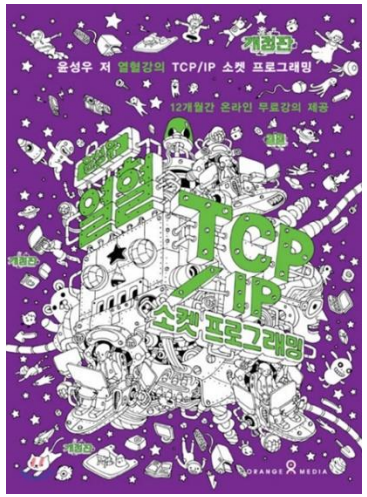
Network Programing

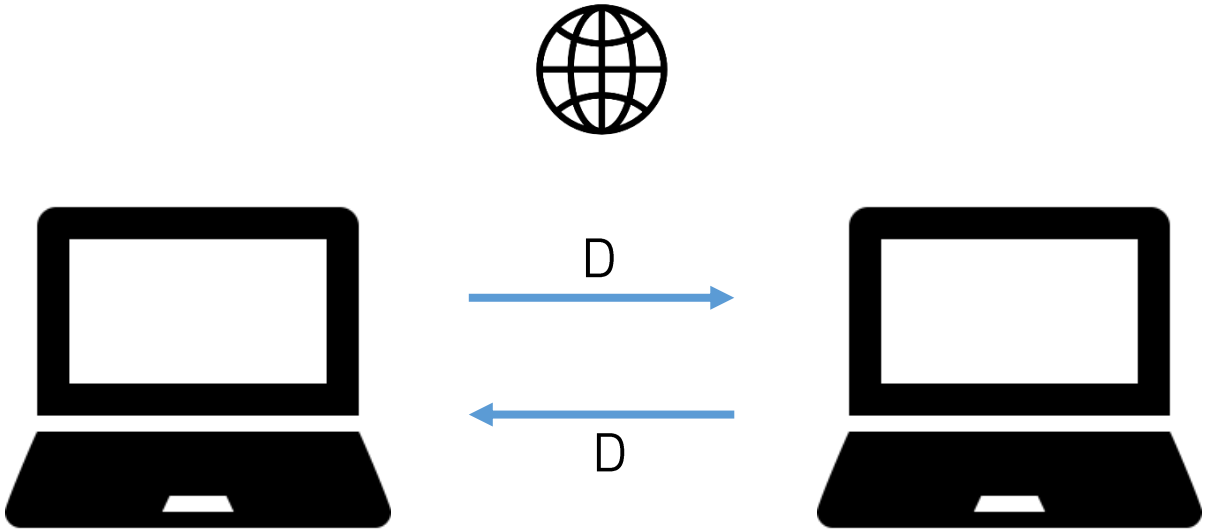
목차

소켓?



소켓을 이용한
프로그램

Network Programing?



SOCKET : 네트워크에서 데이터를 통신할 수 있도록 연결해주는 연결부

Linux

파일 조작 = 소켓 조작

소켓을 파일의 일종으로 구분

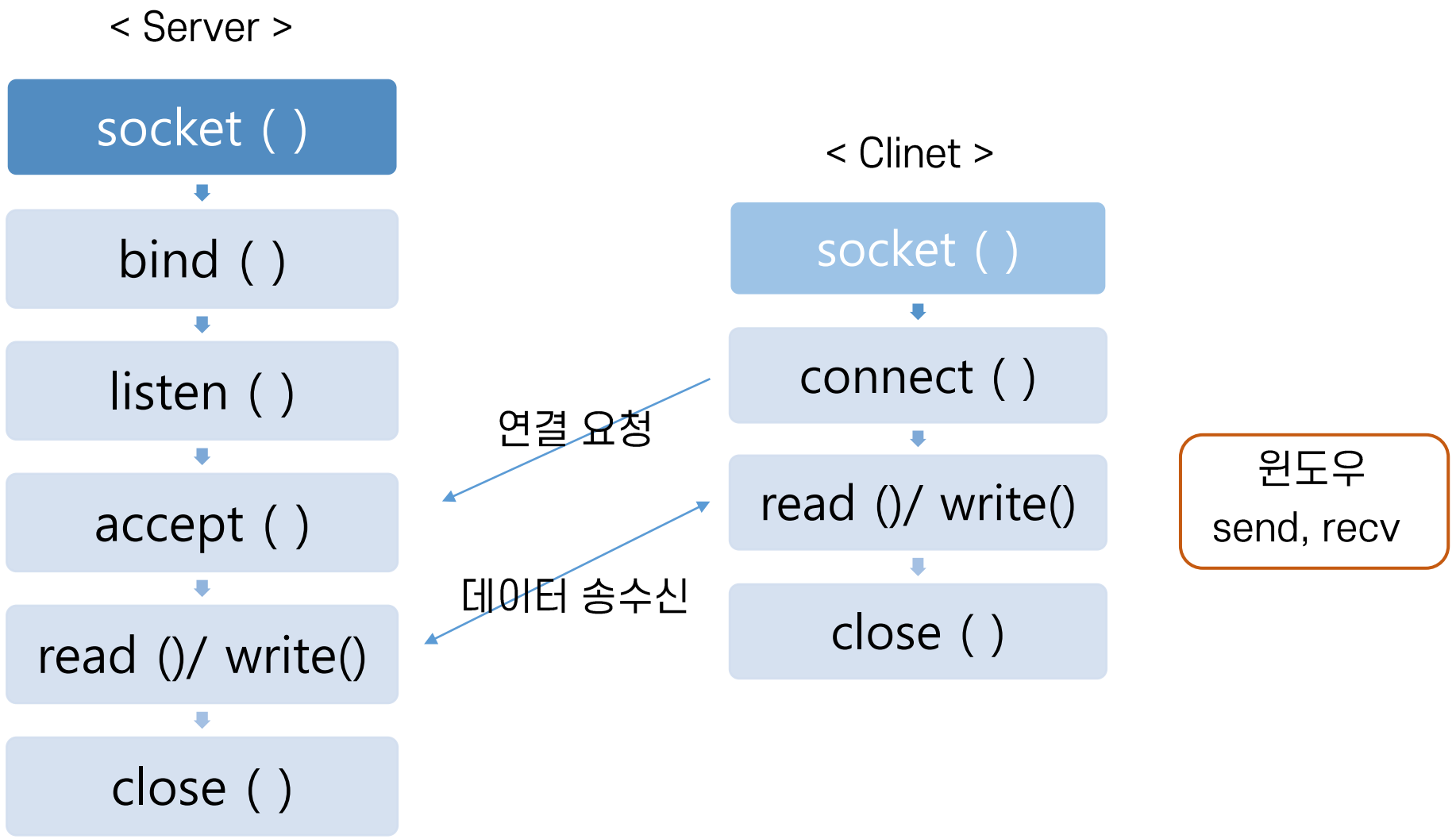
파일 입출력 함수 사용 가능

Window

파일 조작 ≠ 소켓 조작

별도의 데이터 송수신 함수 참조

함수 흐름


< Server >
socket ( )
bind ( )
listen ( )
accept ( )
read ()/ write()
close ( )
연결 요청
데이터 송수신
< Clinet >
socket ( )
connect ( )
read ()/ write()
close ( )
윈도우
send, recv

Linux

```
#include <sys/socket.h>      // 네트워크 통신을 위한 소켓 인터페이스를 위한 자료형, 구조체, 함수들을 정의
int socket (int domain, int type, int protocol);
```

// 소켓이 사용할 프로토콜 체계 정보 전달

-> 성공 : 파일 디스크립터, 실패 : -1

이름	프로토콜 체계
PF_INET	IPv4 인터넷 프로토콜 체계
PF_INET6	IPv6 인터넷 프로토콜 체계
그	외

Linux

```
#include <sys/socket.h>      // 네트워크 통신을 위한 소켓 인터페이스를 위한 자료형, 구조체, 함수들을 정의
int socket (int domain, int type, int protocol);
                                  // 소켓 데이터 전송 방식
```

-> 성공 : 파일 디스크립터, 실패 : -1

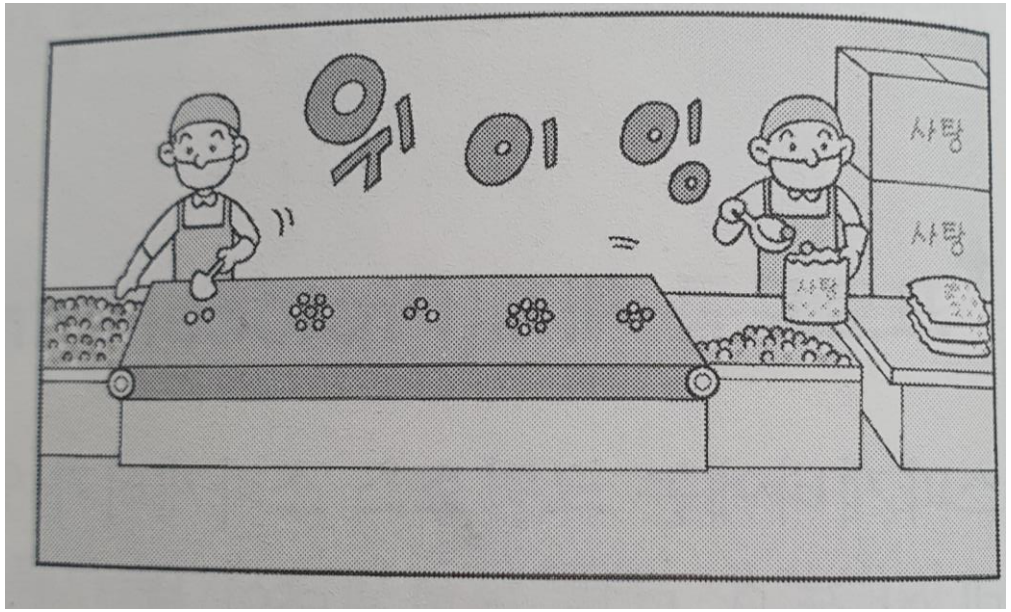
Why?

프로토콜 체계에도 둘 이상의 데이터 전송방식이 존재

Socket type

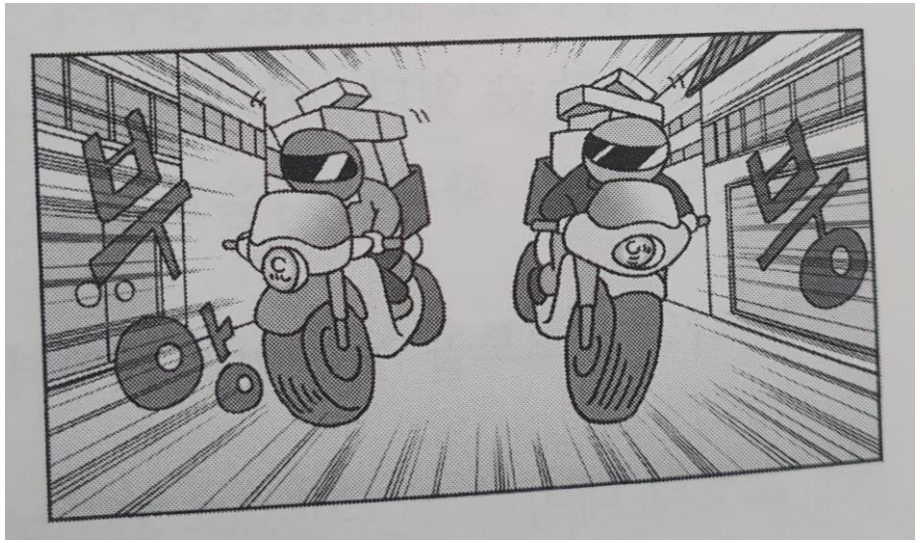
연결지향형 소켓, 비 연결지향형 소켓

Socket type - 연결지향형 소켓(SOCK_STREAM)



- ✓ 중간에 데이터가 소멸되지 않고 목적지로 전송
- ✓ 전송 순서대로 데이터 수신
- ✓ 전송되는 데이터의 경계가 존재하지 않음

Socket type - 비 연결지향형 소켓(SOCK_DGRAM



- ✓ 전송 순서에 상관없이 가장 빠른 전송 지향
- ✓ 전송된 데이터는 손실, 파손의 우려가 있음
- ✓ 전송되는 데이터의 경계가 존재
- ✓ 한번에 전송할 수 있는 데이터의 크기 제한

Linux

```
#include <sys/socket.h>      // 네트워크 통신을 위한 소켓 인터페이스를 위한 자료형, 구조체, 함수들을 정의
int socket (int domain, int type, int protocol);
                                        // 실제 소켓이 사용할 최종 프로토콜 정보
```

-> 성공 : 파일 디스크립터, 실패 : -1

✓ 대부분 0 넘겨줘도 원하는 소켓 생성 가능

Why?

소켓 타입이 같지만, 그 안에서 프로토콜이 다시 나뉘는
상황이 있을 수도 있어서 정보 구체화

Window

```
#include <winsock2.h>    // 윈도우에서 네트워크 프로그래밍을 지원해주는 라이브러리
SOCKET socket (int af, int type, int protocol);
```

// 정수로 표현된 소켓의 핸들 값을 저장하기 위해 정의된 자료형

-> 성공 : 소켓 핸들, 실패 : INVALID_SOCKET

Linux

파일 디스크립터?

시스템으로부터 할당 받은 파일 또는 소켓에 부여된 정수

즉, 운영체제가 만든 파일, 소켓의 지칭을 편히 하기 위해 부여된 숫자

파일 디스크립터	대 상
0	표준 입력
1	표준 출력
2	표준 에러

Linux

- ✓ open : 파일 열기
- ✓ close : 파일 닫기
- ✓ wirte : 데이터 쓰기
- ✓ read : 데이터 읽기

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/socket.h>

int main(void)
{
        int fd1, fd2, fd3;
        fd1=socket(PF_INET, SOCK_STREAM, 0);
        fd2=open("test.dat", O_CREAT|O_WRONLY|O_TRUNC);
        fd3=socket(PF_INET, SOCK_DGRAM, 0);

        printf("file descriptor 1: %d\n", fd1);
        printf("file descriptor 2: %d\n", fd2);
        printf("file descriptor 3: %d\n", fd3);

        close(fd1);
        close(fd2);
        close(fd3);
        return 0;
```

```
┌──(hae㉿kali)-[~/문서/socket]
└─$ ./fds
file descriptor 1: 3
file descriptor 2: 4
file descriptor 3: 5
```

Window

-> 성공 : 전송된 바이트 수, 실패 : SOKET_ERROR

```
int send ( SOCKET s, const char * buf, int len, int flags);
int recv ( SOCKET s, const char * buf, int len, int flags);
```

-> 성공 : 수신된 바이트 수, 실패 : SOKET_ERROR

- ✓ s : 데이터 전송 대상과의 연결을 의미하는 소켓 핸들 값
- ✓ buf : 전송할 데이터를 저장하고 있는 버퍼의 주소 값
- ✓ len : 전송할 바이트 수
- ✓ flags : 데이터 전송 시 적용할 다양한 옵션 정보

Linux / Window

```
#include <sys/socket.h>
int bind (int sockfd, struct sockaddr *myaddr, socklem_t addrlen);
```

-> 성공 : 0, 실패 : -1

// 주소 정보를 할당할 소켓의 파일 디스크립터

// 할당하려는 주소 정보를 지니는 구조체 변수의 주소 값

// 두번째 인자로 전달된 구조체 변수의 길이 정보

주소 정보

- ✓ 주소 체계 정보
- ✓ IP 주소
- ✓ Port 번호

```
struct sockaddr_in {
        sa_family_t     sin_family;  // 주소체계
        uint16_t        sin_port;    // port 번호
        struct in_addr  sin_addr;    // ip 주소 정보
        char            sin_zero[8]; // 0으로 채워야함
}
```

Linux / Window

```
#include <sys/socket.h>
int listen (int sock, int backlog);  // 연결 요청 대기 수 크기 정보 전달
```

// 대기 상태로 할 소켓의
파일 디스크립터 전달

-> 성공 : 0, 실패 : -1

Linux / Window

```
#include <sys/socket.h>
int connect (int sock, struct sockaddr * servaddr, socklen_t addrlen);
```

// 두 번째 매개변수에 전달된
주소의 변수 크기

// 클라이언트 소켓의
파일 디스크립터 전달

// 연결 요청할 서버의 주소 정보를
담은 변수의 주소 값

-> 성공 : 0, 실패 : -1

Linux / Window

```
#include <sys/socket.h>
int accept (int sock, struct sockaddr * addr, socklen_t addrlen);
```

// 서버 소켓의
파일 디스크립터 전달

// 연결 요청을 한 클라이언트의 주소 정보를
담을 변수의 주소 값

// 두 번째 매개변수에 전달된 주소의 변수
크기를 바이트 단위로 전달

Window

```
#include <winsock2.h>
int closesocket (SOCKET s);
```

-> 성공 : 0, 실패 : SOCKET_ERROR

실습 - 리눅스(Server)

```c
int main(int argc, char *argv[]) //실행할 때 들어오는 문자열
{
    int serv_sock;
    int clnt_sock;

    struct sockaddr_in serv_addr; //서버 주소 정보
    struct sockaddr_in clnt_addr; //클라이언트 주소 정보
    socklen_t clnt_addr_size; //사이즈

    char message[]="Hello World!"; //성공 시 출력해 줄 문구

    if(argc!=2){
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }
```

```
┌──(hae㉿kali)-[~/문서/socket]
└─$ ./hserver 9190
```

실습 - 리눅스(Server)

✓ 소켓 생성

```
serv_sock=socket(PF_INET, SOCK_STREAM, 0); // IPv4 프로토콜 체계에서 동작하는 연결지향형 소켓
if(serv_sock == -1) //소켓 생성 실패하면
    error_handling("socket() error"); //에러 문구 출력
```

실습 - 리눅스(Server)

```
//주소 초기화
memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family=AF_INET; //IPv4 인터넷 프로토콜에 적용하는 주소체계
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
serv_addr.sin_port=htons(atoi(argv[1]));

//ip, port 할당
if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))==-1 )
    error_handling("bind() error");
```

- ✓ memset(초기화 대상, 초기화할 값,크키)
- ✓ htonl : long 형 데이터를 호스트 바이트 순서에서 네트워크 바이트 순서로 변환
- ✓ INADDR_ANY: 서버의 ip 주소를 자동으로 찾아서 대입
- ✓ atoi : 문자열 정수로 변환

실습 - 리눅스(Server, Client)

```
//연결 요청 받아들일 수 있는 상태로 변경
if(listen(serv_sock, 5)==-1) //대기 큐 5개
    error_handling("listen() error"); //실패하면 에러메시지
```

Client

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

Server

```
//클라이언트에게 요청 오면 연결 수락
clnt_addr_size=sizeof(clnt_addr);
clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr,&clnt_addr_size); //연결요청 수락,
//연경 요청 없는데 호출되면 있을 때까지 반환 X
if(clnt_sock==-1)
    error_handling("accept() error");
```

실습 – 리눅스(Server, Client)

Server

```
    //데이터 전송
    write(clnt_sock, message, sizeof(message));

    //소켓 닫음
    close(clnt_sock);
    close(serv_sock);
    return 0;
}
```

```
char message[]="Hello World!"; //성공 시 출력해 줄 문구
```

Client

```
//파일 읽기
str_len=read(sock, message, sizeof(message)-1);
if(str_len==-1)
    error_handling("read() error!");

printf("Message from server: %s \n", message);
close(sock);
return 0;
```

```
┌──(hae㉿kali)-[~/문서/socket]
└─$ ./hclient 127.0.0.1 9190
Message from server: Hello World!
```

감사합니다.

정보보호학전공

91914145 장혜선

04. 01 발표