Part 03. 리눅스 소켓 프로그래밍

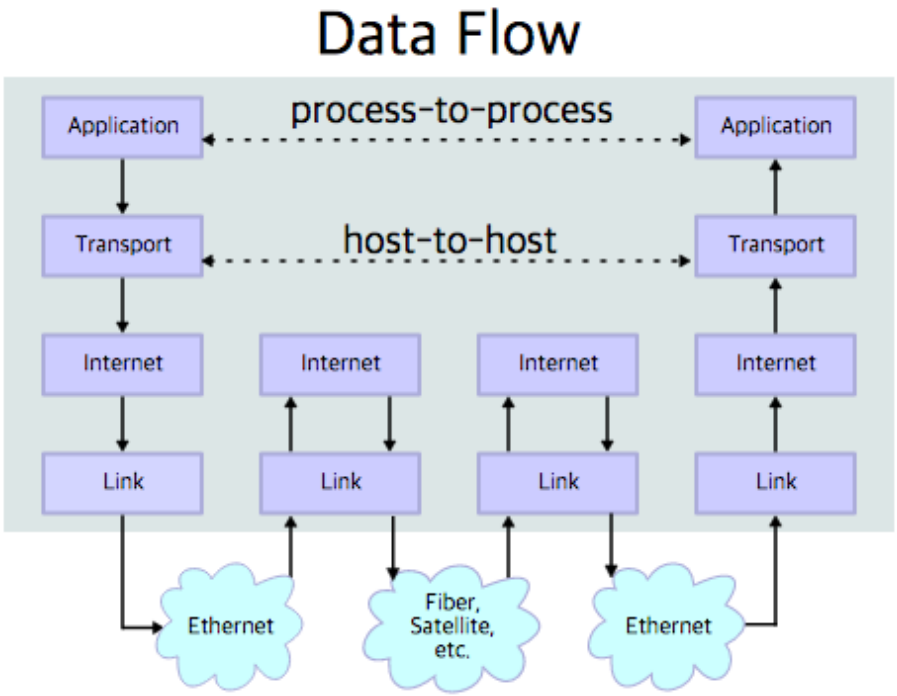
Chapter 01. 소켓 개요

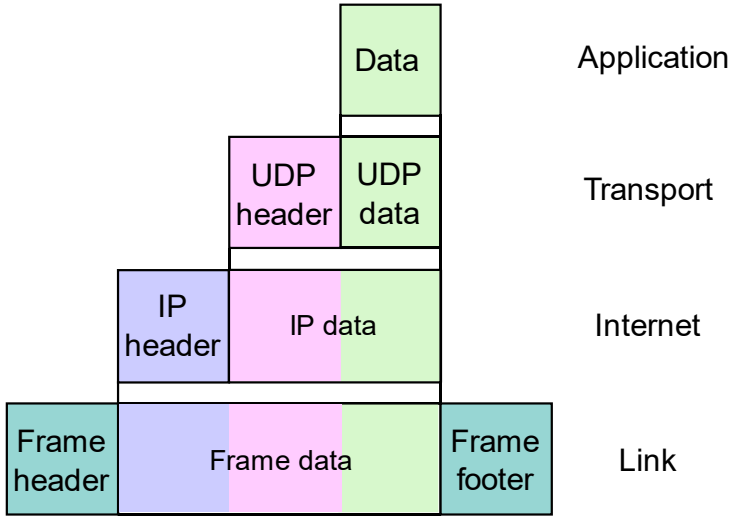
진행 순서

Chapter 01_01	TCP/IP
Chapter 01_02	소켓
Chapter 01_03	바이트 순서

Chapter 01_01 TCP/IP

TCP/IP 란?
인터넷 프로토콜 제품군(Internet Protocol Suite) 은 인터넷 및 유사한 컴퓨터 네트워크에서 사용되는 개념적 모델 및 통신 프로토콜 집합입니다. 제품군의 기본 프로토콜은 TCP (Transmission Control Protocol) 및 IP (Internet Protocol)이기 때문에 일반적으로 TCP / IP 라고 합니다.





Encapsulation / Decapsulation

출처: https://en.wikipedia.org/wiki/Internet_protocol_suite

Chapter 01_01 TCP/IP

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

```
struct iphdr {
#if defined(__LITTLE_ENDIAN_BITFIELD)
        __u8    ihl:4,
                version:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
        __u8    version:4,
                ihl:4;
#else
#error  "Please fix <asm/byteorder.h>"
#endif
        __u8    tos;
        __be16  tot_len;
        __be16  id;
        __be16  frag_off;
        __u8    ttl;
        __u8    protocol;
        __sum16 check;
        __be32  saddr;
        __be32  daddr;
        /*The options start here. */
};
```

Chapter 01_01 TCP/IP

TCP segment header

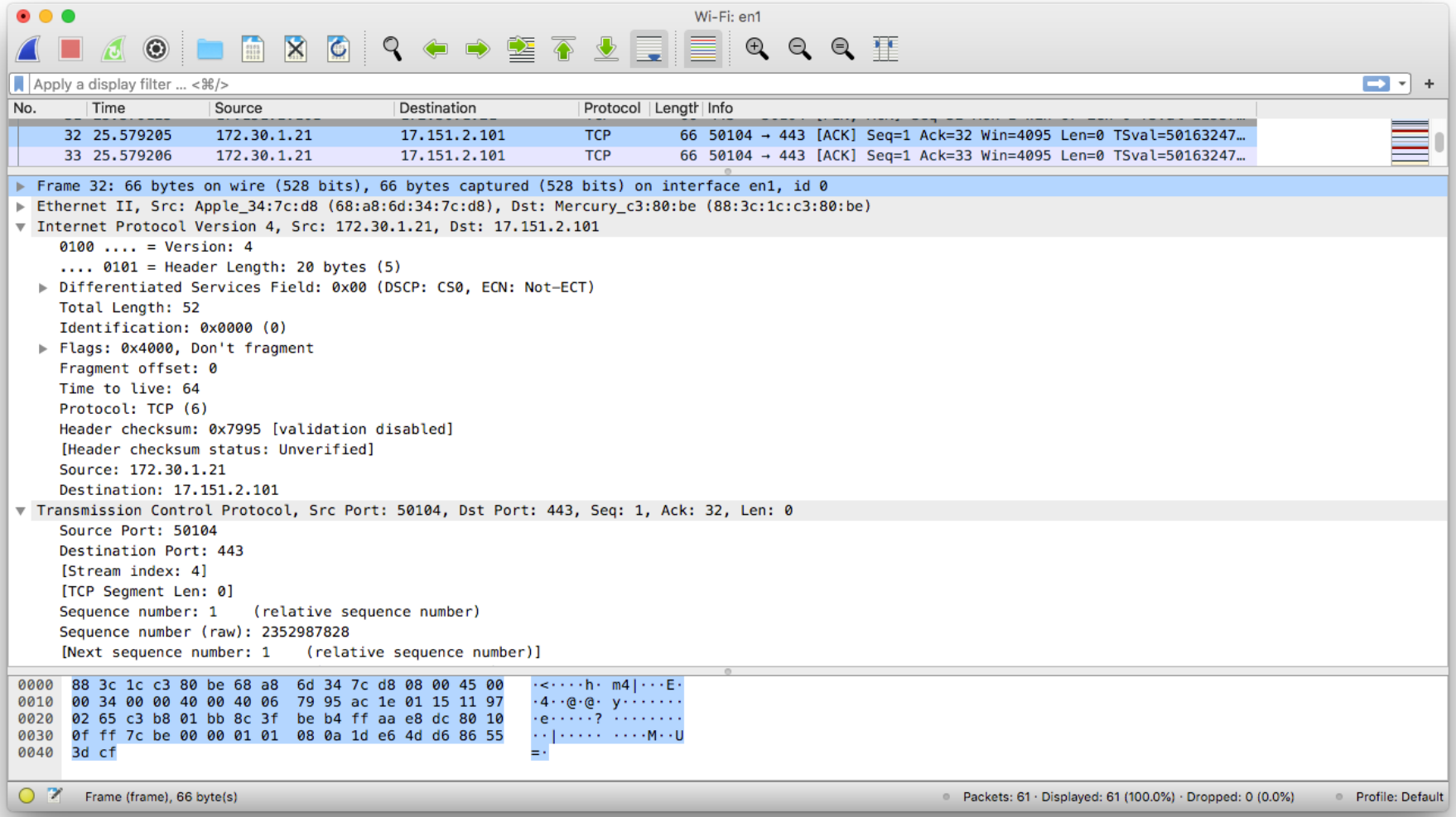
Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)													
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

```
struct tcphdr {
    __u16   source;
    __u16   dest;
    __u32   seq;
    __u32   ack_seq;
```

```
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u16   res1:4,
        doff:4,
        fin:1,
        syn:1,
        rst:1,
        psh:1,
        ack:1,
        urg:1,
        ece:1,
        cwr:1;
```

```
#elif defined(__BIG_ENDIAN_BITFIELD)
    __u16   doff:4,
        res1:4,
        cwr:1,
        ece:1,
        urg:1,
        ack:1,
        psh:1,
        rst:1,
        syn:1,
        fin:1;
#else
#error  "Adjust your <asm/byteorder.h> defines"
#endif
    __u16   window;
    __u16   check;
    __u16   urg_ptr;
};
```

Chapter 01_01 TCP/IP



https://www.wireshark.org/

Chapter 01_02 소켓

소켓이란?
네크워크 소켓은 컴퓨터 네트워크의 노드 내에서 데이터를 보내거나 받는 내부 끝점(endpoint)을 의미합니다.
리눅스에서 소켓은 I/O 인터페이스의 한 종류이며 네트워크 프로그래밍의 가장 기본적인 부분을 담당합니다.

리눅스 소켓의 구분

도메인 분류 (범위)

네트워크 도메인 소켓 – 네트워크 주소(IP/PORT)를 통해 통신
유닉스 도메인 소켓 – 시스템(호스트)내에서 파일 경로를 통해 통신

타입 분류

스트림 소켓 – TCP, 연결 지향형, 연결 필요
데이터그램 소켓 – UDP, 비 연결 지향형, 연결 없이 데이터 전송 가능
raw 소켓 – 전송 계층 포맷없이 인터넷 프로토콜 패킷을 직접 주고받는 소켓 (프로토콜 헤더 포함)
응용 계층에서는 사용자 데이터만 바라보고, 전송 계층에서는 TCP 헤더와 데이터,
네트워크 계층에서는 IP 헤더, TCP 헤더, 데이터를 바라보는데,
raw 소켓의 의미는 각 계층에서 IP 헤더, TCP 헤더, 데이터를 모두 바라보는 것을 의미

Chapter 01_02 소켓

소켓 생성

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

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

socket ()은 통신을 위한 엔드포인트를 생성하고 해당 엔드포인트를 참조하는 파일 디스크립터를 리턴합니다.
성공적인 호출에 의해 리턴 된 파일 디스크립터는 현재 프로세스에서 열리지 않은 가장 낮은 번호의 파일 디스크립터입니다.

domain	
AF_UNIX	유닉스 도메인 소켓(AF_LOCAL)
AF_INET	IPv4 네트워크 도메인 소켓
AF_INET6	IPv6 네트워크 도메인 소켓

type	
SOCK_STREAM	TCP 스트림 소켓
SOCK_DGRAM	UDP 데이터그램 소켓
SOCK_RAW	raw 소켓

protocol	
IPPROTO_IP	IP 프로토콜 사용
IPPROTO_TCP	TCP 프로토콜 사용 (SOCK_STREAM)
IPPROTO_UDP	UDP 프로토콜 사용 (SOCK_DGRAM)
IPPROTO_ICMP	ICMP 프로토콜 사용

Chapter 01_02 소켓

소켓도 파일 디스크립터이므로 사용이 완료되면 close()를 이용해 닫아 주어야 합니다.
성공 시 새 소켓 의 파일 디스크립터가 리턴, 오류 발생 시 -1이 리턴되고 errno가 적절하게 설정됩니다.

errno	
EACCESS	권한 오류
EINVAL	protocol, type 등 인자 오류
EMFILE	프로세스에서 열린 파일 디스크립터 개수 제한
ENFILE	시스템에서 열린 파일 디스크립터 개수 제한
ENOBUFS	메모리 부족
ENOMEM	메모리 부족

```
int tcp_sock, udp_sock;

tcp_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (tcp_sock == -1) {
        /* error */
}
udp_sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (udp_sock == -1) {
        /* error */
}
…
close(tcp_sock);
close(udp_sock);
```

Chapter 01_03 바이트 순서

엔디안(바이트 순서)이란?
컴퓨팅에서 엔디안 (endianness)은 숫자의 이진 표현 내에서 바이트의 순서를 나타냅니다.

빅 엔디안 순서는 최상위 바이트를 가장 먼저, 최하위 바이트는 마지막에 두고 리틀 엔디안 순서는 반대입니다.
예를 들어, 부호가 없는 16 진 숫자 0x1234를 생각해보면 이는 최소 2 바이트를 나타내야 합니다.
빅 엔디안 순서에서는 0x12 0x34이고 리틀 엔디안 순서에서는 바이트가 0x34 0x12로 정렬됩니다 ('first'가 왼쪽에 있다고 가정).

시스템에서 엔디안은 CPU 종속적인 규칙인데
빅 엔디안은 네트워킹 프로토콜 (IP, TCP, UDP)에서 기본적으로 사용되는 순서이며
RISC 프로세서(Sparc, Motorola CPU) 들이 사용하고 있는 방식입니다.
반대로, 리틀 엔디안은 인텔 호환 계열 CPU에서 사용되는 방식입니다.

여기서 주의할 점이 우리가 사용하는 대부분의 컴퓨터는 인텔 호환 계열 CPU를 사용하므로
내부적으로 리틀 엔디안 방식을 사용하고,
소켓 프로그래밍 시 사용되는 네트워킹 프로토콜 들은 기본적으로 빅 엔디안 방식을 사용합니다.

Big Endian (0x1234)

0x12	0x34
address 0x0000	address 0x0001

Little Endian (0x1234)

0x34	0x12
address 0x0000	address 0x0001

Chapter 01_03 바이트 순서

바이트 순서 변환 함수

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

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

htonl () 함수는 부호없는 정수 hostlong을 호스트 바이트 순서에서 네트워크 바이트 순서로 변환합니다.

htons () 함수는 부호없는 짧은 정수 hostshort를 호스트 바이트 순서에서 네트워크 바이트 순서로 변환합니다.

ntohl () 함수는 부호없는 정수 netlong을 네트워크 바이트 순서에서 호스트 바이트 순서로 변환합니다.

ntohs () 함수는 부호없는 짧은 정수 netshort를 네트워크 바이트 순서에서 호스트 바이트 순서로 변환합니다.

Chapter 01_03 바이트 순서

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

union longbyte {
	long l;
	unsigned char c[4];
};

int main()
{
	union longbyte lb;

	lb.l = 0x1234;
	printf("[Host Order] %02x, %02x, %02x, %02x\n", lb.c[0], lb.c[1], lb.c[2], lb.c[3]);

	lb.l = htonl(lb.l);
	printf("[Network Order] %02x, %02x, %02x, %02x\n", lb.c[0], lb.c[1], lb.c[2], lb.c[3]);

	return 0;
}
```

```
[root@localhost ch01]# gcc -g endian_test.c -o endian_test
[root@localhost ch01]# ./endian_test
[Host Order] 34, 12, 00, 00
[Network Order] 00, 00, 12, 34
```