Chapter 02

소켓의 타입과 프로토콜의 설정

프로토콜의 이해와 소켓의 생성

- 프로토콜(Protocol)이란?
  - 개념적으로 약속의 의미를 담고 있다.
  - 컴퓨터 상호간의 데이터 송수신에 필요한 통신규약
  - 소켓을 생성할 때 기본적인 프로토콜을 지정해야 한다.

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

int socket(int domain, int type, int protocol);
-> 성공 시 파일 디스크립터, 실패 시 -1 반환
```

  - domain: IPv4, IPv6, IPX, low level socket 등 설정
  - type: TCP, UDP 설정
  - protocol: 특정 프로토콜 사용을 지정 (보통 0)

  - 매개변수(domain, type, protocol) 모두가 프로토콜 정보와 관련이 있음

```
int sock;
sock = socket(PF_INET, SOCK_STREAM, 0);
```

프로토콜 체계(Protocol Family)

- 프로토콜 체계
  - 프로토콜의 종류에 따라서 부류가 나눔
    - 부류를 프로토콜 체계라고 함
  - PF_INET은 IPv4 인터넷 프로토콜 체계를 의미
    - IPv4 를 기반으로 소켓 프로그래밍을 학습
    - domain 파라미터에 적용

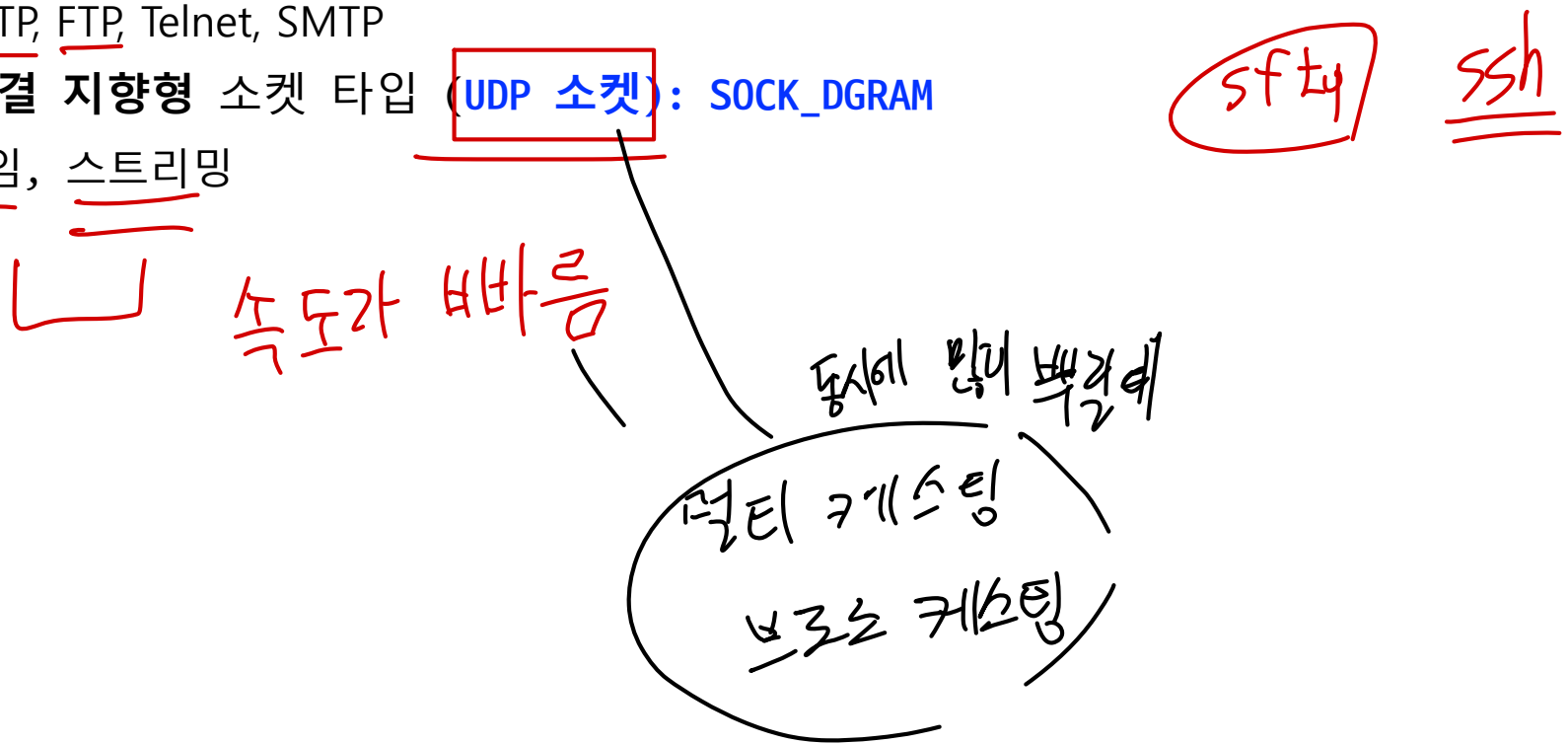
- 대표적인 프로토콜 체계

이름	프로토콜 체계(Protocol Family)
PF_INET	• IPv4 인터넷 프로토콜 체계
PF_INET6	• IPv6 인터넷 프로토콜 체계
PF_LOCAL	• 로컬 통신을 위한 UNIX 프로토콜 체계
PF_PACKET	• Low Level 소켓을 위한 프로토콜 체계
PF_IPX	• IPX 노벨 프로토콜 체계

소켓의 타입

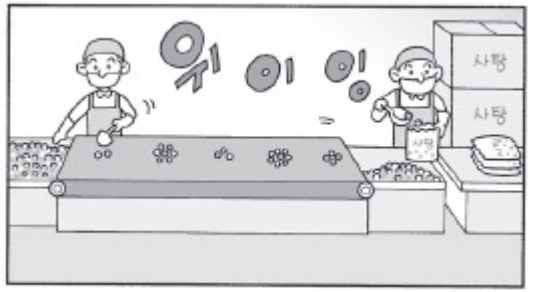
- 소켓의 타입
  - 데이터 전송방식을 의미함
  - 소켓이 생성될 때 소켓의 타입도 결정되어야 함

- 프로토콜 체계 PF_INET의 대표적인 소켓 타입
  - **연결 지향형** 소켓 타입(TCP 소켓): SOCK_STREAM
    - HTTP, FTP, Telnet, SMTP
  - **비 연결 지향형** 소켓 타입 (UDP 소켓): SOCK_DGRAM
    - 게임, 스트리밍



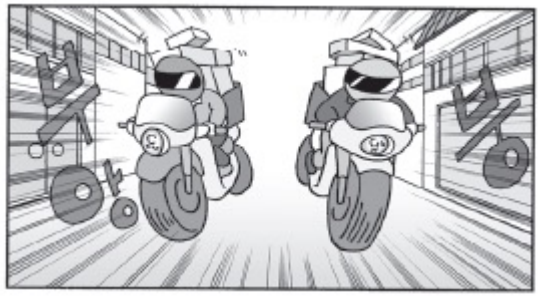
두 타입의 소켓

- 연결지향형 소켓(SOCK_STREAM)의 데이터 전송 특성
  - TCP 소켓
  - 중간에 데이터가 소멸되지 않음 ∨ 재전송 요청
  - 전송 순서대로 데이터가 수신
  - 소켓 대 소켓의 연결은 반드시 1대 1의 구조
  - 신뢰성 있는 순차적인 바이트 기반



TCP 데이터 전송특성

- 비 연결지향형 소켓(SOCK_DGRAM)의 데이터 전송 특성
  - UDP 소켓
  - 전송 순서에 상관없이 빠른 속도의 전송을 지향 ∨ 순서 보장 안됨
  - 데이터 손실 및 파손의 우려
  - 데이터의 경계가 존재
  - 한번에 전송할 수 있는 데이터의 크기가 제한



UDP 데이터 전송특성 ∨

프로토콜의 선택

- TCP 소켓 사용
  - IPv4 인터넷 프로토콜 체계에서 동작하는 연결지향형 데이터 전송 소켓

```
int tcp_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

- UDP 소켓 사용
  - IPv4 인터넷 프로토콜 체계에서 동작하는 비 연결지향형 데이터 전송 소켓

```
int ucp_socket = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

- 프로토콜 선택
  - 첫 번째, 두 번째 인자로 전달된 정보를 통해서 소켓의 프로토콜이 결정
  - 세 번째 인자로 0을 전달해도 됨
    - IPPROTO_TCP, IPPROTO_UDP 대신 0을 사용

데이터 경계

- TCP 소켓
  - 전송되는 데이터의 경계(boundary)가 존재하지 않음
  - 데이터 전송 회수와 수신 회수가 일치하지 않음
    - 1회 최대 전송 크기인 MTU(Maximum Transmission Unit) 만큼 전송됨
    - Ethernet: 1500 bytes

- UDP 소켓
  - 전송되는 데이터의 경계(boundary)가 존재함
  - 데이터 전송 회수와 수신 회수가 일치해야 됨

택배

```
$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 155.230.120.234  netmask 255.255.255.0  broadcast 155.230.120.255
        inet6 fe80::8946:6f4f:b082:1ebe  prefixlen 64  scopeid 0x20<link>
        ether 48:5b:39:8a:d2:e2  txqueuelen 1000  (Ethernet)
        RX packets 33352629  bytes 2565805282 (2.5 GB)
        RX errors 0  dropped 1438319  overruns 0  frame 0
        TX packets 376489  bytes 55868418 (55.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 3  collisions 0
```

연결지향형 소켓: TCP 소켓의 예

```
if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))==-1)
    error_handling("bind() error");
if(listen(serv_sock, 5)==-1)
    error_handling("listen() error");
clnt_addr_size=sizeof(clnt_addr);

clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr,&clnt_addr_size);

if(clnt_sock==-1)
    error_handling("accept() error");

write(clnt_sock, message, sizeof(message));
close(clnt_sock);
close(serv_sock);
```

tcp_server.c의 데이터 전송
(hello_server.c와 동일)

char message[] = “Hello World!” 전송

1 byte씩 읽어옴
read_len==0: 데이터가 없음

tcp_client.c의
데이터 수신

```
if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
    error_handling("connect() error!");

while(read_len=read(sock, &message[idx++], 1))
{
    if(read_len==-1)
    {
        error_handling("read() error!");
        break;
    }
    str_len+=read_len;
}

printf("Message from server: %s \n", message);
printf("Function read call count: %d \n", str_len);
```

tcp_server.c tcp_client.c 동작 과정

- TCP
  - 송신 횟수와 수신 횟수가 일치하지 않음

13 bytes 데이터

tcp_server.c

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
H	e	l	l	o		W	o	r	l	d	!	\0

1번 전송

```
write(clnt_sock, message, sizeof(message))
```

한 번에 모든 데이터 전송 (13bytes)

서버 → 클라이언트

tcp_client.c

수신 버퍼
(소켓 버퍼)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
H	e	l	l	o		W	o	r	l	d	!	\0

13 bytes ⇒ 큐 (소켓 버퍼 저장)

```
read(sock, &message[idx++], 1)
```

1 byte씩 수신 버퍼에서 읽음
(총 13번의 read()함수 호출)

```
Message from server: Hello World!
Function read call count: 13
```

tcp_server.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>

void error_handling(char *message);

int main(int argc, char *argv[])
{
    int serv_sock;
    int clnt_sock;

    struct sockaddr_in serv_addr;
    struct sockaddr_in clnt_addr;
    socklen_t clnt_addr_size;

    char message[]="Hello World!";
    if(argc!=2){
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }
    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    if(serv_sock == -1)
        error_handling("socket() error");
    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(argv[1]));

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

    if(listen(serv_sock, 5)==-1)
        error_handling("listen() error");

    clnt_addr_size=sizeof(clnt_addr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr,&clnt_addr_size);

    if(clnt_sock==-1)
        error_handling("accept() error");

    write(clnt_sock, message, sizeof(message));

    close(clnt_sock);
    close(serv_sock);
    return 0;
}

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

tcp_client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>

void error_handling(char *message);

int main(int argc, char* argv[])
{
    int sock;
    struct sockaddr_in serv_addr;
    char message[30];
    int str_len=0;
    int idx=0, read_len=0;
    if(argc!=3){
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock=socket(PF_INET, SOCK_STREAM, 0);
    if(sock == -1)
        error_handling("socket() error");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_addr.sin_port=htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
        error_handling("connect() error!");

    while(read_len=read(sock, &message[idx++], 1)) {
        if(read_len==-1)
            error_handling("read() error!");
        str_len+=read_len;
    }

    printf("Message from server: %s \n", message);
    printf("Function read call count: %d \n", str_len);
    close(sock);
    return 0;
}

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

한 바이트씩 데이터를 읽음

누적 ← 실제 읽은 바이트수

Questions?

LMS 질의 응답 게시판에 올려주세요.