네트워크 프로그래밍

05. 소켓에 인터넷 주소 할당하기

주소정보의 표현

IPv4 기반의 주소표현을 위한 구조체


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

주소체계

PORT번호

32비트 IP주소

사용되지 않음

IP주소와 PORT번호는 구조체 sockaddr_in의 변수에 담아서 표현한다.

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

32비트 IPv4 인터넷 주소

Datatype	Description	Header
int8_t	Signed 8-bit integer	<sys/types.h>
uint8_t	Unsigned 8-bit integer	<sys/types.h>
int16_t	Signed 16-bit integer	<sys/types.h>
uint16_t	Unsigned 16-bit integer	<sys/types.h>
int32_t	Signed 32-bit integer	<sys/types.h>
uint32_t	Unsigned 32-bit integer	<sys/types.h>
sa_family_t	Address family of socket address structure	<sys/socket.h>
socklen_t	Length of socket address structure, normally uint32_t	<sys/socket.h>
in_addr_t	IPv4 address, normally uint32_t	<netinet/in.h>
in_port_t	TCP or UDP port, normally uint16_t	<netinet/in.h>

POSIX에서 정의하고 있는 자료형

구조체 sockaddr_in의 멤버에 대한 분석

- 멤버 sin_family
  - 주소체계 정보 저장

address family 의 의미 = = address family와 같은 의미

주소체계(Address Family)	의 미
AF_INET	IPv4 인터넷 프로토콜에 적용하는 주소체계
AF_INET6	IPv6 인터넷 프로토콜에 적용하는 주소체계
AF_LOCAL	로컬 통신을 위한 유닉스 프로토콜의 주소체계

- 멤버 sin_port
  - 16비트 PORT번호 저장
  - 네트워크 바이트 순서로 저장

포트, ip 는 넣어줄때 network byte order로 넣어줘야함

- 멤버 sin_addr
  - 32비트 IP주소정보 저장
  - 네트워크 바이트 순서로 저장
  - 멤버 sin_addr의 구조체 자료형 in_addr 사실상 32비트 정수자료형

```
#define PF_INET 2
#define AF_INET PF_INET
```

(/usr/include/bits/socket.h 참조)

- 멤버 sin_zero
  - 특별한 의미를 지니지 않는 멤버
  - 반드시 0으로 채워야 한다.

구조체 sockaddr_in의 활용의 예


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

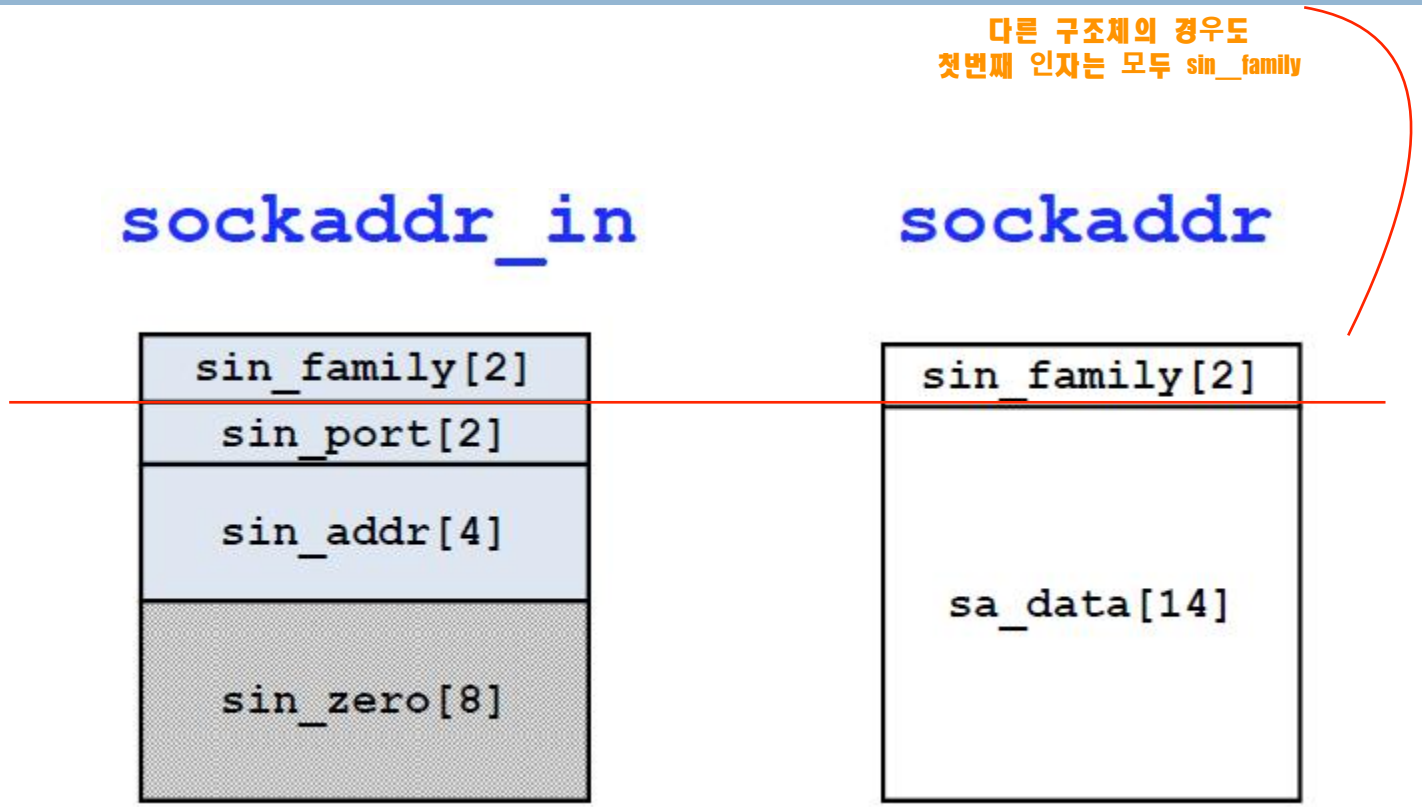
바인드 == 소켓과 포트의 연결
〉〉주소를 알아야 연결해줌

sockaddr_in은 IPv4 전용 주소정보 표현 구조체

```
struct sockaddr
{
    sa_family_t    sin_family;    // 주소체계(Address Family)
    char           sa_data[14];   // 주소정보
};
```

sockaddr은 다양한 주소체계의 주소정보를 담을 수 있는 구조체

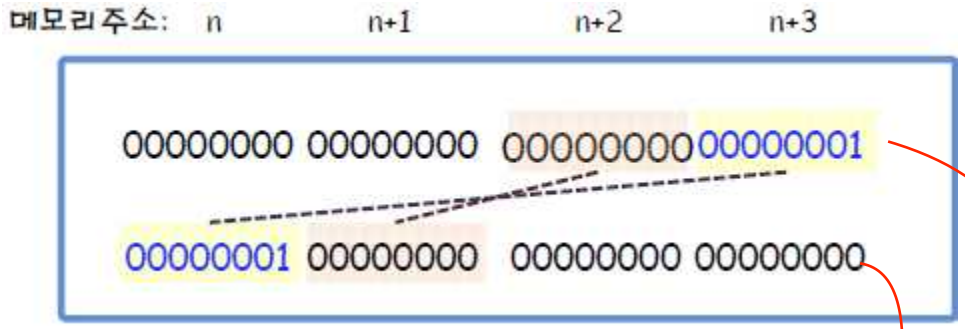
주소표현을 위한 구조체 구조



네트워크 바이트 순서

CPU에 따라 달라지는 정수의 표현

정수 1을 저장하는 두 가지 방법



방법1: 큰 값을 표현하는 상위 바이트(MSB) 부터 메모리 주소에 저장 (Big-endian)

방법2: 작은 값을 하위 바이트(LSB) 부터 메모리 주소에 저장 (Little-endian)

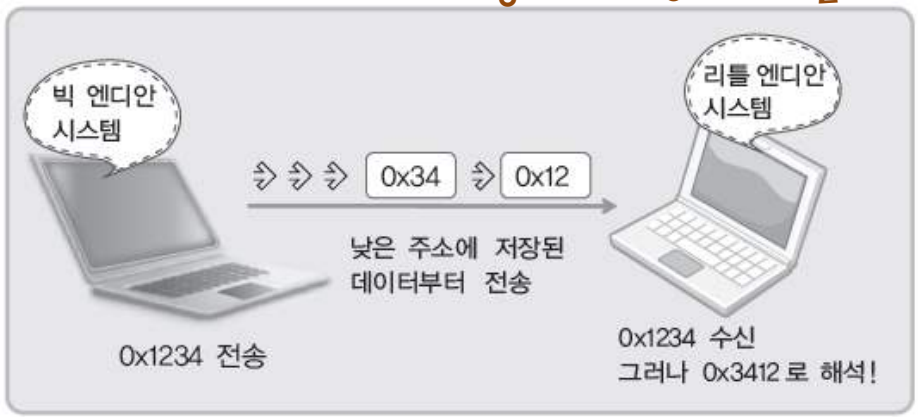
CPU마다 데이터를 표현 및 해석하는 방식이 다르다!

바이트 순서(Order)와 네트워크 바이트 순서

- 빅 엔디안(Big Endian)
  - 상위 바이트의 값을 작은 번지수에 저장
- 리틀 엔디안(Little Endian)
  - 상위 바이트의 값을 큰 번지수에 저장
- 호스트 바이트 순서
  - CPU별 데이터 저장방식을 의미함
- 네트워크 바이트 순서
  - 통일된 데이터 송수신 기준을 의미함
  - 빅 엔디안이 기준이다!

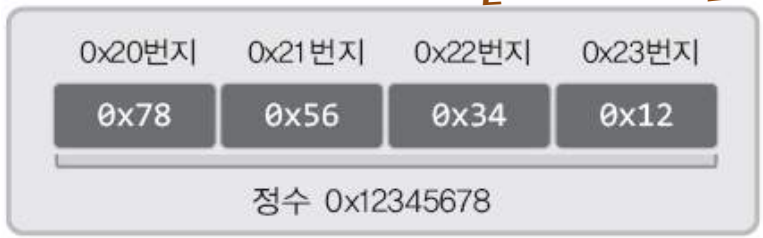
이러한 차이점을 극복하기 위해
〉네트워크에 데이터를 올릴때
어떤 순서로 올릴지 표준을 정해놓은것
〉〉"빅엔디안" 기준

데이터 송수신 과정에서의 문제

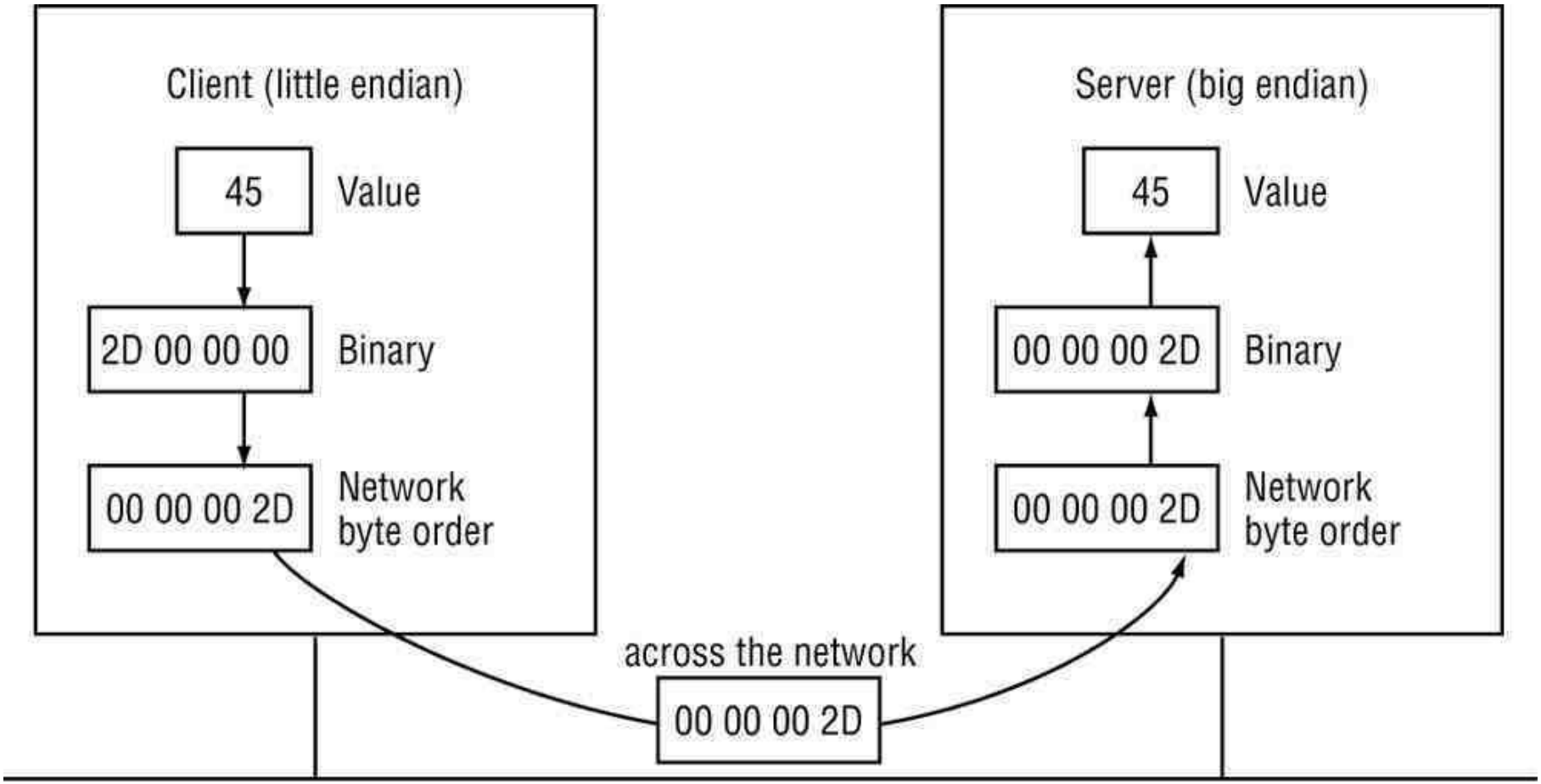


빅 엔디안 모델

리틀 엔디안 모델



바이트 순서(Order)와 네트워크 바이트 순서



바이트 순서의 변환

바이트 변환함수

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

host to network

Tip: 1byte 단위로 저장되므로 char처럼 1byte짜리는 빅이든 리틀이든 똑같이 저장함

실수(float number)같은 경우 >api가 없음, 보통 문자화, 정수화해서 보낸다음 받고 다시 float형으로 변환해줌

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

이 기준을 적용하면 위 함수가 의미하는 바를 이해할 수 있다.

바이트 변환의 예


```
int main(int argc, char *argv[])
{
    unsigned short host_port=0x1234;
    unsigned short net_port;
    unsigned long host_addr=0x12345678;
    unsigned long 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;
}
```

바뀌었음
〉리틀엔디안 시스템

실행결과

```
root@my_linux:/tcpip# gcc endian_conv.c -o conv
root@my_linux:/tcpip# ./conv
Host ordered port: 0x1234
Network ordered port: 0x3412
Host ordered address: 0x12345678
Network ordered address: 0x78563412
```

인터넷 주소의 초기화와 할당

문자열 정보를 네트워크 바이트 순서의 정수로 변환


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

in_addr_t inet_addr(const char * string);
```

➜ 성공 시 빅 엔디안으로 변환된 32비트 정수 값, 실패 시 INADDR_NONE 반환

"211.214.107.99" 와 같이 점이찍힌 10진수로 표현된 문자열을 전달하면, 해당 문자열 정보를 참조해서 IP주소정보를 32비트 정수형으로 반환!

```
int main(int argc, char *argv[])
{
    char *addr1="1.2.3.4";
    char *addr2="1.2.3.256";
    unsigned long conv_addr=inet_addr(addr1);
    if(conv_addr==INADDR_NONE)
        printf("Error occured! \n");
    else
        printf("Network ordered integer addr: %#lx \n", conv_addr);

    conv_addr=inet_addr(addr2);
    if(conv_addr==INADDR_NONE)
        printf("Error occureded \n");
    else
        printf("Network ordered integer addr: %#lx \n\n", conv_addr);
    return 0;
}
```

실행결과

```
root@my_linux:/tcpip# gcc inet_addr.c -o addr
root@my_linux:/tcpip# ./addr
Network ordered integer addr: 0x4030201
Error occureded
```

역시 리틀엔디안임을 보여줌

inet_aton


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

int inet_aton(const char * string, struct in_addr * addr);
```

➡ 성공 시 1(true), 실패 시 0(false) 반환

- string　변환할 IP주소 정보를 담고 있는 문자열의 주소 값 전달.
- addr　변환된 정보를 저장할 in_addr 구조체 변수의 주소 값 전달.

inet_addr 함수와 동일.
다만, in_addr형 구조체 변수에 변환 결과 저장

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

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

```
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;
}
```

실행 결과

```
root@my_linux:/tcpip# gcc inet_aton.c -o aton
root@my_linux:/tcpip# ./aton
Network ordered integer addr: 0x4f7ce87f
```

inet_ntoa


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

char * inet_ntoa(struct in_addr adr);
```

➜ 성공 시 변환된 문자열의 주소 값, 실패 시 -1 반환

inet_aton 함수의 반대기능

```
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);
printf("Dotted-Decimal notation3: %s \n", str_arr);
return 0;
```

포인터에 assign안해줬는데 바뀜 〉조심할 것

실행결과

```
root@my_linux:/tcpip# gcc inet_ntoa.c -o ntoa
root@my_linux:/tcpip# ./ntoa
Dotted-Decimal notation1: 1.2.3.4
Dotted-Decimal notation2: 1.1.1.1
Dotted-Decimal notation3: 1.2.3.4
```

인터넷 주소의 초기화

서버는 바인드할때
클라이언트는 커넥트할때
〉초기화 할 것

일반적인 인터넷 주소의 초기화 과정

```
struct sockaddr_in addr;
char *serv_ip="211.217.168.13";       // IP주소 문자열 선언
char *serv_port="9190";               // PORT번호 문자열 선언
memset(&addr, 0, sizeof(addr));       // 구조체 변수 addr의 모든 멤버 0으로 초기화
addr.sin_family=AF_INET;              // 주소체계 지정
addr.sin_addr.s_addr=inet_addr(serv_ip);   // 문자열 기반의 IP주소 초기화
addr.sin_port=htons(atoi(serv_port));      // 문자열 기반의 PORT번호 초기화
```

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


```
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));
```

주로 서버에서 사용한다
>아이피를 자동으로 찾아서
대입해줌
포트는 이런것을 주세요
0.0.0.0
wild card.
ex)
inaddr_any이면서
port = 8000이면
현재 서버컴퓨터에 9000번포트를
목적지로 하는 모든 요청을
받겠다는 의미

현재 실행중인 컴퓨터의 IP를 사용하라!

주로 서버에서 사용

소켓에 인터넷 주소 할당하기


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

int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);
```

➜ 성공 시 0, 실패 시 -1 반환

- sockfd 주소정보를(IP와 PORT를) 할당할 소켓의 파일 디스크립터.
- myaddr 할당하고자 하는 주소정보를 지니는 구조체 변수의 주소 값.
- addrlen 두 번째 인자로 전달된 구조체 변수의 길이정보.

이번장의 포인트
〉내가 3 +4 를 보낸다고하면
총 4+ 1 + 4바이트를 필요로함
〉int를 넣을때는htonl()로 넣고
캐릭터를 넣을때는 1바이트니
그냥넣는식으로

```
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);

/* 주소정보 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));

/* 주소정보 할당 */
bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
. . . . .
```

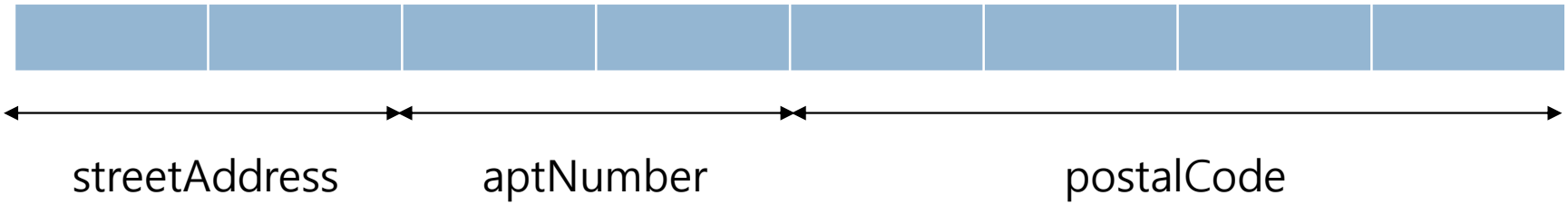
서버프로그램에서의 일반적인

주소할당의 과정!

20

구조체 오버레이:정렬과 패딩

메시지



구조체 오버레이(overlay)

각각의 메세지를 구조체화 해서
〉send 하기 전에 써주면
이해하기 훨씬 편함

```
struct addressInfo {
        uint16_t streetAddress;
        int16_t aptNumber;
        uint32_t postalCode;
} addrInfo;
.
.       /* assign values to field – convert byte order! */
.
send(sock, &addrInfo, sizeof(addrInfo), 0);
```

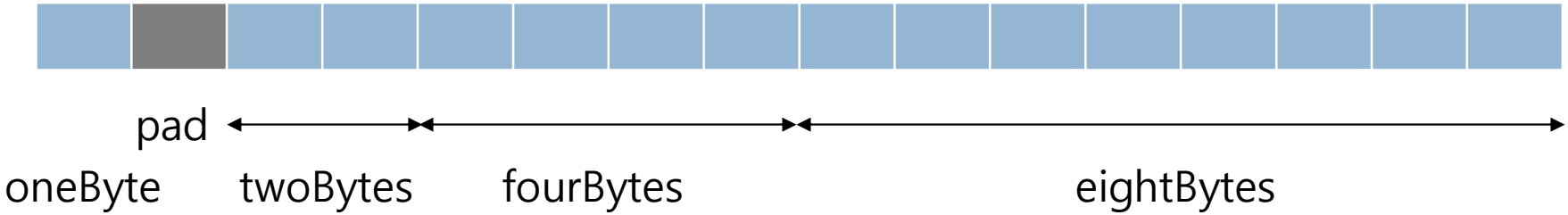
정렬(alignment)과 채우기(padding)

- sizeof(struct integerMessage)는?

단순계산시 15
실제 16이잡힘
oneByte다음에 짝수가 아니므로
1바이트 공백이 생김

```
struct integerMessage {
        uint8_t oneByte;
        uint16_t twoBytes;
        uint32_t fourBytes;
        uint64_t eightBytes;
}
```

정렬(alignment)과 채우기(padding)



정렬(alignment)과 채우기(padding)

"통신"이라는 가정하에
구조체 멤버는 반드시
어떤 메세지를 가지고 있게됨
ex)포인터의 경우 구조체내에
있을 필요가 없음

- Data structures are maximally aligned, that is, their addresses will be divisible by the size of the largest native integer.
- Other multibyte fields are aligned to their size, that is, a four-byte integer's address will be divisible by four, and a two-byte integer's address will be divisible by two.

멤버중 가장 큰 녀석의 사이즈로
나누어 떨어지는 곳에 위치 메모리들이
위치한다.
>>앞장의경우
8로나누어 떨어지는곳> 짝수
이므로 1바이트짜리도 짝수위치에있고
2바이트짜리는 홀수위치에 있을 수 없으니
한칸띄고 다음칸에 있다
4바이트짜리도 짝수위치에 존재
8바이트짜리도 짝수위치에 존재

정렬(alignment)과 채우기(padding)

- sizeof(struct backwardMessage)는?

8 4 2 1 순서대로 나누어 떨어지는곳에 들어간다고 보면
15에 다 들어갈 수 있다고 볼 수 있지만 실제로는
16이다 (마지막에 padding이 들어감)
>>단 data크기가 제일 큰녀석으로 나누어 떨어질 수 있도록 만들어짐
(다음 구조체 혹은 데이터가 선언될 때 문제가 발생하지 않도록)

```
struct backwardMessage{
        uint64_t eightBytes;
        uint32_t fourBytes;
        uint16_t twoBytes;
        uint8_t oneByte;
}
```

정렬(alignment)과 채우기(padding)

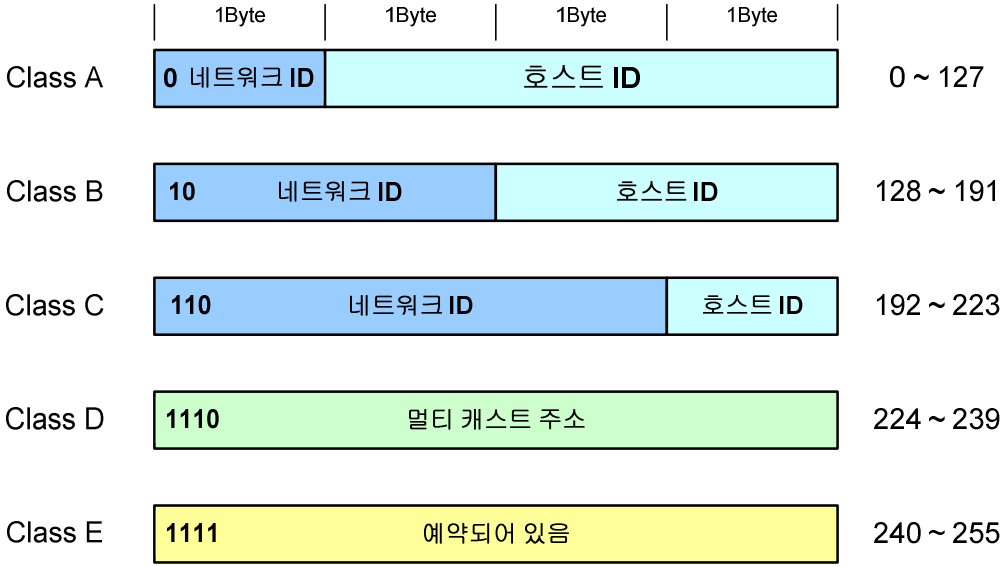

```
struct integerMessage {
        uint8_t oneByte;
        uint8_t padding;
        uint16_t twoBytes;
        uint32_t fourBytes;
        uint64_t eightBytes;
}
```

부록

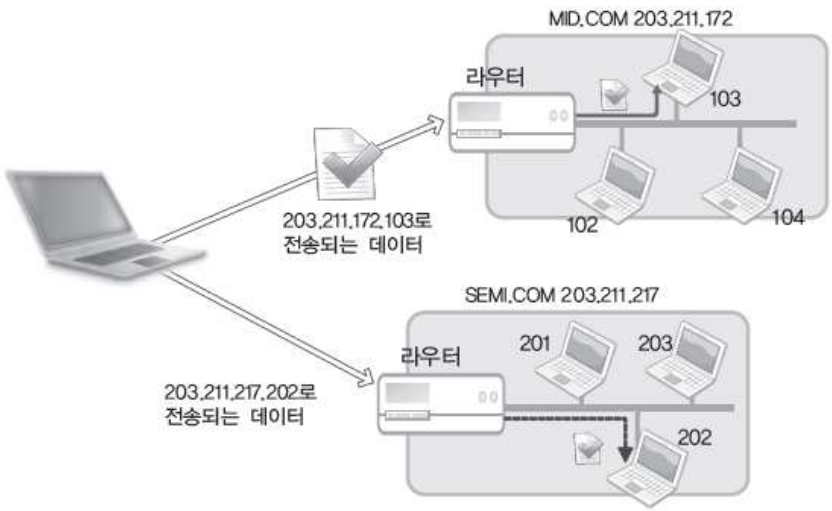
인터넷 주소(Internet Address)

- 인터넷 주소란?
  - 인터넷상에서 컴퓨터를 구분하는 목적으로 사용되는 주소.
  - 4바이트 주소체계인 IPv4와 16바이트 주소체계인 IPv6가 존재한다.
  - 소켓을 생성할 때 기본적인 프로토콜을 지정해야 한다.
  - 네트워크 주소와 호스트 주소로 나뉜다. 네트워크 주소를 이용해서 네트워크를 찾고, 호스트 주소를 이용해서 호스트를 구분한다.

첫 번째 바이트 정보만 참조해도 IP주소의 클래스 구분이 가능하며, 이로 인해서 네트워크 주소와 호스트 주소의 경계 구분이 가능하다.

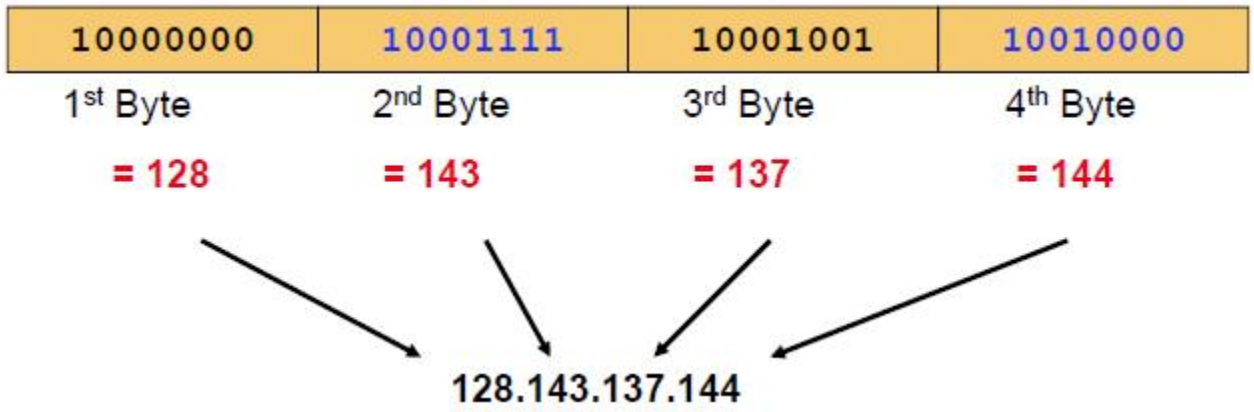


IPv4 인터넷 주소의 체계



인터넷 주소의 역할

IP 주소 표기


10000000
10001111
10001001
10010000
1st Byte
2nd Byte
3rd Byte
4th Byte
= 128
= 143
= 137
= 144
128.143.137.144

루프백 주소 (Loopback Address)

- 컴퓨터 자신을 의미하는 주소로 그 값은 127.0.0.1로 약속되어 있다.
  - 루프백 주소(127.0.0.1)로 데이터를 전송하면, 전송된 데이터는 데이터를 전송한 컴퓨터로 수신이 된다.

- 루프백 주소 확인
  - hosts 파일에서 확인할 수 있다.
    - 유닉스 계열 OS
      - /etc/hosts
    - 윈도우
      - C:₩Windows₩system32₩drivers₩etc₩hosts

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
```