Socket Address Structure and Byte Ordering Functions

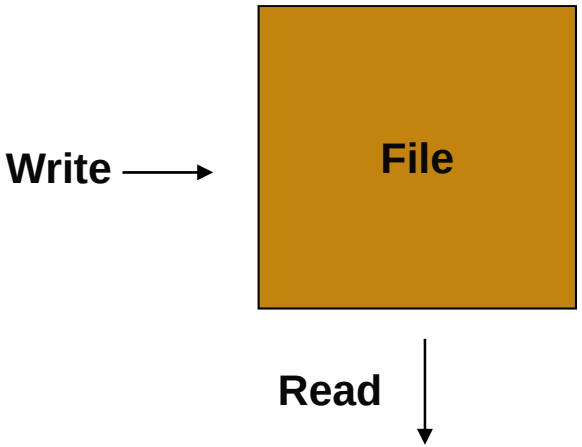
Department of Computer Engineering

Kyung Hee University.

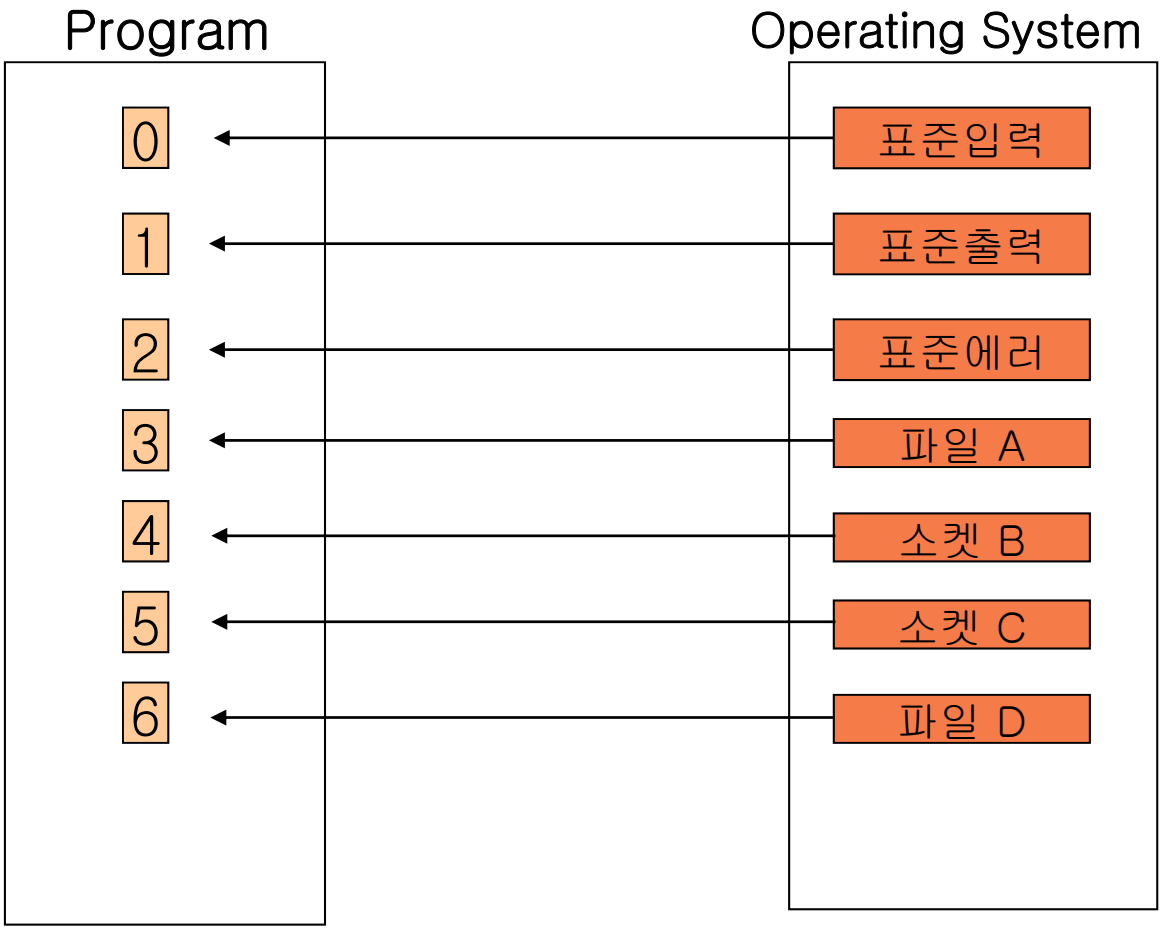
Choong Seon Hong

네트워크 프로그래밍

- 네트워크 프로그래밍 이란?
  - 네트워크로 연결되어 있는 두 호스트간의 데이터 송수신.
  - 파일 입출력과의 차이점은 데이터를 주고 받는 대상에 있다

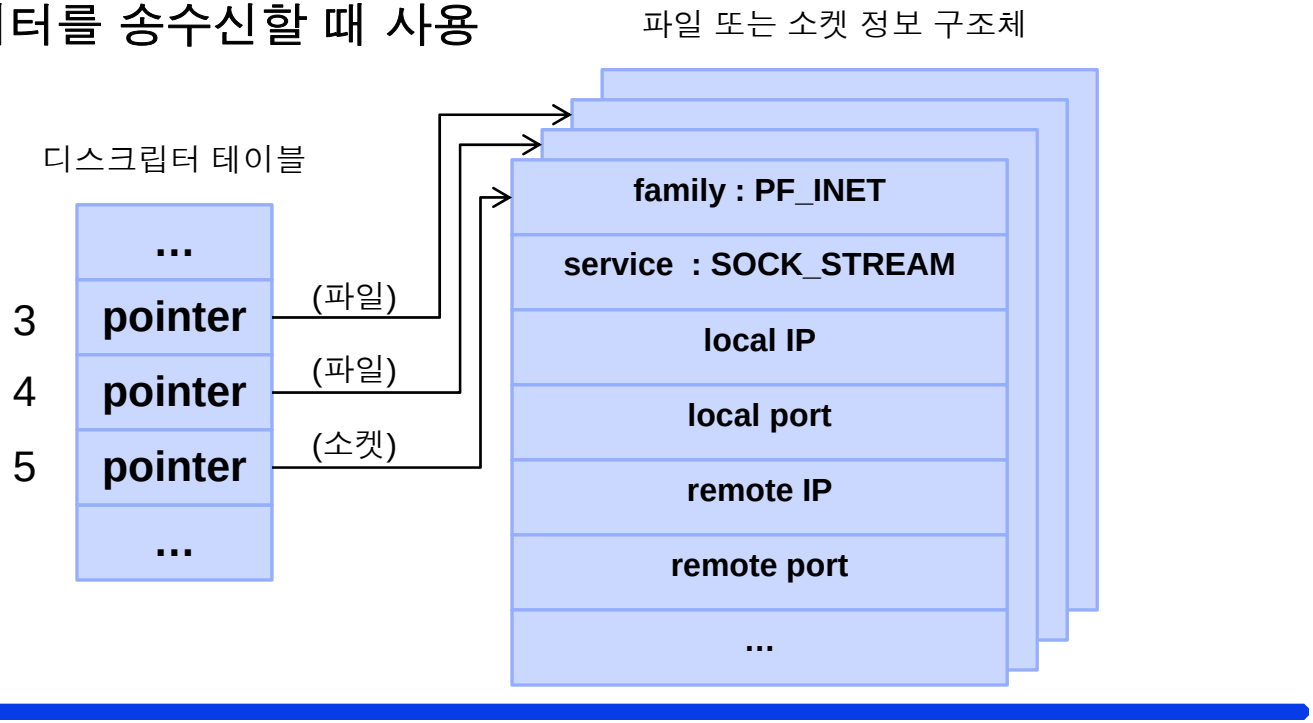


파일 디스크립터


Program
Operating System
0
1
2
3
4
5
6
표준입력
표준출력
표준에러
파일 A
소켓 B
소켓 C
파일 D

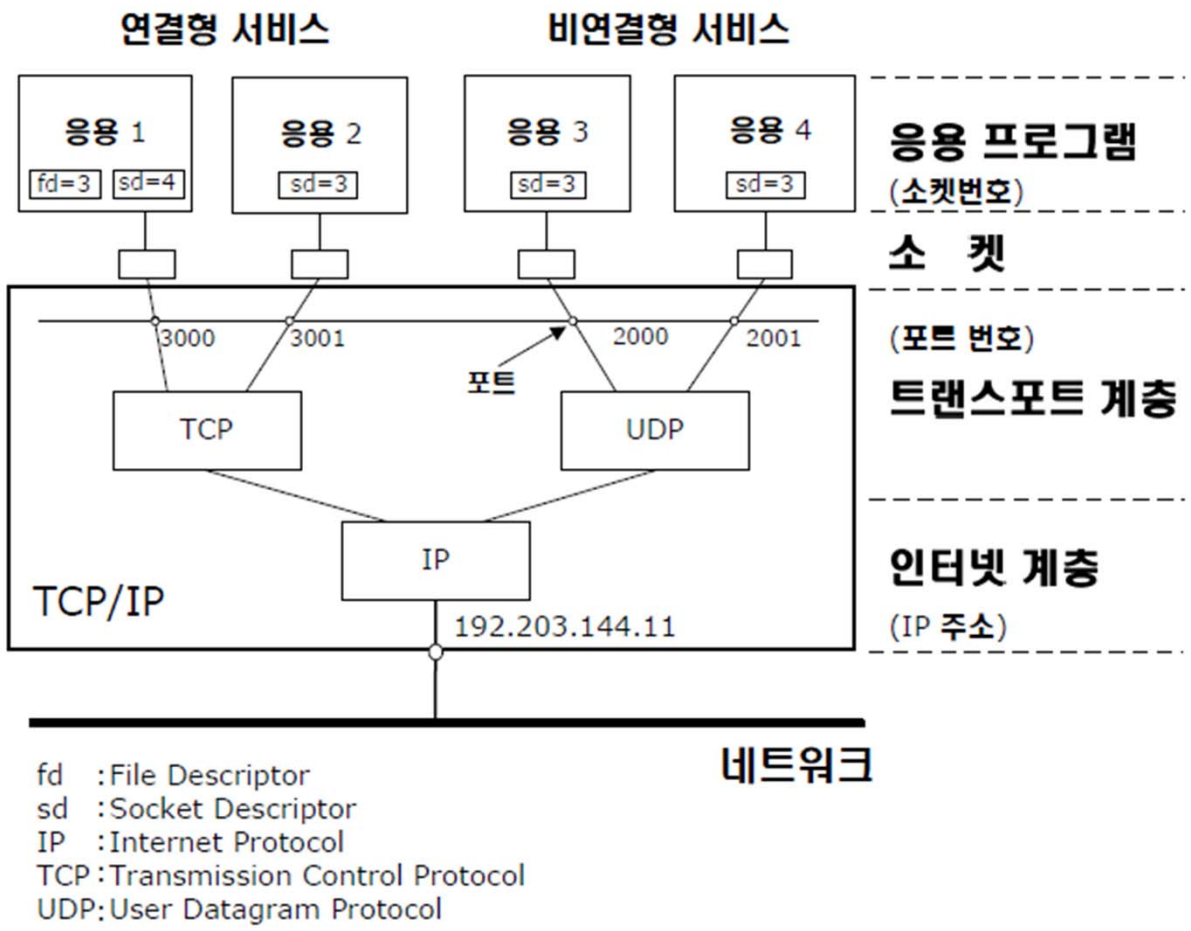
소켓 번호

- 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



소켓 번호

- 응용 프로그램과 소켓 그리고 TCP/IP의 관계



소켓의 생성

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

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

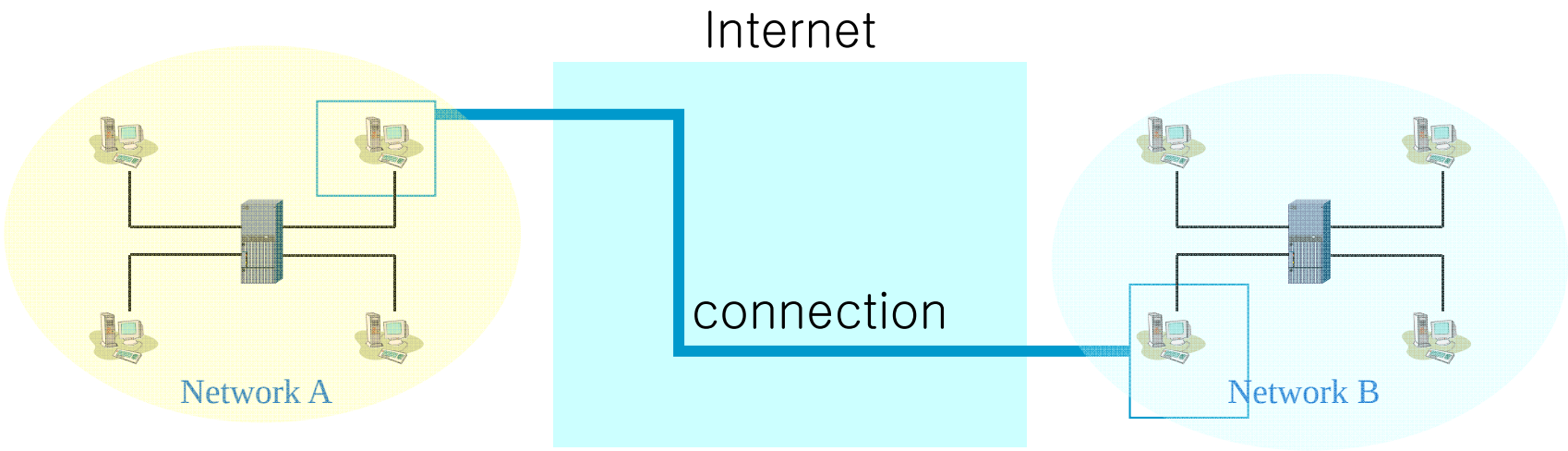
```
PF_INET         // IPv4 프로토콜 체계
PF_INET6        // IPv6 프로토콜 체계
PF_UNIX         // 유닉스 방식의 프로토콜 체계
PF_LOCAL        // Local 통신을 위한 UNIX 프로토콜 체계
PF_PACKET       // 리눅스에서 패킷 캡쳐를 위해 사용
```

  - 서비스 타입

```
SOCK_STREAM     // TCP 소켓 (연결형)
SOCK_DGRAM      // UDP 소켓 (비연결형)
SOCK_RAW        // RAW 소켓
```

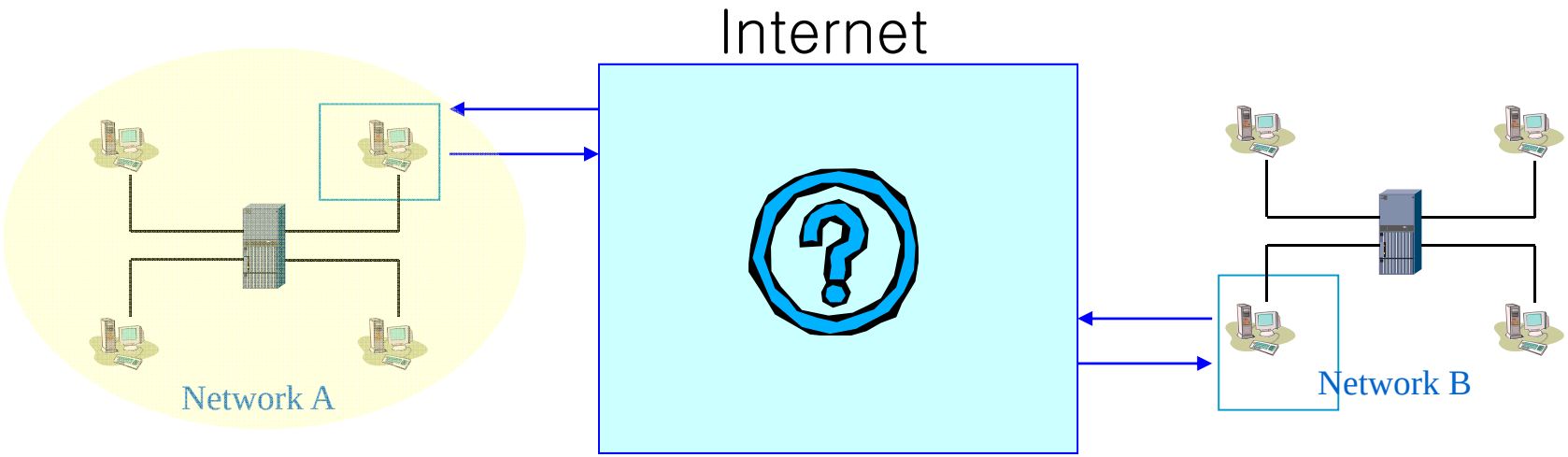
소켓의 타입 1

- 연결 지향형 소켓(SOCK_STREAM, TCP 소켓)
  - 신뢰성 있는 순차적인 바이트 기반의 연결 지향 전송 타입
    - 전송하는 순서대로 데이터가 전달
    - 에러나 데이터의 손실 없이 전달됨
    - 전송되는 데이터의 경계가 존재하지 않음



소켓의 타입 2

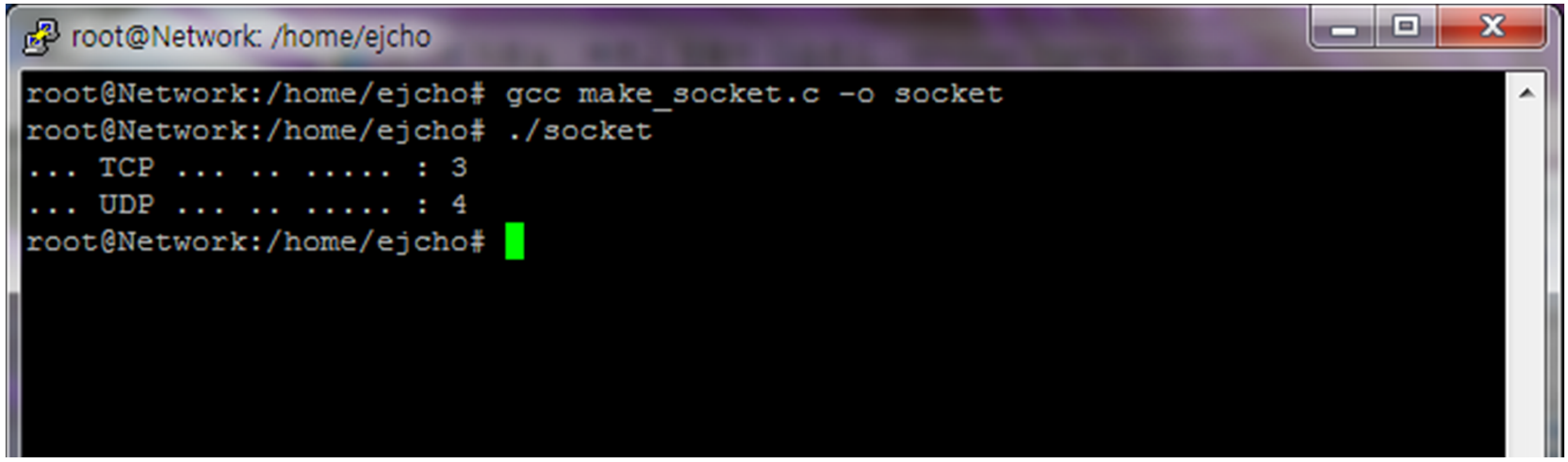
- 비 연결 지향형 소켓(SOCK_DGRAM, UDP 소켓)
  - 전송되는 순서에 상관없이 가장 빠른 전송을 지향
  - 전송되는 데이터는 손실 될 수도 있고, 에러 발생 가능
  - 전송되는 데이터의 경계가 존재
  - 한번에 전송되는 데이터의 크기는 제한됨



프로그램 예제 확인

프로그램 예제

- make_socket.c



IPv4의 주소체계를 나타내는 구조체

- 클라이언트 또는 서버의 구체적인 주소를 표현을 위해 사용
  - 주소체계
  - IP 주소
  - 포트번호

```
struct   sockaddr_in {
        sa_family_t         sin_family;     // 주소 체계
        uint16_t            sin_port;       // 16비트 포트번호
        struct in_addr      sin_addr;       // 32 비트 IP 주소
        char                sin_zero[8];    // 사용되지 않음
};
```

```
struct in_addr {
        uint32_t            s_addr;       // 32비트  IP 주소를 저장하는 구조체
};
```

☛주의 : 모든 데이터는 네트워크 바이트 순서로 저장 해야 한다.

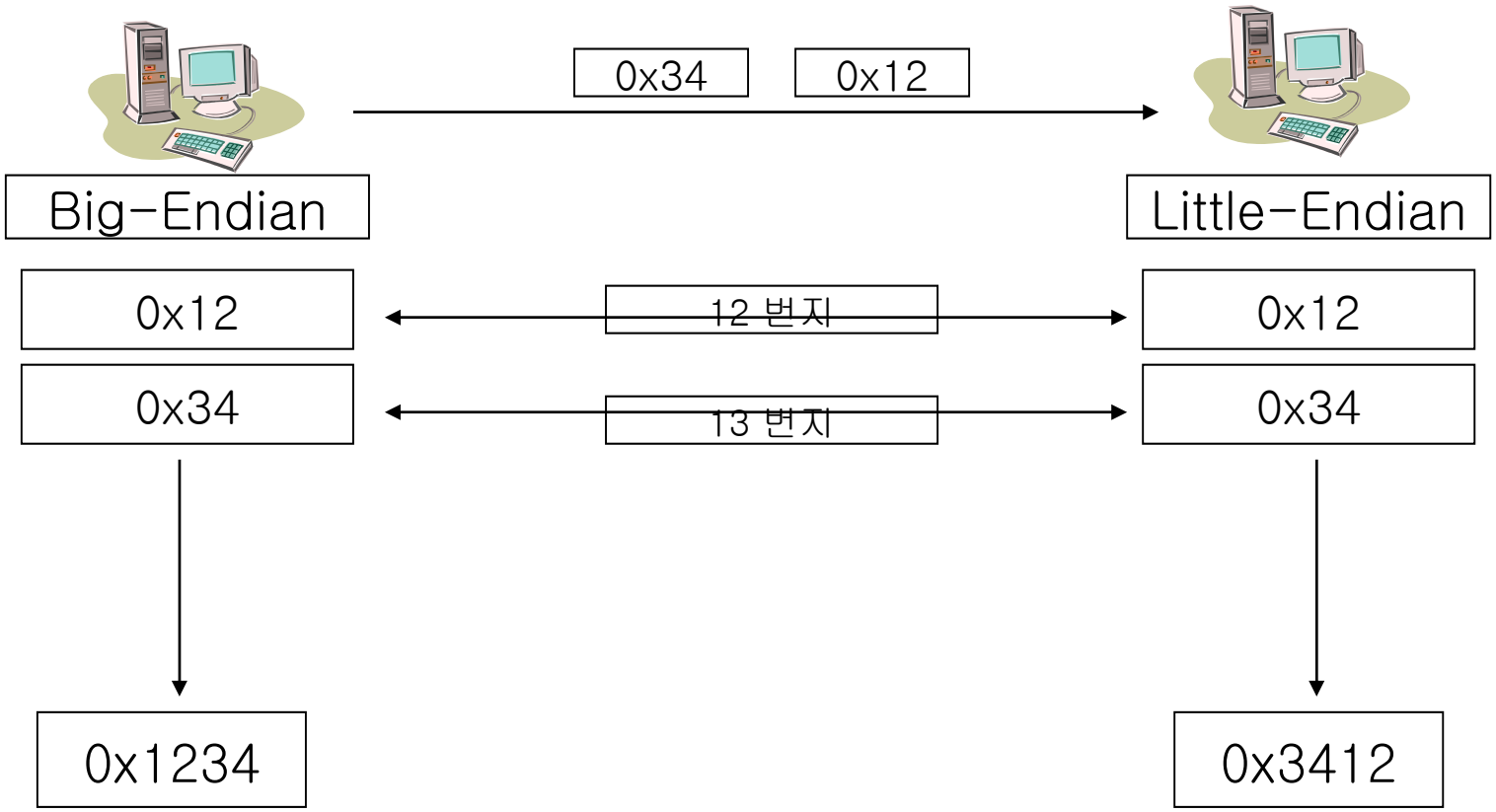
데이터 타입

Data type	Description	Header
int8_t	signed 8-bit *int* (signed char)	<sys/types.h>
uint8_t	unsigned 8-bit *int* (unsigned char)	
int16_t	signed 16-bit *int* (signed short)	
uint16_t	unsigned 16-bit *int* (unsigned short)	
int32_t	signed 32-bit *int* (signed long)	
uint_32_t	unsigned 32-bit *int* (unsigned long)	
sa_family_t	address family	<sys/socket.h>
socklen_t	length of struct	
in_addr_t	IPv4 address, normally *uint32_t*	<netinet/in.h>
in_port_t	TCP or UDP port, normally *uint16_t*	<netinet/in.h>

바이트 순서

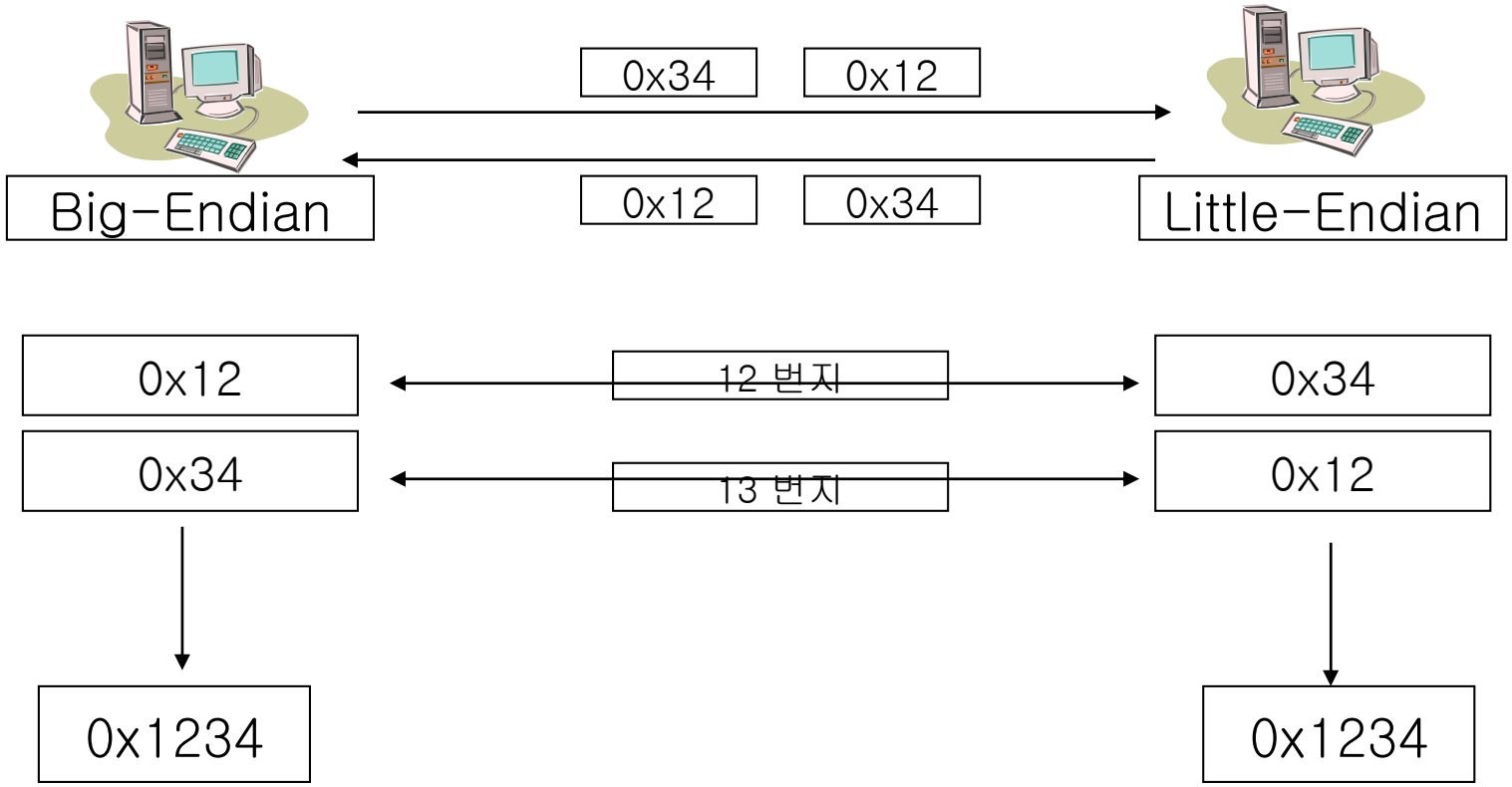
- 호스트 바이트 순서
  - 컴퓨터가 내부 메모리에 숫자를 저장하는 순서
  - CPU의 종류에 따라 다름
    - 80x86 : little-endian
    - MC68000 : big-endian
  - 네트워크 바이트 순서
    - 포트번호나 IP 주소와 같은 정보를 바이트 단위로 전송하는 순서
    - high-order (big-endian) 로 전송
  - 80x86과 MC68000 간의 데이터 전송
    - 바이트 순서가 바뀜

데이터 표현 방식에 따른 문제점



네트워크 바이트 순서

- 네트워크 바이트 순서는 Big-Endian 방식을 적용하기로 약속



바이트 순서 변환 함수

-unsigned short integer (2바이트) 변환

unsigned short **htons**(unsigned short); → host-to-network 바이트 변환

unsigned short **ntohs**(unsigned short); → network-to-host 바이트 변환

- unsigned long integer (4바이트) 변환

unsigend long **htonl**(unsigned long); → host-to-network 바이트 변환

unsigned long **ntohl**(unsigned long); → network-to-host 바이트 변환

‘h’ : host byte order ‘n’ : network byte order

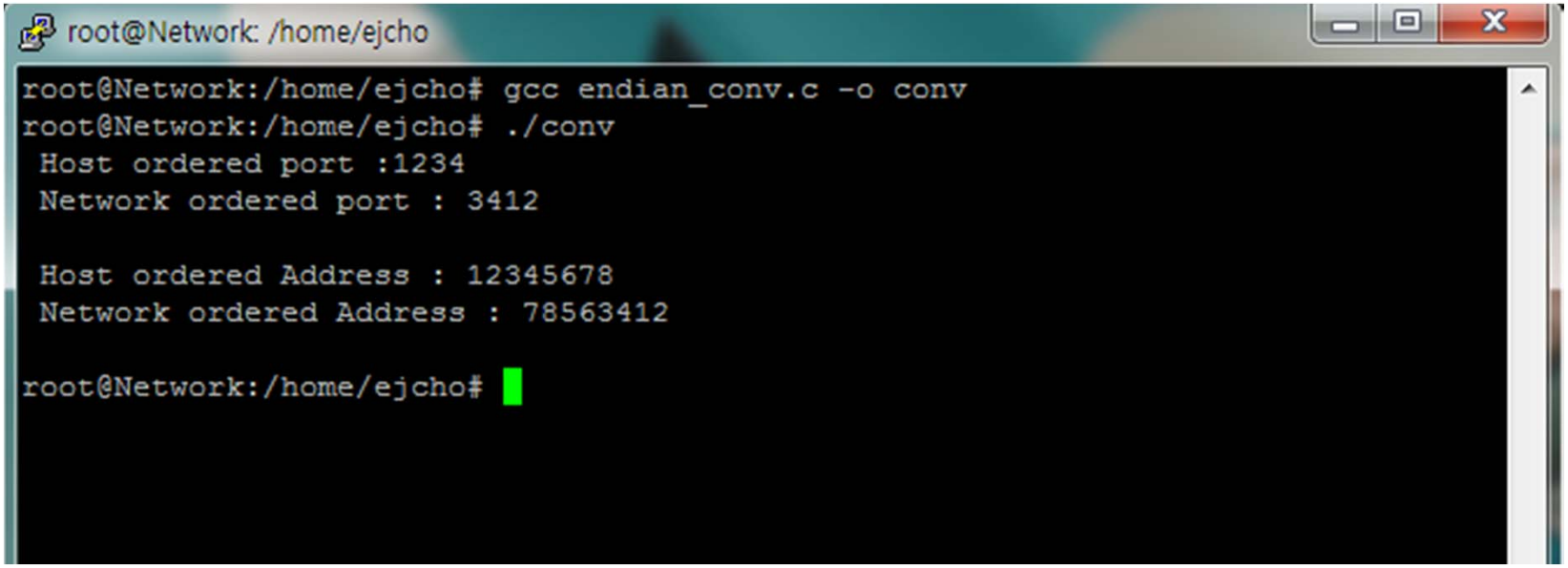
‘s’ : short (16 bit) ‘l’ : long (32 bit)

프로그램 예제 확인

1. 프로그램 예제

- endian_conv.c

2. 실행결과



Byte Manipulation Functions

```
#include <string.h>

void bzero (void *dest, size_t nbytes);

void bcopy (const void *src, void *dest, size_t nbytes);

int bcmp (const void *ptr1, const void *ptr2, size_t nbytes);
```

```
#include <string.h>

void *memset (void *dest, int c, size_t len);

void memcpy (void *dest, const void *src, size_t nbytes);

int memcmp (const void *ptr1, const void *ptr2, size_t nbytes);
```