Chapter 03

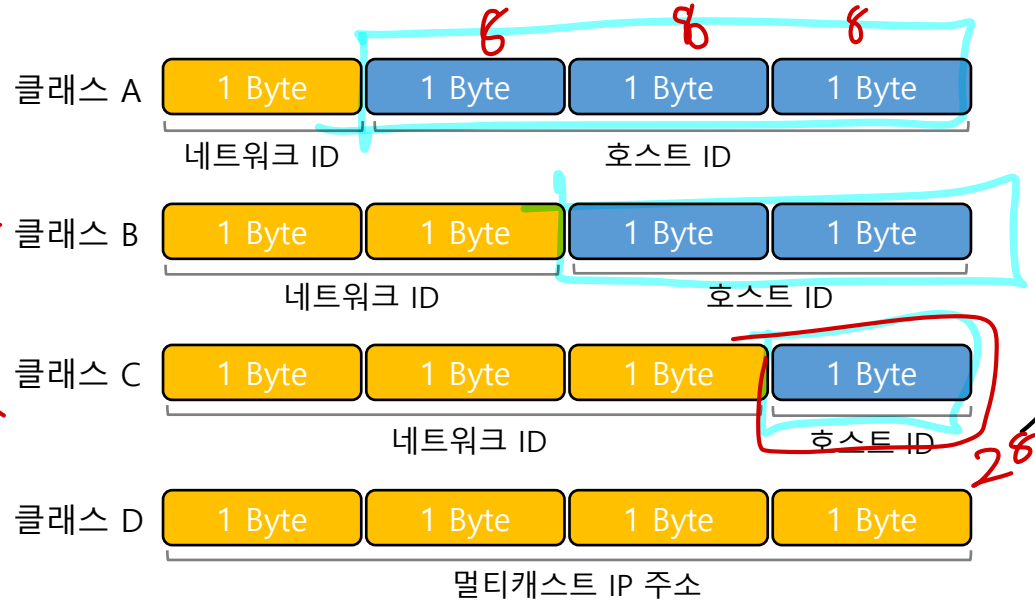
주소체계와 데이터 정렬

인터넷 주소

IP 주소 — 컴퓨터를 구별할 역할

- 인터넷 주소란? + 장비
  - 인터넷상에서 컴퓨터를 구분하는 목적으로 사용되는 주소
  - 4바이트 주소체계인 IPv4와 16바이트 주소체계인 IPv6가 존재
  - 네트워크 주소와 호스트 주소로 나뉨
  - 네트워크 주소를 이용해서 네트워크를 찾고, 호스트 주소를 이용해서 해당 네트워크에서 호스트를 검색함 (subnet mask)

같은 네트워크 구별 !!

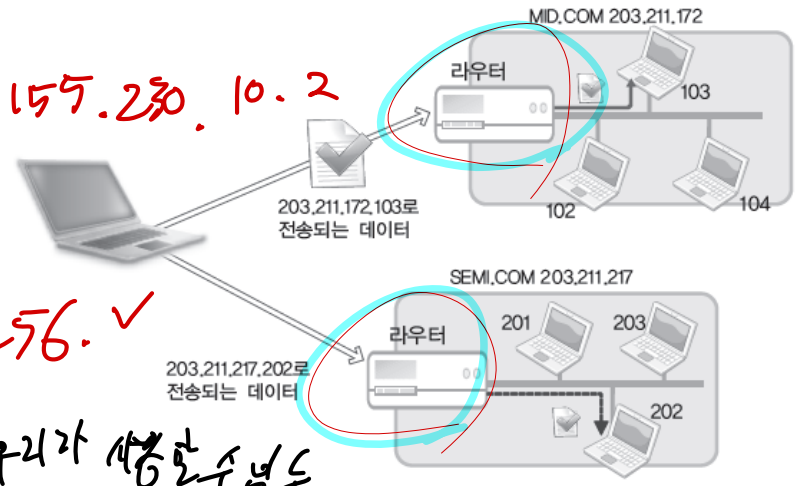


IPv4 인터넷 주소의 체계

2^{24} ✓

$2^8 = 256$ ✓

↓ 우리가 생각하는 IP 클래스



인터넷 주소의 역할

IP : 155. 230. 120. 100

subnet mask 255. 255 252. 0 bit 연산

서브넷 크기 결정

클래스 별 네트워크 주소와 호스트 주소의 경계

- 클래스별 주소 범위

클래스	시작 주소	끝 주소	첫 번째 바이트 2진수 표현
클래스 A	0.0.0.0	127.255.255.255	0000 0000 ~ 0111 111
클래스 B	128.0.0.0	191.255.255.255	1000 0000 ~ 1011 1111
클래스 C	192.0.0.0	223.255.255.255	1100 0000 ~ 1101 1111
클래스 D	224.0.0.0	239.255.255.255	1110 0000 ~ 1110 1111
클래스 E	240.0.0.0	255.255.255.255	1111 0000 ~ 1111 1111

  - 첫 번째 바이트 정보만 참조해도 IP 주소의 클래스 구분이 가능함
  - 네트워크 주소와 호스트 주소의 경계를 구분

- 특수 범위 주소

주소	시작 주소	끝 주소
local loopback	127.0.0.0	127.255.255.255
사설망	192.168.0.0	192.168.255.255
멀티캐스트 주소 (클래스 D)	224.0.0.0	239.255.255.255

소켓의 구분에 활용되는 Port번호

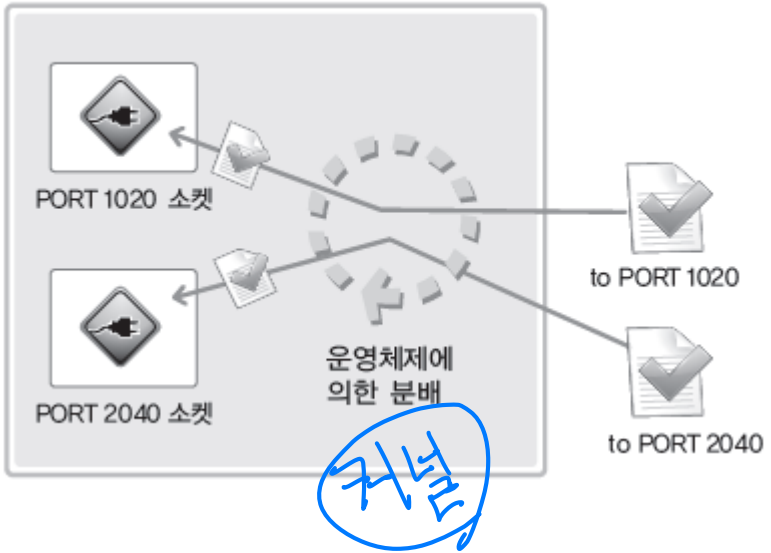
./server 9190

./client 서버IP주소 9190

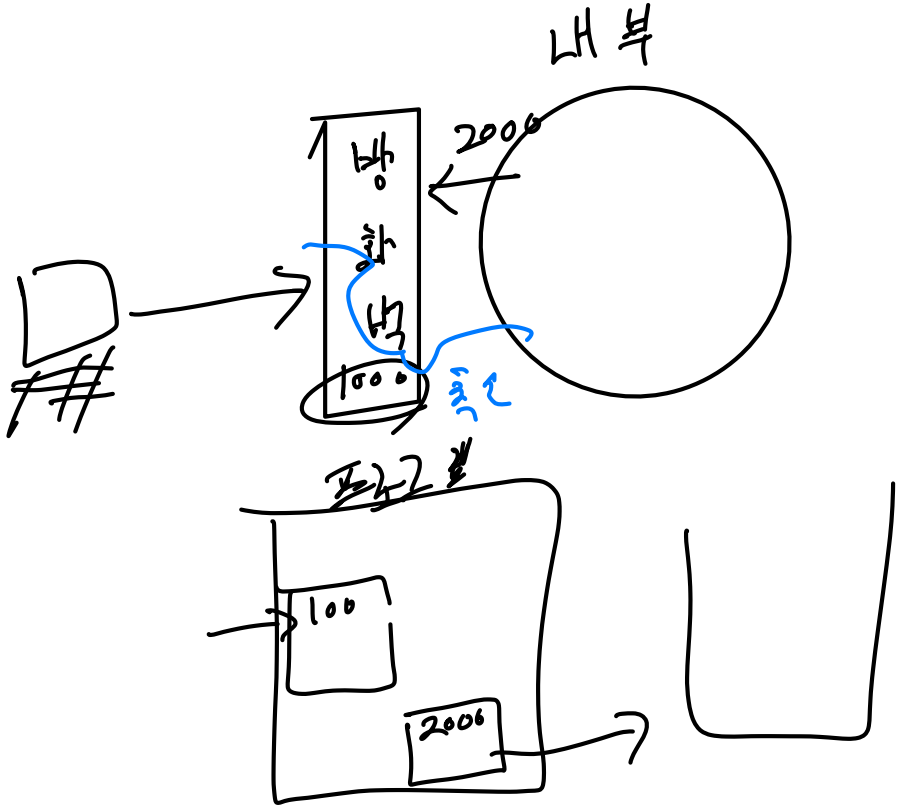
127.0.0.1

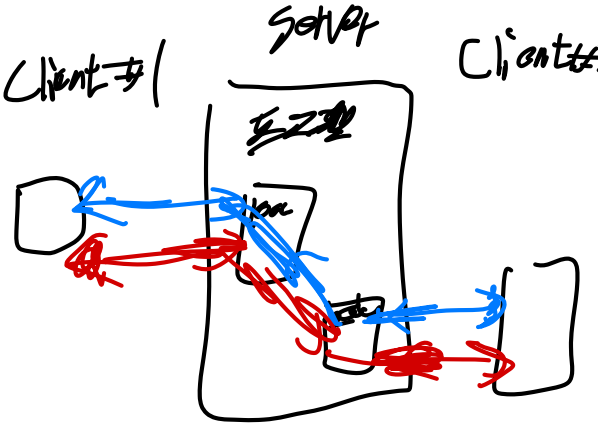
포트번호

- Port번호
  - IP 주소: 컴퓨터를 구분하는 용도로 사용
  - Port번호: 소켓을 구분하는 용도로 사용 : 소켓을 사용한 응용프로그램을 구분.
  - 하나의 프로그램 내에서는 둘 이상의 소켓이 존재할 수 있으므로,
    - 둘 이상의 Port가 하나의 프로그램에 의해 할당될 수 있음
  - Port번호는 16비트로 표현, 따라서 그 값은 0 ~ 65535 이하 (2^{16})
  - 0 ~ 1023은 잘 알려진 PORT(Well-known PORT)라 해서 이미 용도가 결정되어 있음
    - Well-known Port: https://ko.wikipedia.org/wiki/TCP/UDP의_포트_목록



PORT번호에 의한 소켓의 구분과정





port forwarding

Well-known port (0~1023) 예

포트번호	TCP	UDP	설명
20	TCP		FTP (파일 전송 프로토콜)
21	TCP		FTP 제어 포트
22	TCP		SSH (secure shell)- ssh, scp, sftp
23	TCP		Telnet protocol – 암호화하지 않은 텍스트 통신
80	TCP	UDP	HTTP (HyperText Transfer Protocol)
110	TCP		POP3 (Post Office Protocol version 3)
123		UDP	NTP (Network Time Protocol) – 시간 동기화
161		UDP	SNMP (Simple Network Management Protocol)

Port Scan

- nmap (Network Mapper)
  - Linux 설치: $ sudo apt install nmap
  - Mac 설치: $ brew install nmap
  - 포트 스캔, 호스트 탐지

```
$ nmap localhost
Starting Nmap 7.80 ( https://nmap.org ) at 2022-09-13 16:47
KST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00088s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
631/tcp open  ipp
```

```
iMac:~$  % nmap localhost
Starting Nmap 7.93 ( https://nmap.org ) at 2023-03-10
11:18 KST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000040s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
5000/tcp  open  upnp
7000/tcp  open  afs3-fileserver
49152/tcp open  unknown


Nmap done: 1 IP address (1 host up) scanned in 0.10
seconds
```

IPv4 기반의 주소표현을 위한 구조체

- IPv4 주소 표현 구조체
  - IP 주소와 Port 번호는 sockaddr_in 구조체 변수를 이용하여 표현

```
struct sockaddr_in
{
    sa_family_t     sin_family;
    uint16_t        sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
};
```

- 주소 체계
- Port 번호
- 32비트 IP 주소
- 사용되지 않음

0으로 초기화

```
struct in_addr
{
    in_addr_t s_addr;
};
```

sin_addr.s_addr = 0

64bit 주소 길이

32bit

16bit

8bit

자료형 이름	자료형에 담길 정보
int8_t	• signed 8-bit int
uint8_t	• unsigned 8-bit int (unsigned char)
int16_t	• unsigned 16-bit int
uint16_t	• unsigned 16-bit int (unsigned short)
int32_t	• signed 32-bit int
uint32_t	• unsigned 32-bit int (unsigned long)
sa_family_t	• 주소 체계 (address family): unsigned short
socklen_t	• 길이 정보: unsigned int
in_addr_t	• IP 주소 정보, uint32_t로 정의
in_port_t	• Port 번호 정보, uint16_t로 정의

CPU 발전방향

구조체 sockaddr_in의 멤버에 대한 분석

- sockaddr_in 구조체 멤버
  - sin_family:
    - 주소체계 저장
  - sin_port
    - 16비트 Port 번호 저장
    - 네트워크 바이트 순서로 저장
  - sin_addr
    - 32비트 IP주소 저장
    - 네트워크 바이트 순서로 저장
    - 멤버 sin_addr의 구조체 자료형 in_addr: 32비트 정수형
  - sin_zero[8]
    - 특별한 의미를 지니지 않는 멤버
    - sockaddr 구조체와 크기를 맞추기 위해 사용
    - 반드시 0으로 초기화

이름	주소 체계(Address Family)
AF_INET	• IPv4 인터넷 프로토콜 체계
AF_INET6	• IPv6 인터넷 프로토콜 체계
AF_LOCAL	• 로컬 통신을 위한 UNIX 프로토콜 체계

```
struct sockaddr
{
    sa_family_t sa_family;   // address family(2 bytes)
    char sa_data[14];    // IP address + Port number(14 bytes)
};
```

```
struct sockaddr_in addr;
        . . .
memset(&addr, 0, sizeof(addr));
```

구조체 sockaddr_in의 활용의 예

- sockaddr_in: bind() 함수의 인자로 전달
  - bind() 함수의 매개변수 타입이 sockaddr이므로 형변환이 필요

```
struct sockaddr_in serv_addr;
    . . .
if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))==-1)
    error_handling("bind() error");
```

type casting

- 구조체 sockaddr
  - 다양한 주소 체계의 주소 정보를 저장할 수 있도록 정의되어 있음
  - IPv4의 주소 정보를 저장하기 불편
    - 이에 동일한 바이크 크기를 가지는 구조체 sockaddr_in을 정의
    - 구조체 sockaddr_in을 사용하여 쉽게 IPv4의 주소 정보를 저장함

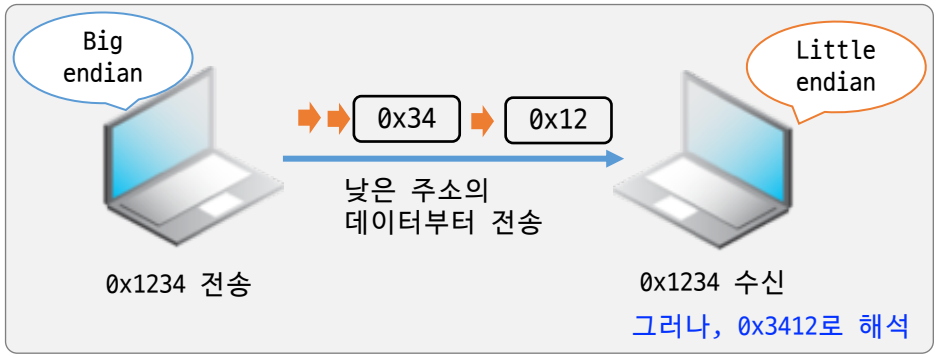
```
struct sockaddr
{
    sa_family_t sa_family;   // address family (2 bytes)
    char sa_data[14];    // IP address + Port number (14 bytes)
};
```

바이트 순서(Order)와 네트워크 바이트 순서

- 빅 엔디안(Big Endian)
  - 상위 바이트의 값을 작은 번지수에 저장
    - 상위 바이트의 값을 앞에 위치

- 리틀 엔디안(Little Endian)
  - 하위 바이트의 값이 작은 번지수에 저장
    - 상위 바이트의 값을 큰 번지수에 저장
    - Intel, Apple silicon

- 네트워크 바이트 순서 (Network Byte Order)
  - 통일된 데이터 송수신 기준을 의미
  - 빅 엔디언 사용
  - 데이터 송수신 과정
    - 바이트 단위로 데이터를 전송
    - 바이트 변환 과정이 필요 없음



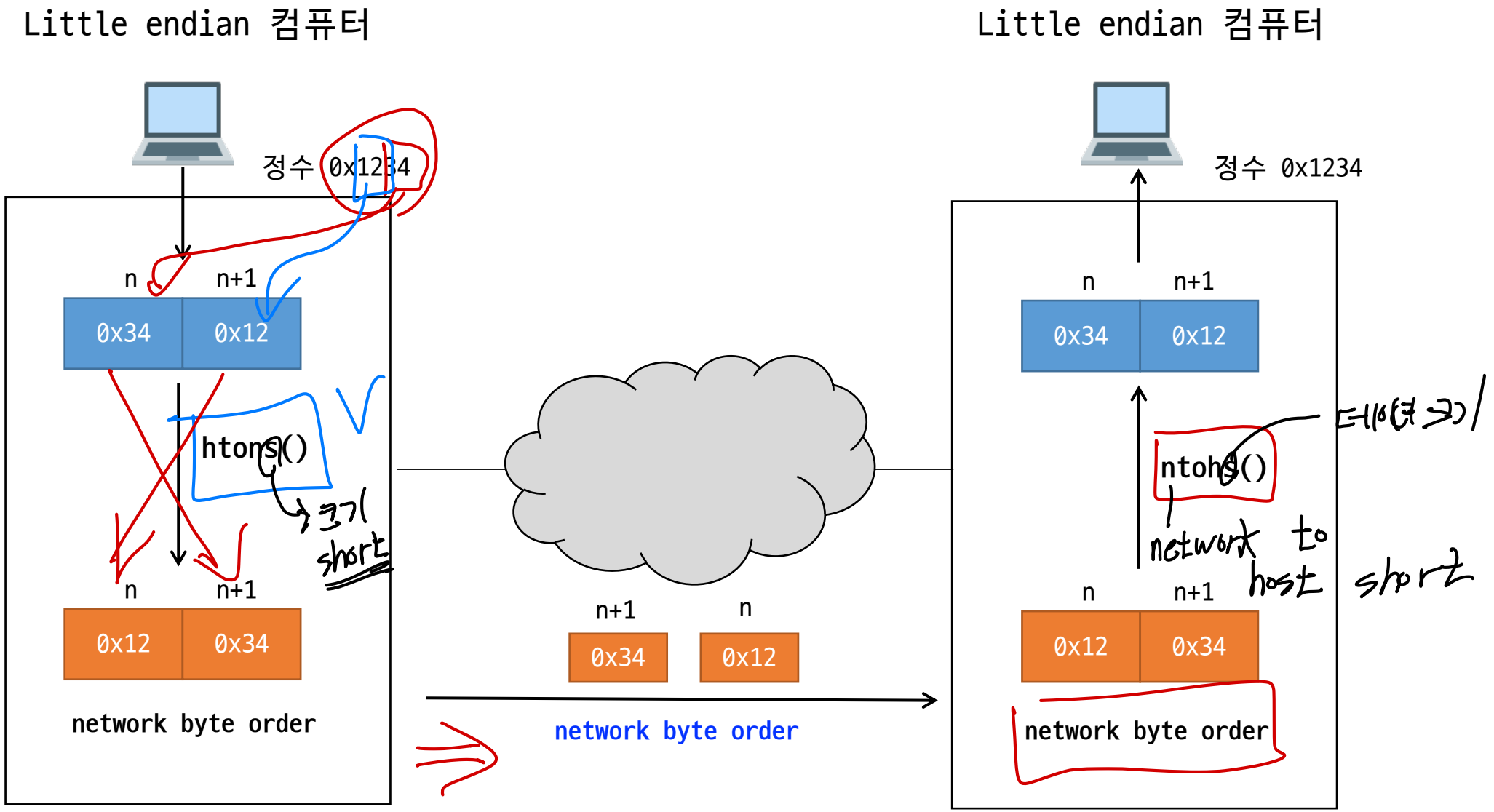
정수: 0x12345678

Big endian 메모리 주소	0x100	0x101	0x102	0x103
저장값	0x12	0x34	0x56	0x78

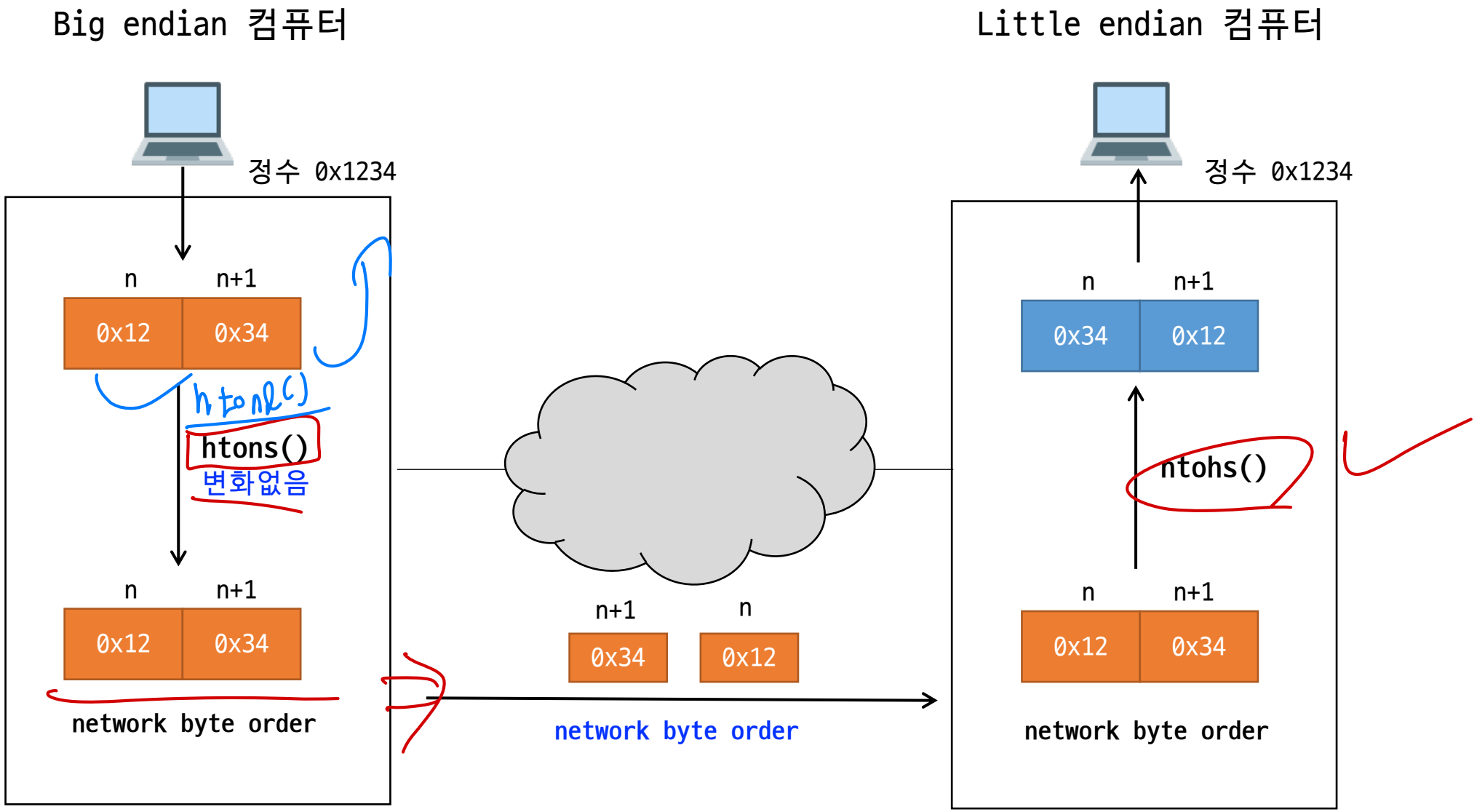
Little endian 메모리 주소	0x100	0x101	0x102	0x103
저장값	0x78	0x56	0x34	0x12

ARM : 퀄컴 저전력

Little endian -> Little endian 통신



Big endian -> Little endian 통신



바이트 순서의 변환 (Endian Conversions)

- 바이트 순서 변환 함수

```
• unsigned short htons(unsigned short);
• unsigned short ntohs(unsigned short);
• unsigned long htonl(unsigned long);
• unsigned long ntohl(unsigned long);
```

➢ 바이트 순서 변환은 sockaddr_in 구조체 변수에 주소 정보를 저장할 때만 사용하면 됨

- htons에서
  - h는 호스트(host) 바이트 순서를 의미
  - n은 네트워크(network) 바이트 순서를 의미
  - s는 자료형 short를 의미
- htonl에서 l은 자료형 long을 의미

htonl() 소스 코드

- Big Endian System 사용자인 경우, htonl()을 호출해야 되는가?

```
#include <stdint.h>
#include <netinet/in.h>

uint32_t htonl(uint32_t x)
{
#if BYTE_ORDER == BIG_ENDIAN
    return x;
#elif BYTE_ORDER == LITTLE_ENDIAN
    return __bswap_32 (x);
#else
# error "What kind of system is this?"
#endif
}
```

Big Endian인 경우,
원래의 값(x)를 그대로 리턴함

바이트 변환의 예 (endian_conv.c)

```c
#include <stdio.h>
#include <arpa/inet.h>
typedef unsigned short u16;
typedef unsigned long u32;

int main(int argc, char *argv[])
{
    u16 host_port=0x1234;
    u16 net_port;
    u32 host_addr=0x12345678;
    u32 net_addr;

    net_port=htons(host_port);
    net_addr=htonl(host_addr);

    printf("Host ordered port: %#x \n", host_port);
    printf("Network ordered port: %#x \n", net_port);
    printf("Host ordered address: %#lx \n", host_addr);
    printf("Network ordered address: %#lx \n", net_addr);
    return 0;
}
```

이런 식으로 사용해 보고 해봐!!

host ⇒ network

htons()

htonl()

```
$ gcc endian_conv.c -o endian_conv
$ ./endian_conv

Host ordered port: 0x1234
Network ordered port: 0x3412
Host ordered address: 0x12345678
Network ordered address: 0x78563412
```

inet_addr() 함수

- inet_addr(const char *string)
  - "211.214.107.99"와 같이 10진수(dotted decimal)형태로 표시된 문자열을
    - 네트워크 바이트 순서의 32비트 정수형으로 반환

```
#include <arpa/inet.h>

in_addr_t inet_addr(const char* string);
-> 성공 시 빅 엔디안으로 변환된 32비트 정수값, 실패 시 INADDR_NONE 반환
```

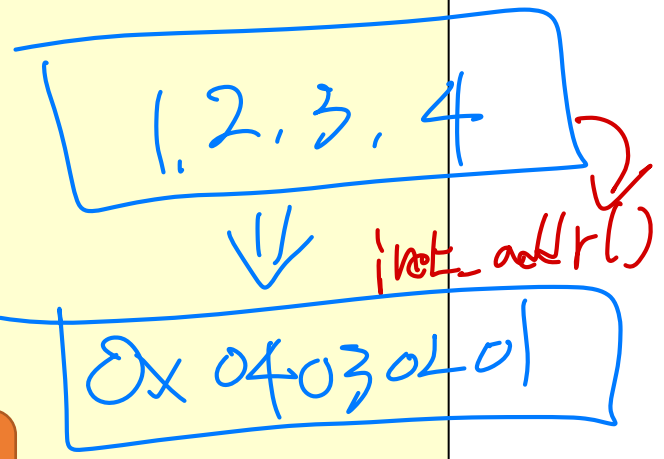
  - in_addr_t 타입: uint32_t 형태(32비트)
    - /usr/include/netinet/in.h 파일

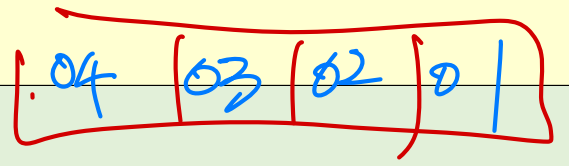
```
typedef uint32_t in_addr_t;
struct in_addr
{
    in_addr_t s_addr;
};
```

inet_addr() 예제

```c
#include <stdio.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    char *addr1 = "1.2.3.4";
    char *addr2 = "1.2.3.256";
    uint32_t conv_addr;

    conv_addr = inet_addr(addr1);
    if(conv_addr==INADDR_NONE)
        printf("Error occured! \n");
    else
        printf("Network ordered integer addr: %#x \n", conv_addr);

    conv_addr = inet_addr(addr2);
    if(conv_addr==INADDR_NONE)
        printf("Error occureded \n");
    else
        printf("Network ordered integer addr: %#x \n\n", conv_addr);

    return 0;
}
```

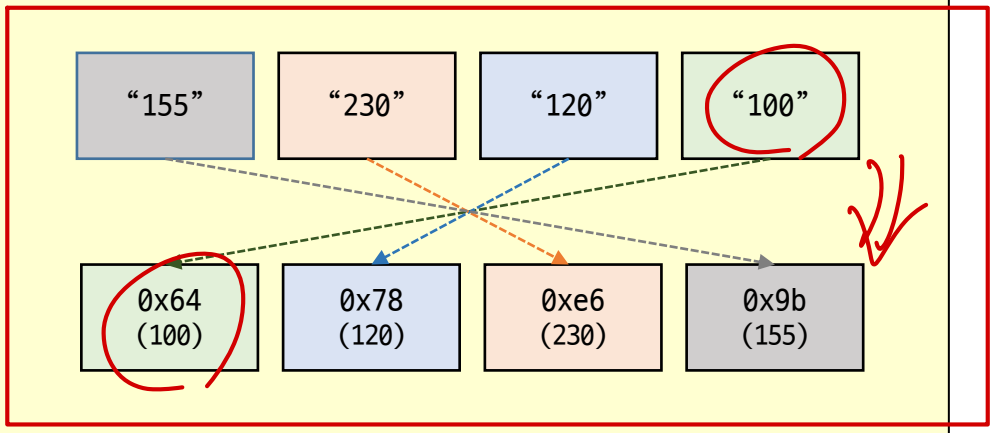
```
$ ./inet_addr
Network ordered integer addr: 0x4030201
Error occureded
```





inet_addr.c

```c
#include <stdio.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    char *addr1 = "155.230.120.100";
    char *addr2 = "192.168.0.1";
    uint32_t conv_addr;

    conv_addr = inet_addr(addr1);
    if(conv_addr==INADDR_NONE)
        printf("Error occurred! \n");
    else
        printf("Network ordered integer addr: %#x \n", conv_addr);

    conv_addr = inet_addr(addr2);
    if(conv_addr==INADDR_NONE)
        printf("Error occurred! \n");
    else
        printf("Network ordered integer addr: %#x \n\n", conv_addr);

    return 0;
}
```




```
$ ./inet_addr
Network ordered integer addr: 0x6478e69b
Network ordered integer addr: 0x100a8c0
```

inet_aton (문자열 형태 -> in_addr 구조체로 변환)

- inet_aton() 함수
  - inet_addr() 함수와 동일 기능
  - in_addr 구조체 변수에 변환 결과가 저장되는 점이 inet_addr()과 다름

```
#include <arpa/inet.h>

int inet_aton(const char *string, struct in_addr *addr);
-> 성공 시 1(true), 실패 시 0(false) 반환
```

변환 및 결과 저장

  - string: 변환할 IP 주소 정보를 담고 있는 문자열
  - addr: 변환될 정보를 저장할 in_addr 구조체 변수의 주소값 전달

- inet_aton() 사용 예

```
char *addr="127.232.124.79";
struct sockaddr_in addr_inet;

inet_aton(addr, &addr_inet.sin_addr))
printf("Network ordered integer addr: %#x \n", addr_inet.sin_addr.s_addr);
```

inet_aton.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
void error_handling(char *message);

int main(int argc, char *argv[])
{
    char *addr="127.232.124.79";
    struct sockaddr_in addr_inet;

    if(!inet_aton(addr, &addr_inet.sin_addr))
        error_handling("Conversion error");
    else
        printf("Network ordered integer addr: %#x \n", addr_inet.sin_addr.s_addr);
    return 0;
}

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

struct in_addr을 직접 사용 가능

```c
struct in_addr addr_inet;

if(!inet_aton(addr, &addr_inet))
    error_handling("Conversion error");
else
    printf("Network ordered integer addr: %#x\n",
    addr_inet.s_addr);
```

```
$ ./inet_aton
Network ordered integer addr: 0x4f7ce87f
```

inet_ntoa (정수 형태 -> 문자열 형태로 변환)

- inet_ntoa() 함수
  - 네트워크 바이트 순서로 정렬된 정수형을 문자열 형태로 변환
  - inet_aton() 함수의 반대 기능

```
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr addr);
-> 성공 시 변환된 문자열의 주소값, 실패 시 -1 반환
```

```
struct sockaddr_in addr1, addr2;
char *str_ptr;
char str_arr[20];

addr1.sin_addr.s_addr = htonl(0x1020304);
addr2.sin_addr.s_addr = htonl(0x1010101);

str_ptr = inet_ntoa(addr1.sin_addr);
strcpy(str_arr, str_ptr);
printf("Dotted-Decimal notation1: %s \n", str_ptr);

inet_ntoa(addr2.sin_addr);
printf("Dotted-Decimal notation2: %s \n", str_ptr);
```

왜 바뀔까요?

```
$ gcc inet_ntoa.c -o ntoa
$ ./ntoa
Dotted-Decimal notation1: 1.2.3.4
Dotted-Decimal notation2: 1.1.1.1
```

inet_ntoa() 함수 분석

- inet_ntoa() 함수
  - 내부적으로 static 변수(char buffer[18])에 변환된 주소 값이 저장
    - 프로그램이 종료될 때까지 buffer 변수의 메모리는 유지됨
  - 새로운 주소를 이용하여 inet_ntoa()함수를 다시 호출할 경우,
    - buffer가 가리키는 메모리(주소)의 값은 변경됨
    - 기존에 buffer를 가리키고 있던 포인터 변수 *str_ptr의 값이 변경됨
      - 따라서 두 번째 inet_ntoa() 함수 호출시에는 리턴값을 받는 변수 사용 안함
  - 전역 static 변수
    - 해당 소스 파일 내부에서만 사용 가능하고 다른 파일에서 사용 못함
    - extern 사용 안됨

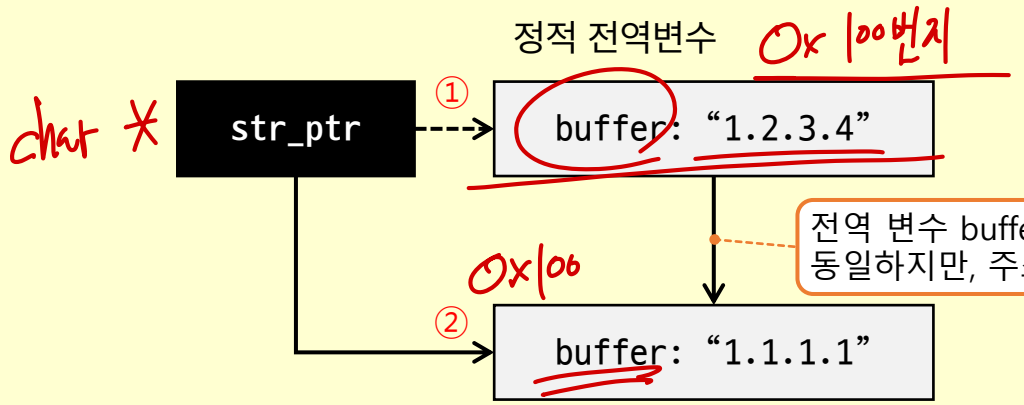
https://github.com/lattera/glibc/blob/master/inet/inet_ntoa.c

```
static __thread char buffer[18];

char *inet_ntoa (struct in_addr in)
{
    unsigned char *bytes = (unsigned char *) &in;
    __snprintf (buffer, sizeof(buffer), "%d.%d.%d.%d",
                bytes[0], bytes[1], bytes[2], bytes[3]);
    return buffer;
}
```

__thread 변수:
스레드마다 독립적인 변수 선언에 사용

inet_ntoa.c

```c
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    struct sockaddr_in addr1, addr2;
    char *str_ptr;
    char str_arr[20];

    addr1.sin_addr.s_addr=htonl(0x1020304);
    addr2.sin_addr.s_addr=htonl(0x1010101);

①   str_ptr = inet_ntoa(addr1.sin_addr);
    strcpy(str_arr, str_ptr); // strcpy(dest, src)
    printf("Dotted-Decimal notation1: %s \n", str_ptr);

②   str_ptr = inet_ntoa(addr2.sin_addr);
    printf("Dotted-Decimal notation2: %s \n", str_ptr);
    return 0;
}
```



변환된 주소 정보를 다시 사용할 경우, 다른 변수에 복사해야 됨

명확하게 리턴값을 받는 변수를 지정할 것

```
$ ./inet_ntoa
Dotted Decimal notation1: 1.2.3.4
Dotted Decimal notation2: 1.1.1.1
```

인터넷 주소의 초기화

- 일반적인 인터넷 주소 초기화 과정

```
struct sockaddr_in addr;
char *serv_ip ="211.217.168.13";          // 서버의 IP 주소 문자열
char *serv_port = "9190";                 // Port 번호 문자열
memset(&addr, 0, sizeof(addr));           // 구조체의 모든 멤버를 0으로 초기화
addr.sin_family = AF_INET;                // 주소 체계 지정
addr.sin_addr.s_addr = inet_addr(serv_ip); // IP 주소를 정수형으로 변환 및 저장
addr.sin_port = htons(atoi(serv_port));   // Port 번호를 정수형으로 변환 및 저장
```

- 서버에서 주소정보를 설정하는 이유!

> "IP 211.217.168.13, PORT 9190으로 들어오는 데이터는 내게로 다 보내라!"

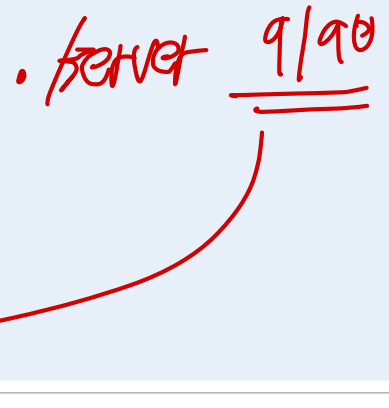
- 클라이언트에서 주소정보를 설정하는 이유!

> "IP 211.217.168.13, PORT 9190으로 연결을 해라!"

INADDR_ANY

- 서버 소켓의 주소 할당
  - 소켓이 동작하는 컴퓨터의 IP 주소가 자동으로 할당
  - 컴퓨터에 두 개 이상의 IP 주소를 할당 받아서 사용하는 경우
    - 어떤 주소를 통해 데이터가 들어오더라도 PORT 번호만 일치하면 수신함

```
struct sockaddr_in addr;

char *serv_port = "9190";
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(atoi(serv_port));
```



  - INADDR_ANY 상수
    - 현재 실행 중인 컴퓨터의 IP 주소를 소켓에 부여할 때 사용 (서버 프로그램 구현에 주로 사용)

```
/* Address to accept any incoming messages. */
#define INADDR_ANY ((in_addr_t) 0x00000000)
```

➢ /usr/include/netinet/in.h 파일에 정의되어 있음

Chapter 01의 예제 실행 방식

- 서버 실행 방식

```
$ ./hserver 9190
```

  - 서버의 리스닝 소켓 주소는 INADDR_ANY로 지정한 경우
  - 소켓의 PORT 번호만 인자로 전달함

- 클라이언트 실행 방식

```
$ ./hclient 127.0.0.1 9190
```

  - 연결할 서버의 IP 주소와 PORT 번호를 인자로 전달
  - 127.0.0.1은 loopback 주소로 실행하는 컴퓨터 자신을 의미
  - Loopback 주소를 사용하는 이유
    - ➢ 한 대의 컴퓨터에서 서버와 클라이언트를 실행시켰기 때문

소켓에 인터넷 주소 할당

- 소켓에 인터넷 주소 할당

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

int bind(int sockfd, struct sockaddr *myaddr, socklent_t addrlen);
-> 성공 시 0, 실패 시 -1 반환
```

↳ IP, Port 할당

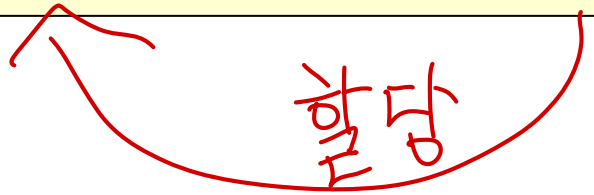
  - sockfd: 주소 정보를 할당할 소켓의 파일 디스크립터
  - myaddr: 할당할 주소 정보를 가지는 구조체 변수의 주소 값
  - addrlen: 두 번째 인자로 전달된 구조체 변수의 길이

- bind() 함수 호출이 성공하면
  - 첫 번째 인자에 해당하는 소켓에 두 번째 인자로 전달된 주소 정보가 할당됨

소켓에 인터넷 주소 할당 과정

```
int serv_sock;
struct sockaddr_in serv_addr;
char *serv_port = "9190";

/* 서버 소켓(리스닝 소켓) 생성 */
serv_sock = socket(PF_INET, SOCK_STREAM, 0);

/* 주소 정보 초기화 */
memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(serv_port));

/* 주소 정보 할당 */
bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
```

서버 프로그램의 일반적인 주소할당 과정

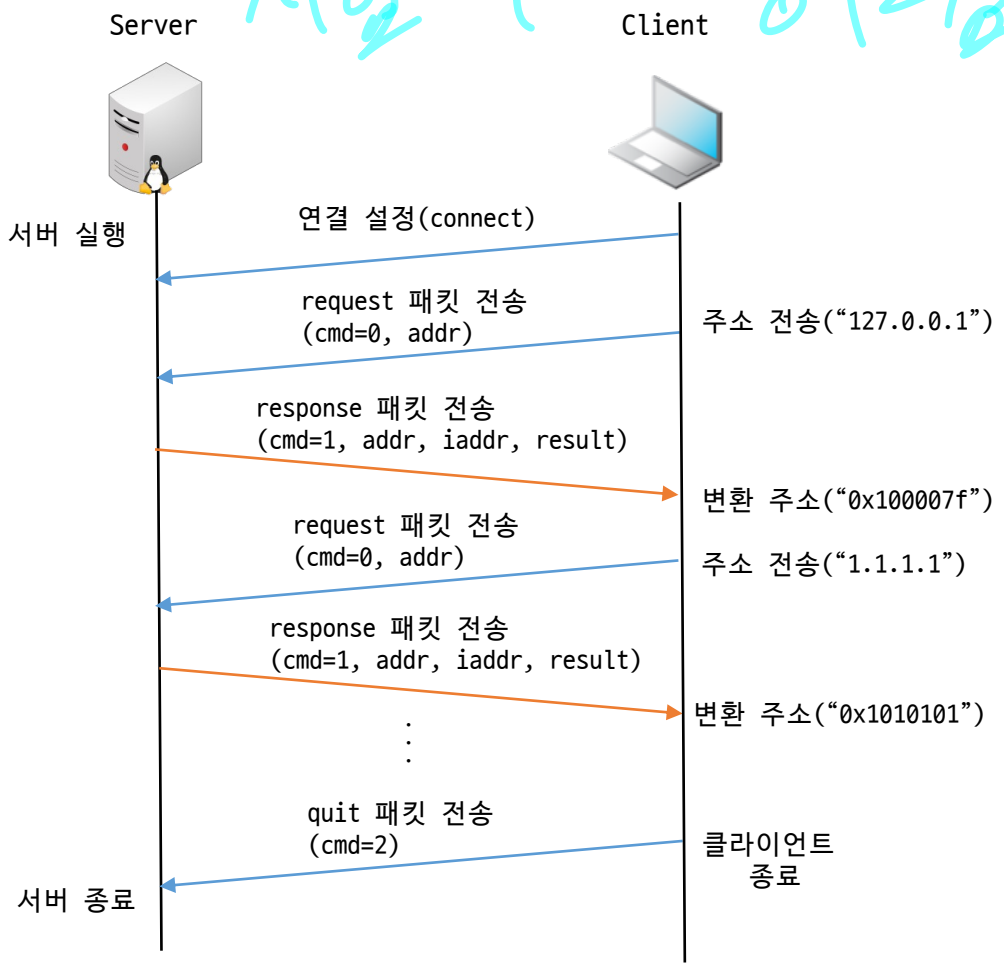


과제 프로토콜 구현 예제 #1

과제!! 시험 미리보기!!

```
#define BUF_SIZE 20

// result 필드값
#define ERROR 0
#define SUCCESS 1

// cmd 필드값
#define REQUEST 0
#define RESPONSE 1
#define QUIT 2

typedef struct {
    int cmd;              // 0: request, 1: response, 2: quit
    char addr[BUF_SIZE];  // dotted-decimal address(20 bytes)
    struct in_addr iaddr; // inet_aton()의 변환 주소 저장
    int result;           // 0: Error, 1: Success
}PACKET;
```

보고 같게 구현해라!!



과제 프로토콜 구현 예제 #2

- 서버 구현 예시

```
while(1) {
    rx_len = read(clnt_sock, &recv_packet, sizeof(PACKET));
    if(rx_len == 0)
        break;

    if(recv_packet.cmd == REQUEST) {
        printf("[Rx] Received Dotted-Decimal Address: %s\n", recv_packet.addr);

        if(inet_aton(recv_packet.addr, &iaddr) == 0) {
            // 주소 변환 실패
            write(. . .);
        }
        else {
            // 주소 변환 성공
             write(. . .);
        }
    }
    else if(recv_packet.cmd == QUIT) {
        . . .
        break;
    }
    else {
        printf("[Rx] Invalid command: %d\n", recv_packet.cmd);
        break;
    }
}
```

수신된 패킷의 cmd를 반드시 확인

예외 처리

Questions?

LMS Q&A 게시판에 올려주세요.