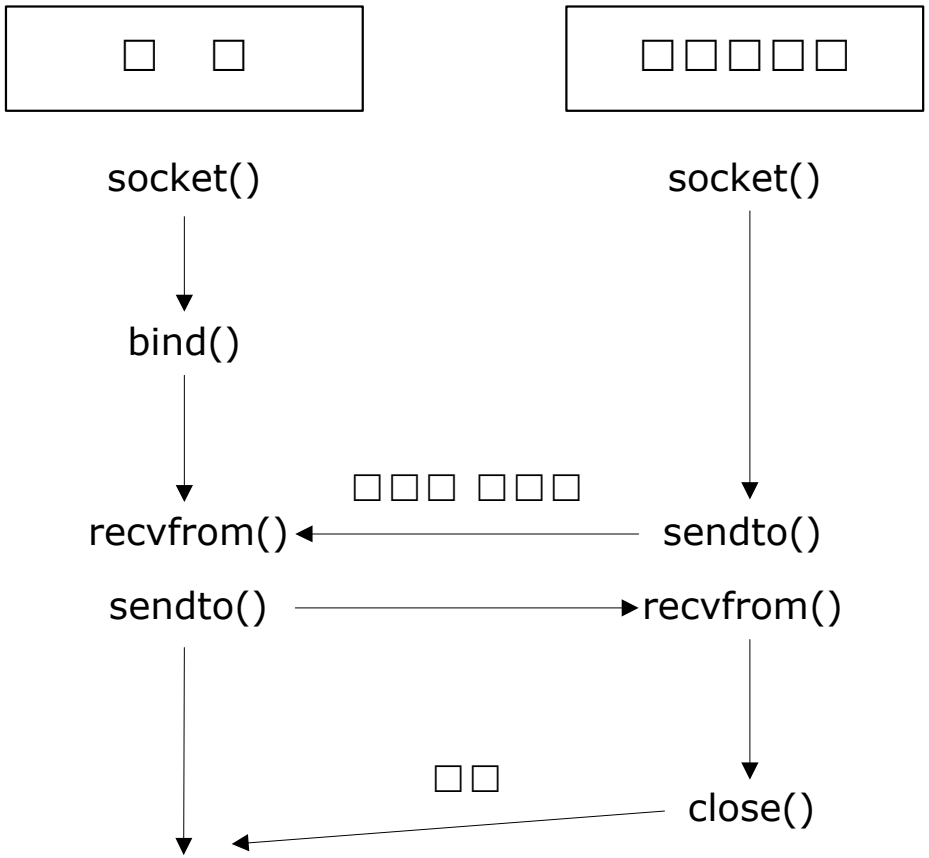
소켓 사용 절차

- TCP(**연결형**) **소켓 프로그래밍 절차**



소켓 사용 절차

- UDP(**비연결형**) **소켓 프로그래밍 절차**

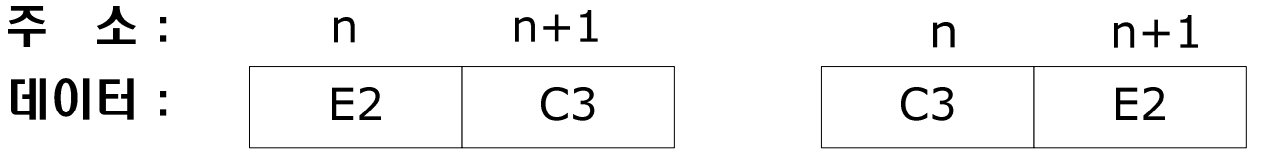


2.2 인터넷 주소변환

- **컴퓨터마다 숫자에 대한 내부 표현 방식이 다르다.**
  - **바이트 순서를 맞추는 절차가 필요**
- **인터넷 주소를 표현하는 방식간의 변환 방법**
  - **도메인 네임**, 32**비트** IP **주소**, dotted decimal

바이트 순서

- **호스트 바이트 순서**
  - **컴퓨터가 내부 메모리에 숫자를 저장하는 순서**
  - CPU**의 종류에 따라 다름**
    - **인텔 계열** : little-endian
    - **모토롤라 계열** : big-endian
  - **예** : 0xC3E2(**십진수** 50146)**의 호스트 바이트 순서 비교**



(a) little-endian
(80x86 **계열**)

(b) big-endian
(MC68000**계열**)

- **네트워크 바이트 순서**
  - **포트번호나** IP **주소와 같은 정보를 바이트 단위로 전송하는 순서**
  - high-order(big-endian)**로 전송**
- **인텔 계열과 모토롤라 계열 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-host **바이트 변환**

  - Unsigned long integer **변환**

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

- **네트워크로 전송하기 전에** htons() **함수를 사용하여 네트워크 바이트 순서로 바꾸고,**
- **반대로 네트워크로 부터 수신한 숫자는** ntohs() **함수를 사용하여 호스트 바이트 순서로 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
};
```

  - servent **구조체는 네트워크로부터 얻은 정보이므로 호스트 화면에 출력하려면 호스트 바이트 순서로 변환시켜 출력**

byte_order.c (2)

- **주요 코드**

컴파일 : cc –o byte_order byte_order.c

```
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));
```

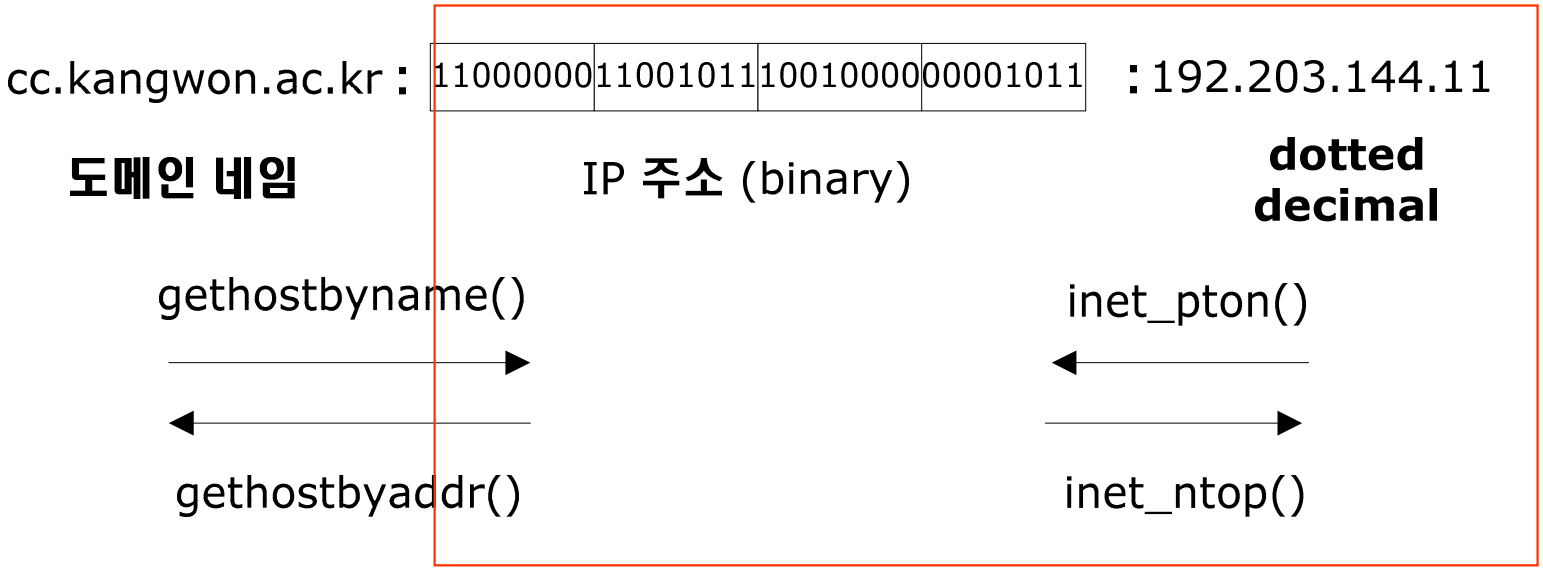
- **실행 결과**

```
// big-endian 사용하는 시스템
$ byte_order
UDP 에코 포트번호(네트워크 순서) : 7                #0x0007
UDP 에코 포트번호(호스트 순서) : 7

// 인텔 80x86 계열 PC, little–endian 사용하는 시스템
$ byte_order
UDP 에코 포트번호(네트워크 순서) : 1792             #0x0700
UDP 에코 포트번호(호스트 순서) : 7
```

IP 주소 변환

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

```
// IPv4 와 IPv6 주소들을 바이너리 형태에서 텍스트(dotted decimal) 형태로 변경
const char *inet_ntop(int af, const void *src, char *dst, size_t cnt);

//  IPv4 와 IPv6 주소들을 텍스트에서 바이너리 형태로 변경
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**

```
struct in_addr  inaddr;         // 32비트 IP 주소 구조체
char   buf[20];

if(argc < 2) {
        printf("사용법 : %s IP 주소(dotted decimal) \n", argv[0]);
        exit(0);
}
printf("* 입력한 dotted decimal IP 주소: %s\n", argv[1]);
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);
```

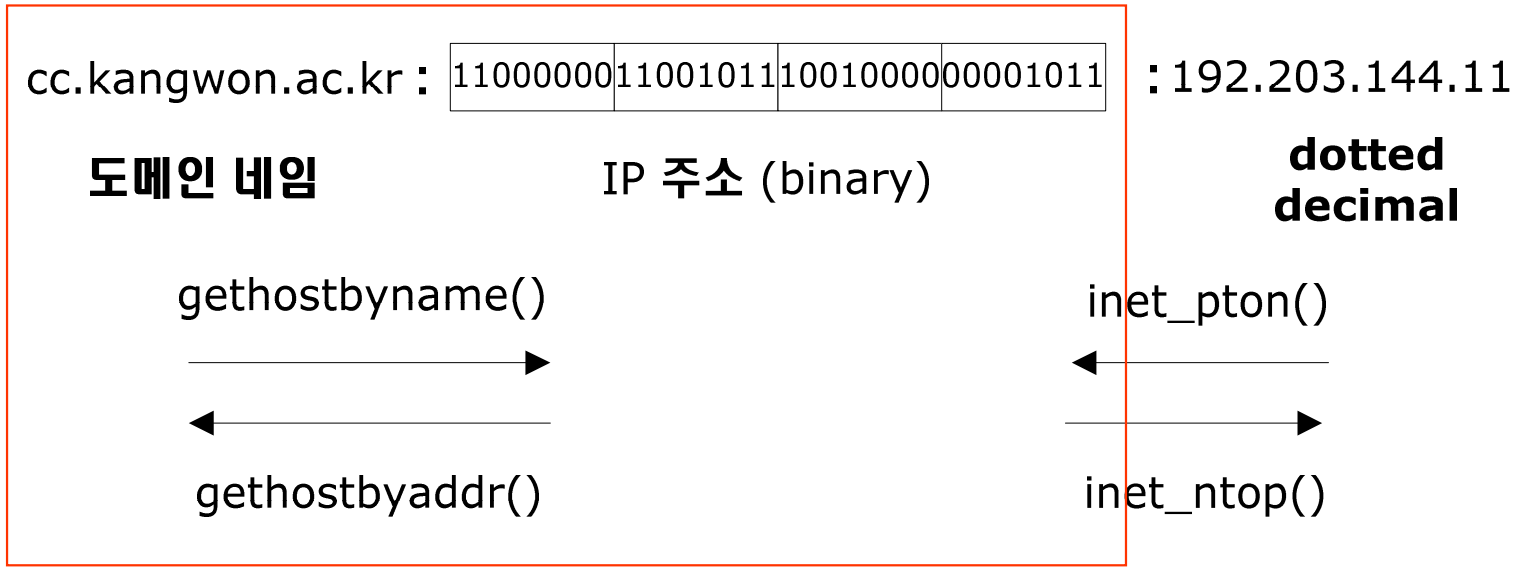
- **컴파일 및 실행 결과**

```
// 컴파일 : cc -o ascii_ip ascii_ip.c
$ 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**

```
// 도메인 네임을에 대한 hostent 를 반환
struct hostent *gethostbyname(const char *hname);
// 바이너리 형태의 IP 주소에 대한 hostent 를 반환
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, IPv4에서 4바이트
  char** h_addr_list;              // IP 주소 리스트
};

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

get_hostent.c :도메인 이름을 인자로 받아 호스트 이름, 별명, 주소체계, dotted decimal 주소를 출력

- **실행** : get_hosttent www.kangwon.ac.kr
- **주요 코드**

컴파일 : gcc -o get_hostent get_hostent.c

```
$ get_hostent www.kangwon.ac.kr
호스트 이름              : guide.kangwon.ac.kr
호스트 주소타입 번호 : 2
호스트 주소의 길이    : 4
IP 주소(1 번째)         : 192.203.144.27
호스트 별명(1 번째)   : www.kangwon.ac.kr
```

```
struct hostent *hp;
struct in_addr in;
int i;
char   buf[20];
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)); /*dotted decimal로 변환 */
  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

- dotted decimal**을 인자로 받아 도메인 이름을 화면에 출력**
- **실행** : get_host_byaddr 211.32.119.151
- **주요 코드**

```
cc -o get_host_byaddr get_host_byaddr.c
```

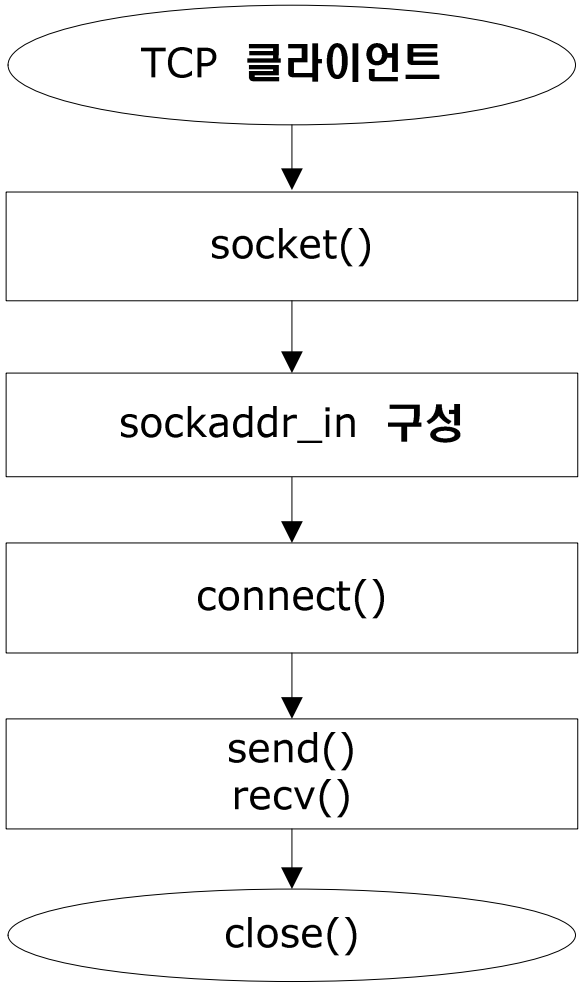
```
$ get_host_byaddr  211.32.119.151
   호스트 이름  : yahoo.co.kr
```

```
struct hostent   *myhost;
struct in_addr   in;

if(argc < 2) {
        printf("사용법 : %s ip_address \n", argv[0]);
        exit(0);
}
/*dotted decimal -> 4바이트 IP주소로 변환*/
inet_pton(AF_INET, argv[1], &in.s_addr);
/* 4바이트 IP주소를  도메인 이름으로 변환 */
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 프로그램의 작성 절차
- 유닉스 서버가 제공하는 서비스를 사용하는 클라이언트 프로그램 작성

TCP 클라이언트 프로그램 작성 절차



소켓 개설

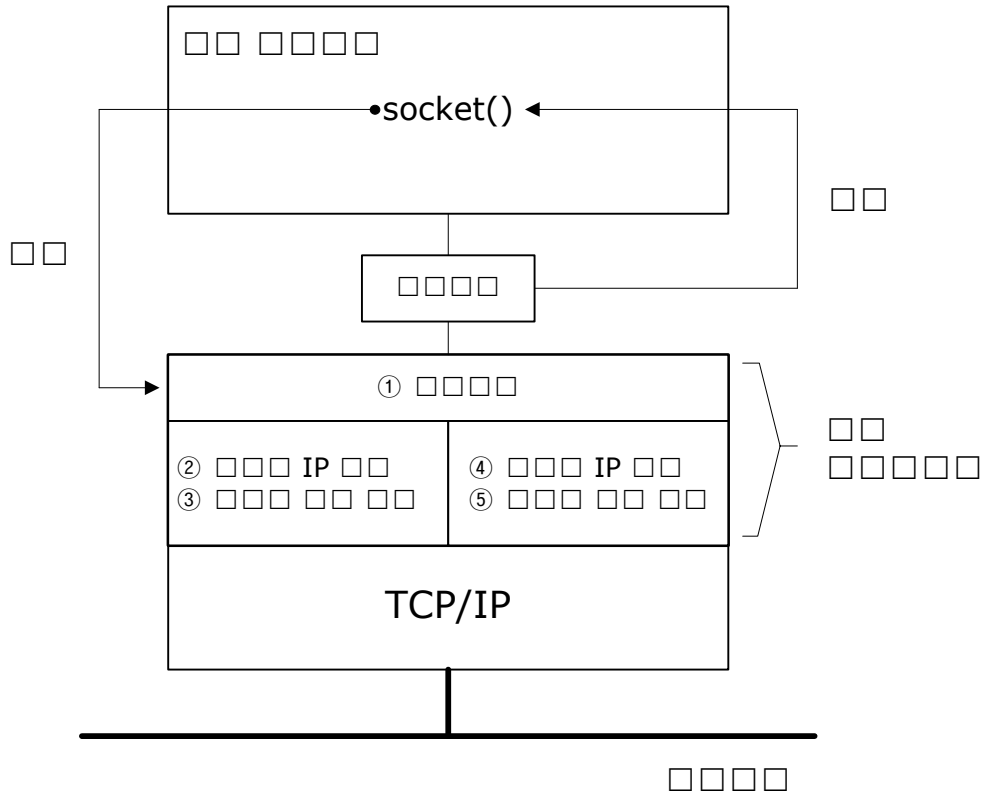
서버의 IP 주소와 포트 번호 지정

서버로 연결 요구

데이터 송수신

socket(), 소켓 개설

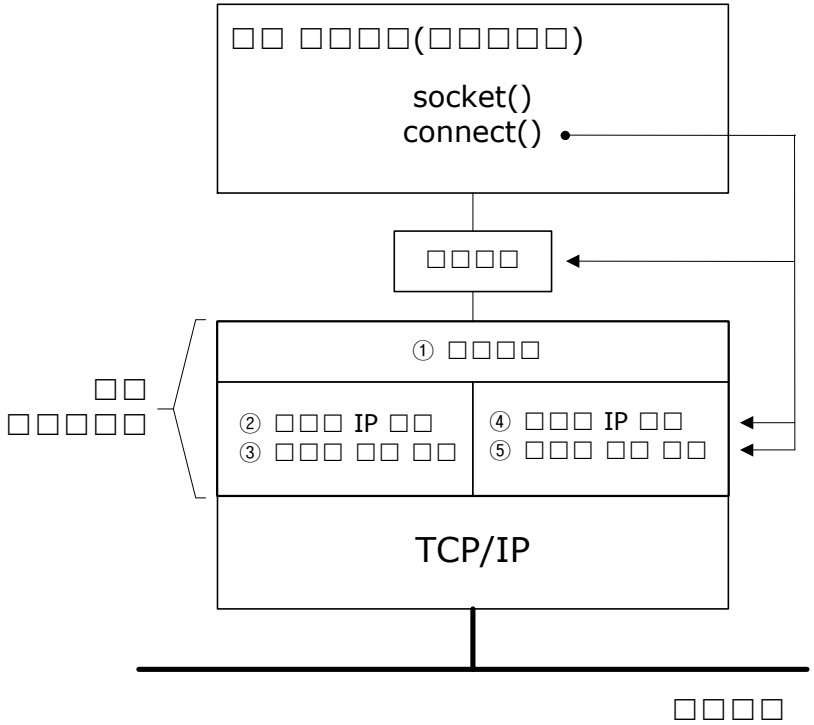
- TCP, UDP **소켓을 선택**
  - TCP : SOCK_STREAM
  - UDP : SOCK_DGRAM
- int socket(int domain, int type, int protocol);
- socket() **호출 시 소켓번호 리턴 과정**



connect(), 서버에 연결 요청

- connect() : **서버와 연결되면 0을 반환**

```
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	**전송 데이타의 크기**
	flags	**보통** 0
int write(int s, const void* buf, int length);	s	**소켓번호**
	buf	**전송할 데이터가 저장된 버퍼**
	length	**전송 데이타의 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 옵션**(SO_LINGER)**을 변경하면 미전송 데이터를 모두 버리고 종료**

TCP 클라이언트 예제 프로그램 : mydaytime.c

- **주요 코드**

```
int s, nbyte;
struct sockaddr_in servaddr;                                  // 서버 소켓 구조체
char buf[ MAXLINE+1 ];
s = socket(PF_INET, SOCK_STREAM, 0);                          // 연결형 소켓 개설
bzero((char *)servaddr, sizeof(servaddr))                     // 서버 소켓 구조체 초기화
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
If(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr)) <0{ // 연결요청
    perror("connect fail");
    exit(1);
}
// 서버가 보내오는 daytime 데이터의 수신 및 화면출력
if((nbyte=read(s, buf, MAXLINE )) < 0) {
      perror("read fail");
      exit(1);
}
buf[nbyte] = 0;
printf("%s", buf);
close(s);
return 0;
```

컴파일 : gcc -o mydaytime mydaytime
수행 : mydaytime 211.221.225.175
결과 : Wed Feb 26 11:00:21 2020

TCP 클라이언트 예제 프로그램 : tcp_echocli.c

- **메세지를 보내면 받은 메시지를 출력**
- **주요 코드**

```
int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr;
    int  s, nbyte;
    char buf[MAXLINE+1];
    if((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }
    // 에코 서버의 소켓주소 구조체 작성
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    servaddr.sin_port = htons(7); // echo 서비스 port


    // 연결요청
    if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("connect fail");
            exit(0);
    }
```

TCP 클라이언트 예제 프로그램 : tcp_echocli.c

- **코드(계속)**

```
    printf("입력: ");
    if (fgets(buf, sizeof(buf), stdin) == NULL) // echo 메시지 입력
        exit(0);
    nbyte = strlen(buf);
    // 에코 서버로 메시지 송신
    if (write(s, buf, nbyte) < 0) {
        printf("write error\n");
        exit(0);
    }
    // 수신된 에코 데이터 화면출력
    printf("수신 : ");
    if( (nbyte=read(s, buf, MAXLINE)) <0) {
        perror("read fail");
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s", buf);

    close(s);
    return 0;
}
```

실행 및 결과

- **실행** : tcp_echocli 127.0.0.1
- **결과**
  - **입력** : abcdefg
  - **수신** : abcdefg

TCP 클라이언트 예제 프로그램 : port_number.c

- **클라이언트에서는 포트 번호를 시스템이 임의로 배정**
  - TCP **소켓** : connect() **호출 직전에 배정**
  - UDP **소켓** : **첫 번째** sendto() **함수 호출 직전에 배정**

- TCP**와** UDP **클라이언트 포트 번호 배정을 확인하는 프로그램**
  - getsockname() **함수를 사용하여 시스템이 배정한 포트번호를 구함**
    - getsockname**은 자신의 호스트의 소켓 정보를 제공**
  - getpeername()**은 상대방의 소켓 정보를 제공**

- **실행 및 결과**
  - **실행** : port_number
  - **결과**
    스트림 소켓 포트번호 = 58895
    데이터그램 소켓 포트번호 = 32771

TCP 클라이언트 예제 프로그램 : port_number.c

```
#define MSG "Test Message"

int main() {
    int sd1, sd2 ;
    int addrlen;
    struct sockaddr_in servaddr, cliaddr;  // 소켓주소 구조체
    unsigned short port1, port2;   // 포트번호

    // 소켓주소 구조체 초기화
    servaddr.sin_family = AF_INET ;
    //INADDR_ANY는 자동으로 자신의 주소를 find
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY) ;
    servaddr.sin_port = htons(7);

    sd1=socket(PF_INET, SOCK_STREAM, 0);
    sd2=socket(PF_INET, SOCK_DGRAM, 0);

    // TCP 소켓의 포트번호 얻기
    if( connect(sd1, (struct sockaddr*)&servaddr, sizeof(servaddr))<0) {
        perror("connect fail");
        exit(1);
    }
```

TCP 클라이언트 예제 프로그램 : port_number.c

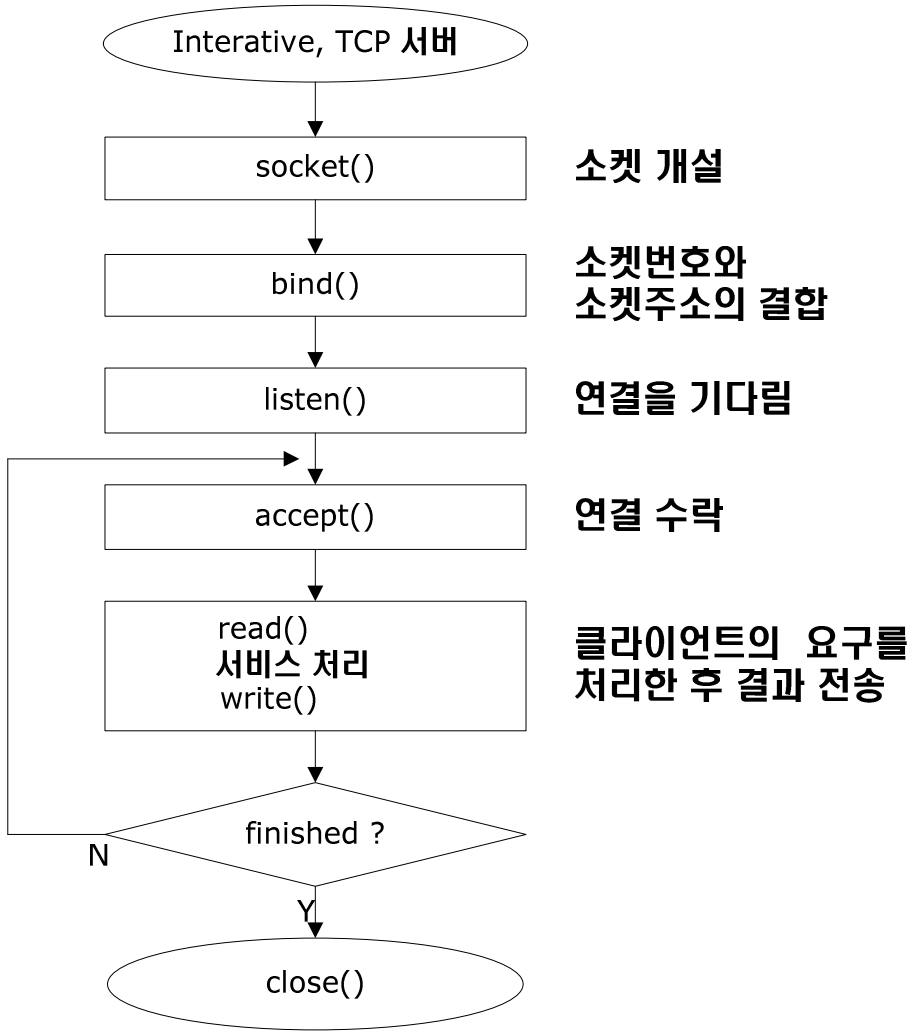
```
    getsockname(sd1, (struct sockaddr*)&cliaddr, &addrlen) ;
    port1 = ntohs(cliaddr.sin_port);
    // UDP 소켓의 포트번호 얻기
    sendto(sd2, MSG, strlen(MSG), 0, (struct sockaddr*)&servaddr, sizeof(servaddr));
    getsockname(sd2, (struct sockaddr*)&cliaddr, &addrlen) ;
    port2 = ntohs(cliaddr.sin_port);

    printf("스트림     소켓 포트번호 = %d\n", port1) ;
    printf("데이터그램 소켓 포트번호 = %d\n", port2) ;
    close(sd1) ;
    close(sd2) ;
    return 0;
}
```

- **실행 및 결과**
  - **실행** : port_number
  - **결과**
    스트림　　소켓 포트번호 = 58895
    데이터그램 소켓 포트번호 = 32771

2.4 TCP **서버 프로그램**

- **서비스 요청을 들어오는 순서로 처리하는**(iterative) TCP **서버프로그램 작성 절차**

TCP 서버 프로그램 작성 절차



socket(), 소켓 생성

- **클라이언트와 통신하기 위해 소켓을 생성해야 함**
  - **연결형 소켓** : SOCK_STREAM
  - **비연결형 소켓** : SOCK_DGRAM
- **연결형 소켓 개설** : socket(PF_INET, SOCK_STREAM, 0);
- **비연결형 소켓 개설** : socket(PF_INET, SOCK_DGRAM, 0);

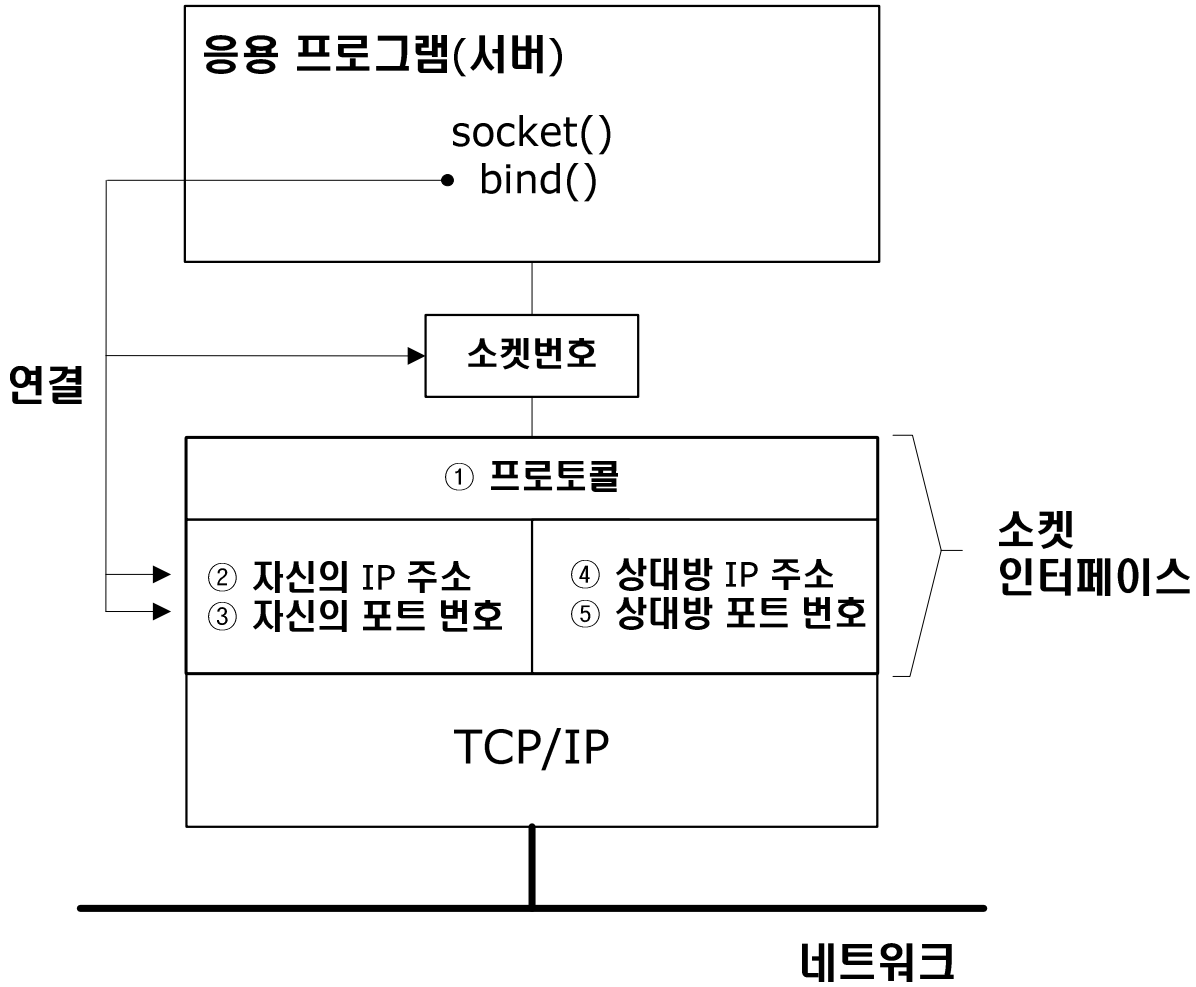
bind()

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

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

```
s = socket(PF_INET, SOCK_STREAM, 0);
struct sockaddr_in servaddr;
servaddr.sin_family = AF_INET;
// multihomed host일 경우 임의의 IP 주소로 오는 모든 데이터그램의 수신을 의미
//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 소켓**

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

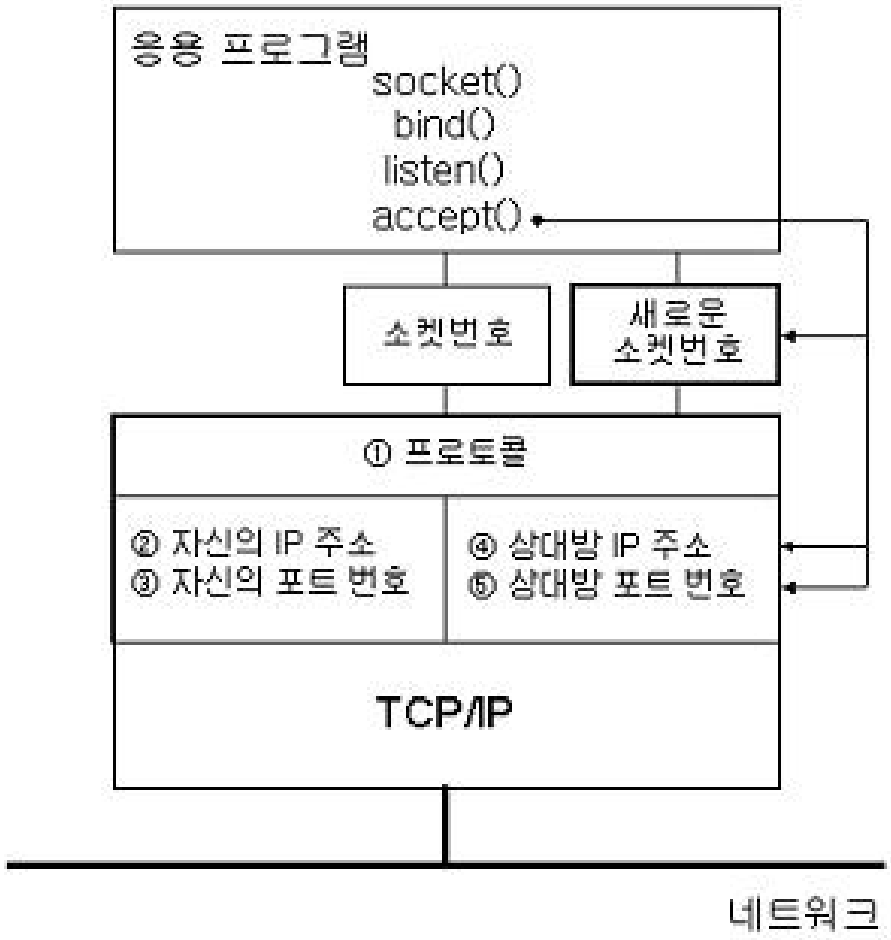
accept()

- **클라이언트와 설정된 연결을 실제로 받아들이기 위해 사용**

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

- **반환 값**
  - **성공 : 접속된 클라이언트와의 통신에 사용할 새로운 소켓 번호를 반환**
  - **실패 : -1 반환**
- **클라이언트가 connect()를 호출시 TCP의 3-way 핸드쉐이크 동작**
  - **클라이언트 : 서버로 SYN(X)를 전송**
  - **서버 : 클라이언트로 ACK(X+1), SYN(Y)를 전송**
  - **클라이언트 : 서버로 ACK(Y+1)을 전송**

accept() 호출시 얻는 정보



TCP 에코 서버프로그램 : tcp_echoserv.c

- **주요 코드**(tcp_echocli.c**에 대응하는 서버프로그램**)

```
struct sockaddr_in servaddr, cliaddr;
int listen_sock, accp_sock,   // 소켓번호
    addrlen=sizeof(cliaddr),    // 주소구조체 길이
    nbyte;
char buf[MAXLINE+1];  // 수신 버퍼

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

bzero((char *)&servaddr, sizeof(servaddr)); // servaddr을 '\0'으로 초기화
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr(htonl(INADDR_ANY);
servaddr.sin_port = htons(atoi(argv[1]));

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

// 소켓을 수동 대기모드로 세팅
listen(listen_sock, 5);
```

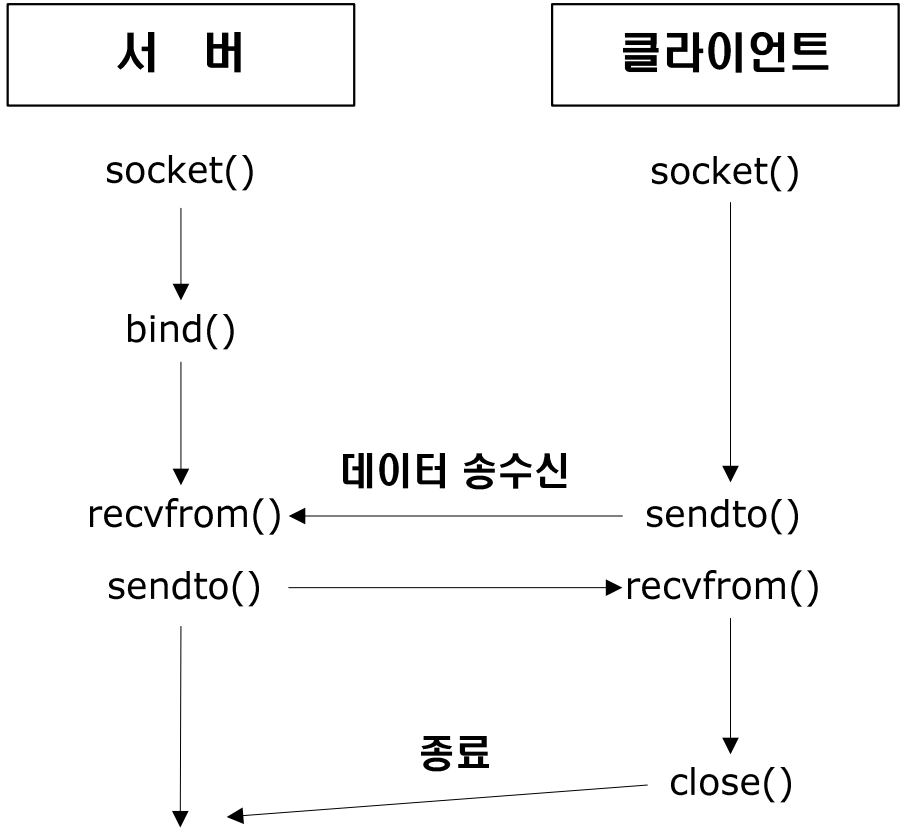
TCP 에코 서버프로그램 : tcp_echoserv.c

- **주요 코드**

```
// iterative  에코 서비스 수행
while(1) {
    puts("서버가 연결요청을 기다림..");
    // 연결요청을 기다림
    accp_sock = accept(listen_sock, (struct sockaddr *)&cliaddr, &addrlen);
    if(accp_sock < 0) {
            perror("accept fail");
            exit(0);
    }
    puts("클라이언트가 연결됨..");
    nbyte = read(accp_sock, buf, MAXLINE);
    write(accp_sock, buf, nbyte);
    close(accp_sock);
}
close(listen_sock);
return 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	**전송 데이타의 크기**
	flags	**보통** 0
	to	**목적지의 소켓주소 구조체**
	tolen	to **의 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 **의 길이**

udp 에코 클라이언트 프로그램 : udp_echocli.c

- **주요 코드**

```
struct sockaddr_in servaddr;
. . .
socket(PF_INET, SOCK_DGRAM, 0);
// 키보드 입력을 받음
 printf("입력 : ");
 if (fgets(buf, MAXLINE , stdin)==NULL) {
     printf("fgets 실패");     exit(0);
 }
 // 에코 서버로 메시지 송신
if (sendto(s, buf, strlen(buf), 0, (struct sockaddr *)&servaddr, addrlen) < 0) {
     perror("sendto fail");
     exit(0);
 }
 // 수신된 에코 메시지
 if ( (nbyte=recvfrom(s, buf, MAXLINE, 0,
                          (struct sockaddr *)&servaddr, &addrlen)) < 0) {
     perror("recevfrom fail");     exit(0);
 }
 buf[ nbyte ]=0; //메시지의 끝에 '\0' 추가
 printf("%s\n", buf);
 close(s);
```

udp 에코 서버 프로그램 : udp_echoserv.c

- **주요코드**

```
if ((s = socket(PF_INET< SOCK_DGRAM, 0)) < 0) { perror("socket fail"); exit(0);}
bzero((char *)&servaddr,addrlen); // servaddr을 '\0'으로 초기화
 bzero((char *)&cliaddr, addrlen);   // cliaddr을 '\0'으로 초기화
 // servaddr 세팅
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servaddr.sin_port = htons(atoi(argv[1]));
 // bind() 호출
 if(bind(s, (struct sockaddr *)&servaddr, addrlen) < 0) {
     perror("bind fail");       exit(0);
 }
 // iterative  에코 서비스 수행
 while(1) {
     nbyte = recvfrom(s, buf, MAXLINE, 0, (struct sockaddr *)&cliaddr, &addrlen);
     if(nbyte< 0) {  perror("recvfrom fail");  exit(1);   }
     buf[nbyte] = 0;
     printf("%d byte recv: %s\n",nbyte, buf);
     if (sendto(s, buf, nbyte, 0, (struct sockaddr *)&cliaddr, addrlen) < 0) {
           perror("sendto fail"); exit(1); }
     }
     puts("sendto complete");
 }
```