Understanding of Socket and File I/O

Department of Computer Engineering

Kyung Hee University.

Choong Seon Hong

컴퓨터 통신 프로토콜

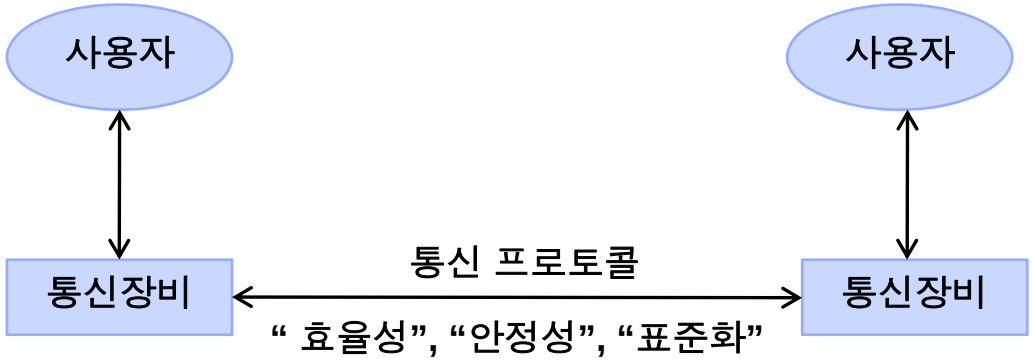
컴퓨터 통신 프로토콜

- 데이터를 원활이 주고 받을 수 있도록 정한 약속
- 컴퓨터 네트워크 프로토콜
  - 통신 장비는 서로간의 통신 방법이 미리 정의 되어 있어야 함
  - 같은 통신 프로토콜을 지원하는 장비 간에만 통신이 가능
- 컴퓨터 통신은 네트워크 형태로 운영
  - 컴퓨터 네트워크 프로토콜이라 부름

통신 프로토콜의 특징

통신 프로토콜

- ‘효율적’, ‘안정적’으로 통신할 수 있도록 미리 정한 약속

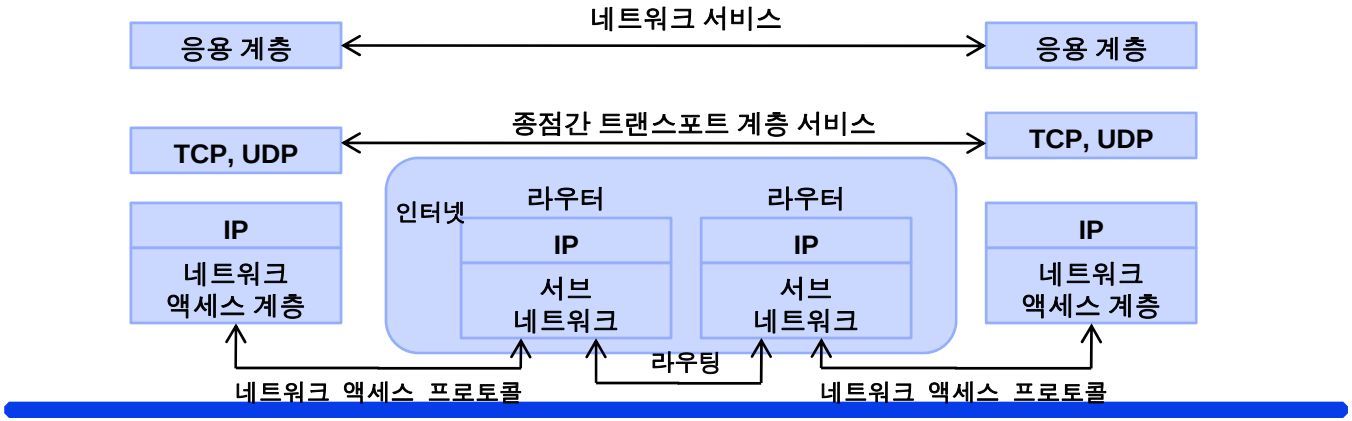


- 효율적
  - 주어진 통신 채널을 최대로 이용할 수 있어야 한다.
- 안정적
  - 비정상적인 장애 발생시에도 안정되게 동작해야 한다.
- 표준화
  - 널리 사용되기 위해서는 미리 표준화 되어야 한다.

인터넷과 서브네트워크

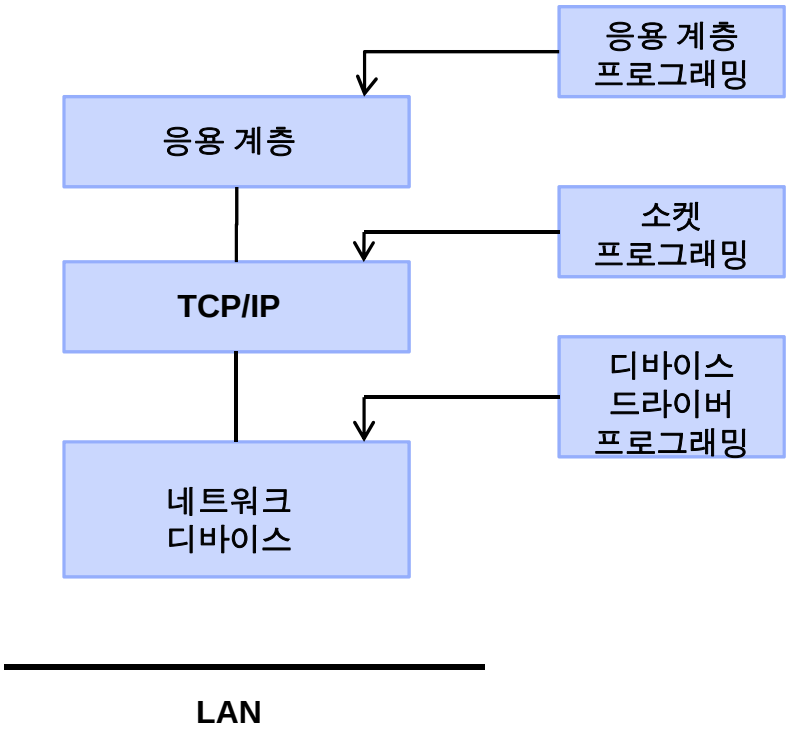
인터넷을 이용한 두 호스트 사이의 통신

- 인터넷을 이용하기 위해 TCP/IP 프로토콜이 설치되어 있어야 함
- 호스트는 라우터들을 경유하여 서로 연결
- 호스트
  - 인터넷에 물리적으로 접속하기 위해 네트워크 액세스 프로토콜이 필요함
  - 네트워크 액세스 프로토콜 : IP 계층이 서브네트워크 이용을 위한 프로토콜
- 서브네트워크
  - 이더넷, 패킷 교환망, DSL, ATM 등 데이터를 실제로 전달해 주는 네트워크



네트워크 프로그래밍 계층별 분류

- 응용계층 프로그래밍
- 트랜스포트 계층 프로그래밍
- 디바이스 드라이버 계층 프로그래밍



응용 계층 프로그래밍

- 네트워크 응용 패키지, TCP/IP 응용 프로토콜을 이용
- 데이터의 송수신을 구체적으로 다루지 않음
  - 유닉스의 rsh, rcp
  - RPC
  - http
- 장점
  - 복잡한 네트워크 서비스를 편리하게 구현할 수 있음
- 단점
  - 하위 계층의 구체적인 동작을 직접 제어할 수 없음

트랜스포트 계층 프로그래밍

- TCP나 UDP와 같은 트랜스포트 계층의 기능을 직접 이용
- 데이터 그램 단위의 데이터 송수신 처리
- TCP를 이용할 경우 연결설정, 흐름제어, 에러제어가 가능
- 대표적인 API (Application Program Interface)
  - UNIX BSD socket, winsock
- 인터넷 응용 프로그래밍의 기초

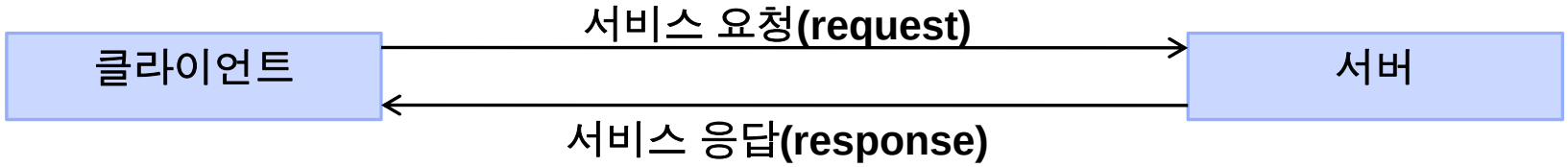
디바이스 드라이버 계층 프로그래밍

- OSI 계층 2 이하의 인터페이스를 직접 다루는 프로그래밍
  - 예) Microsoft 사의 NDIS (Network Driver Interface Specification) API
    - Windows에서 이더넷 프레임의 송수신을 직접 처리
- 프레임 단위의 송수신을 구체적으로 제어, 네트워크 상태 모니터링 등
  - LAN 카드 개발 시 테스트 프로그램 작성에 사용
- 프레임을 송수신하는 기능만을 이용
  - 흐름제어, 에러제어, 인터넷주소 관리 등의 기능은 별도로 구현해야 함

클라이언트-서버 모델

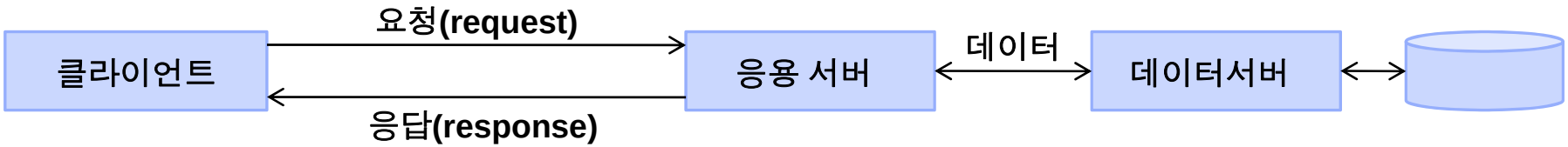
- 대부분의 네트워크 프로그램의 구현 모델
- 서버와 클라이언트
  - 서버 : 서비스를 제공하는 장비
  - 클라이언트 : 서비스를 이용하는 장비
- 서버
  - 일반적으로 클라이언트보다 구현이 복잡
    - 클라이언트의 인증
    - 정보보호
    - 동시 서비스
    - 안정성

2-tier 클라이언트-서버 모델



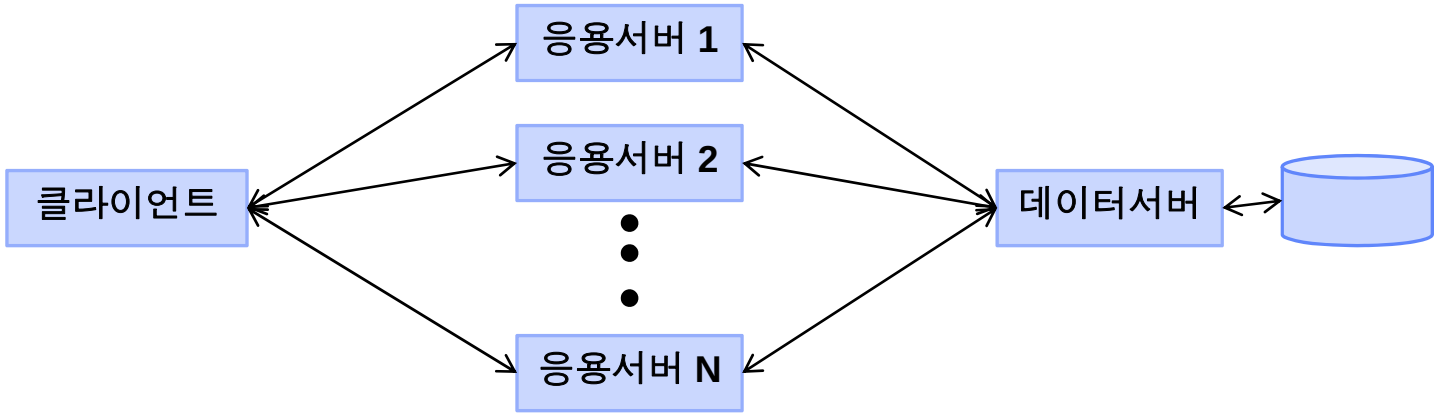
- 대부분의 통신 프로그램
  - 웹(http), ftp, telnet, mail
- 단점
  - 서버에서의 병목 현상
  - 클라이언트의 증가는 서버에 트래픽 집중과 처리 용량 부족 현상 발생

3-tier 클라이언트 서버 모델



- 2-tier 클라이언트-서버 모델의 문제점을 개선한 구조
- '응용서버'와 '데이터서버'로 구분
  - 클라이언트는 응용 서버에 서비스를 요청
  - 응용서버는 데이터 서버로부터 데이터를 얻어 클라이언트에 응답
- 장점
  - 클라이언트는 데이터서버의 정보가 필요없음
  - 클라이언트가 같은 요청을 동시에 하면 응용 서버는 이를 한번만 처리
    - 데이터 서버의 통신 부담 저하

n-tier 클라이언트-서버 모델



- 3-tier 모델을 확장
  - 여러 버전의 응용서버가 존재
  - 기본 동작은 3-tier 모델과 같지만 응용 서버가 여러 형태로 구현
- 장점
  - 클라이언트가 필요에 따라 다른 응용 서버를 선택할 수 있음
  - 서비스 제공 도중 새로운 응용서버 추가가 가능
  - 서비스의 다양성, 확장성, 버전관리 (업그레이드)

P2P 모델

- 서버, 클라이언트의 역할이 미리 정해지지 않음
  - 경우에 따라 서버 또는 클라이언트가 될 수 있음
- 내부적으로는 클라이언트-서버 모델로 동작
  - 참가자가 필요한 시기에 서버와 클라이언트의 역할을 수행
- 순수 P2P 모델
  - 참가자들이 동등한 자격으로 정보를 이용하는 모델
  - 정보 검색을 위해 주변의 참가자에게 문의 (요청)
    - 요청은 정보를 찾을 때까지 인접한 참가자에게 계속 전파
  - 특징
    - 동작이 단순, 비효율적
- 하이브리드형 P2P 모델
  - 순수 P2P 모델을 개선한 방식
  - 인덱스 서버가 존재
    - 정보 검색을 위해 인덱스 서버에게 문의 (요청)
    - 인덱스 서버는 요청한 정보를 가지고 있는 참가자의 주소를 반환

서버 구현 기술

- 서버의 구현 방식이 네트워크 성능에 중요한 영향을 줌
- 서버 구현 기술
  - 연결형과 비연결형 서버
  - iterative와 concurrent 서버

연결형과 비연결형 서버

- 연결형 서비스
  - 종점간 연결 설정/해제, 데이터 송수신 등 세 단계의 절차를 거침
  - 주로 TCP를 이용하여 작성
  - 데이터의 안정적인 전달을 보장
  - TCP 프로토콜, telnet, ftp 등
  - 각 클라이언트마다 연결을 개설
    - 클라이언트 수가 증가하면 서버에 부담이 클 수 있음
- 비연결형 서비스
  - 종점간 연결 설정/해제 작업 없이 바로 데이터를 주고 받는 방식
  - 주로 UDP를 이용하여 작성
  - 안정적인 데이터 전달을 보장하지 않으므로 응용 프로그램에서 처리
  - 클라이언트마다 연결을 설정할 필요가 없음
    - 서버의 부담이 적음 (메모리 사용 등)
  - 방송형, 멀티캐스팅형 서비스에 적합

Iterative와 Concurrent 서버

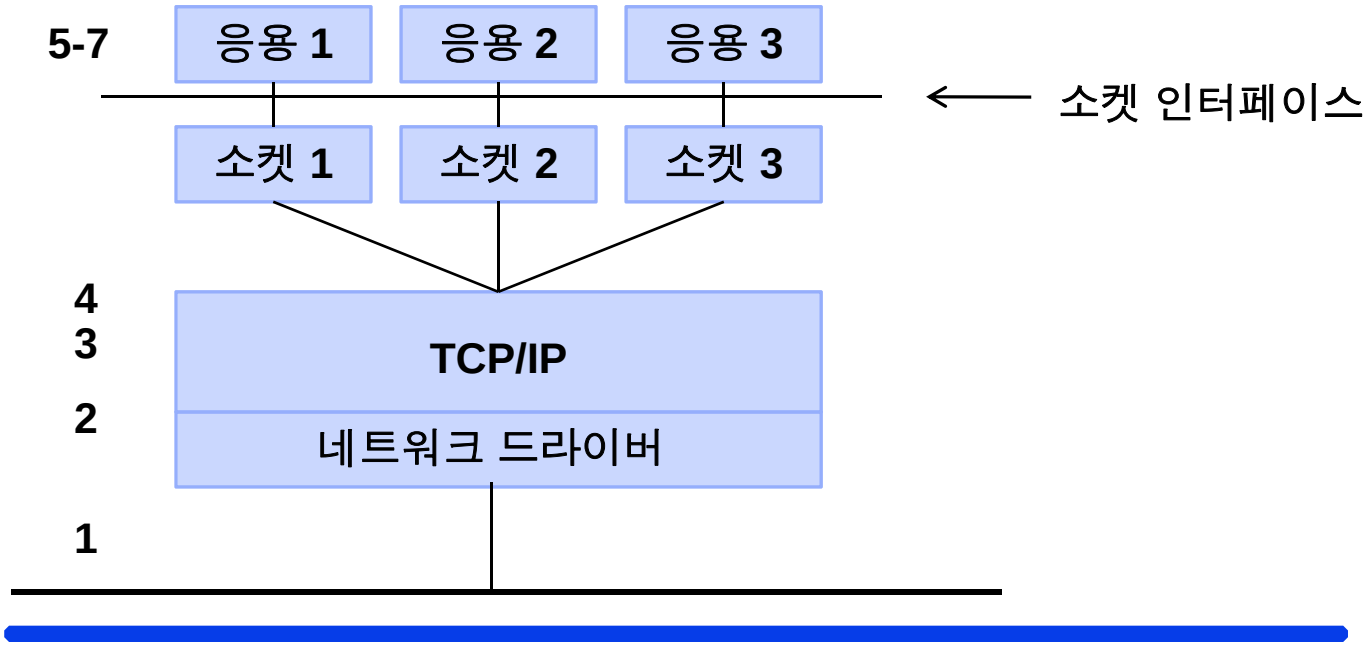
- Iterative 서버
  - 클라이언트의 요청을 순서대로 처리
  - 클라이언트의 요청이 짧은 시간에 처리할 수 있는 경우에 적합
  - Concurrent 서버보다 구현이 간단함
- Concurrent 서버
  - 클라이언트의 요청을 동시에 처리
  - 다중처리 기능이 필요
    - 예) 멀티프로세스에 의한 다중처리
      - 새로운 클라이언트 접속시 이 클라이언트를 담당하는 프로세스 생성
      - 클라이언트의 증가에 따라 프로세스 수도 증가
    - 다중화 이용
      - 여러 작업을 동시에 처리하는 기능
    - 각 서비스 처리 시간이 불규칙적이거나 길 때 필요

네트워크 프로그램 영역

- 네트워크 프로그램의 구현 영역에 따른 분류
  - 컴퓨터 사이의 데이터 통신 (data communication)
    - TCP/IP 등의 통신 프로토콜을 이용한 패킷 송수신
  - 컴퓨터 내의 데이터 처리 (data processing)
    - 파일 입출력, 데이터베이스 액세스, 캐싱, 멀티미디어 신호 처리 등
  - 컴퓨터 내의 프로세스간 통신 (inter-process communication)
    - 멀티프로세스, 멀티스레드
    - 프로세스 작업 분담 기술 등

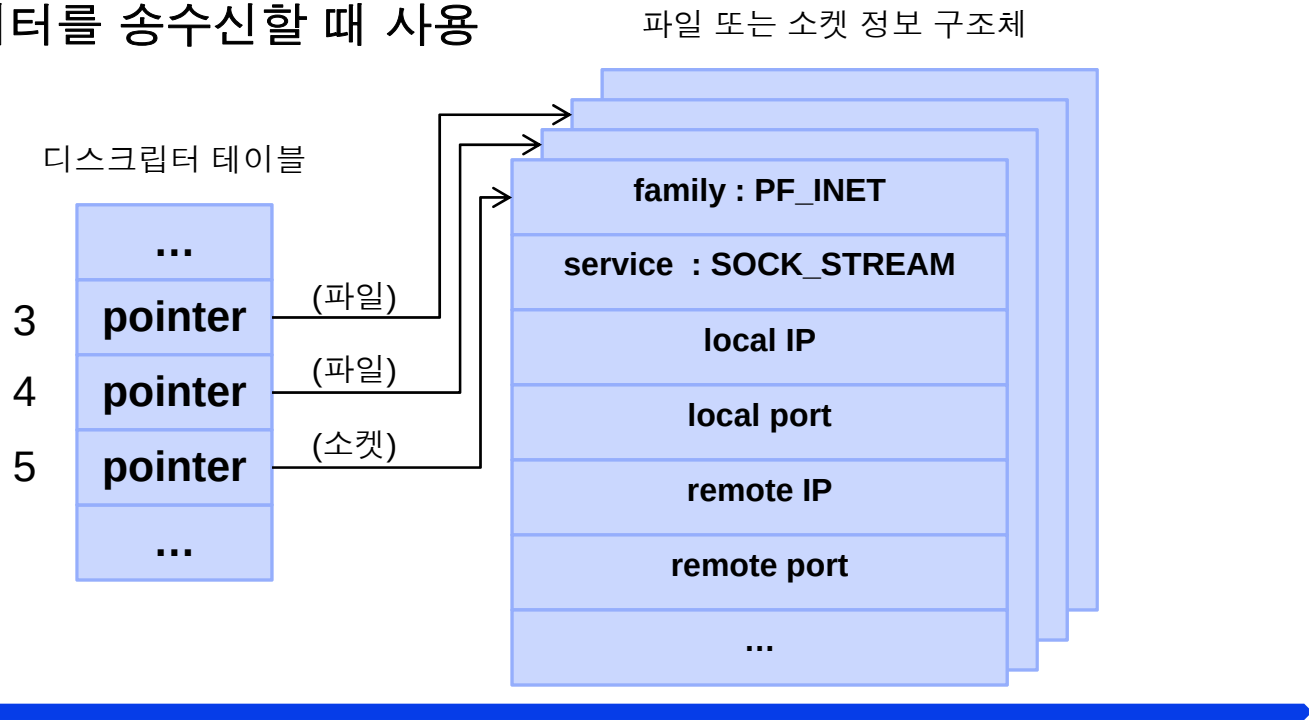
소켓 정의

- TCP나 UDP와 같은 트랜스포트 계층을 이용하는 API
  - 1982년 BSD 유닉스 4.1에서 소개
  - 모든 유닉스 운영체제에서 제공
  - Windows는 Winsock으로 제공
  - Java는 Network 관련 클래스 제공



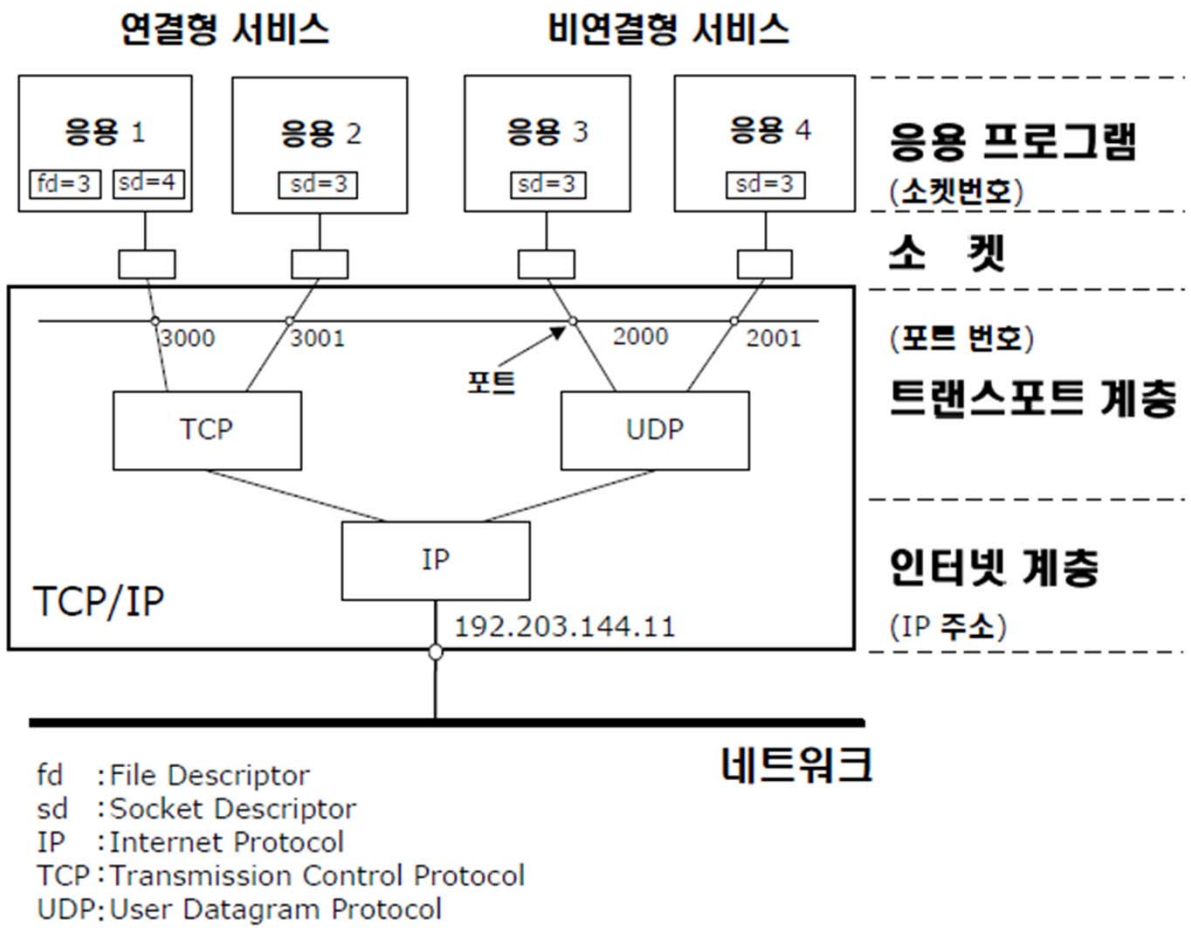
소켓 번호

- 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

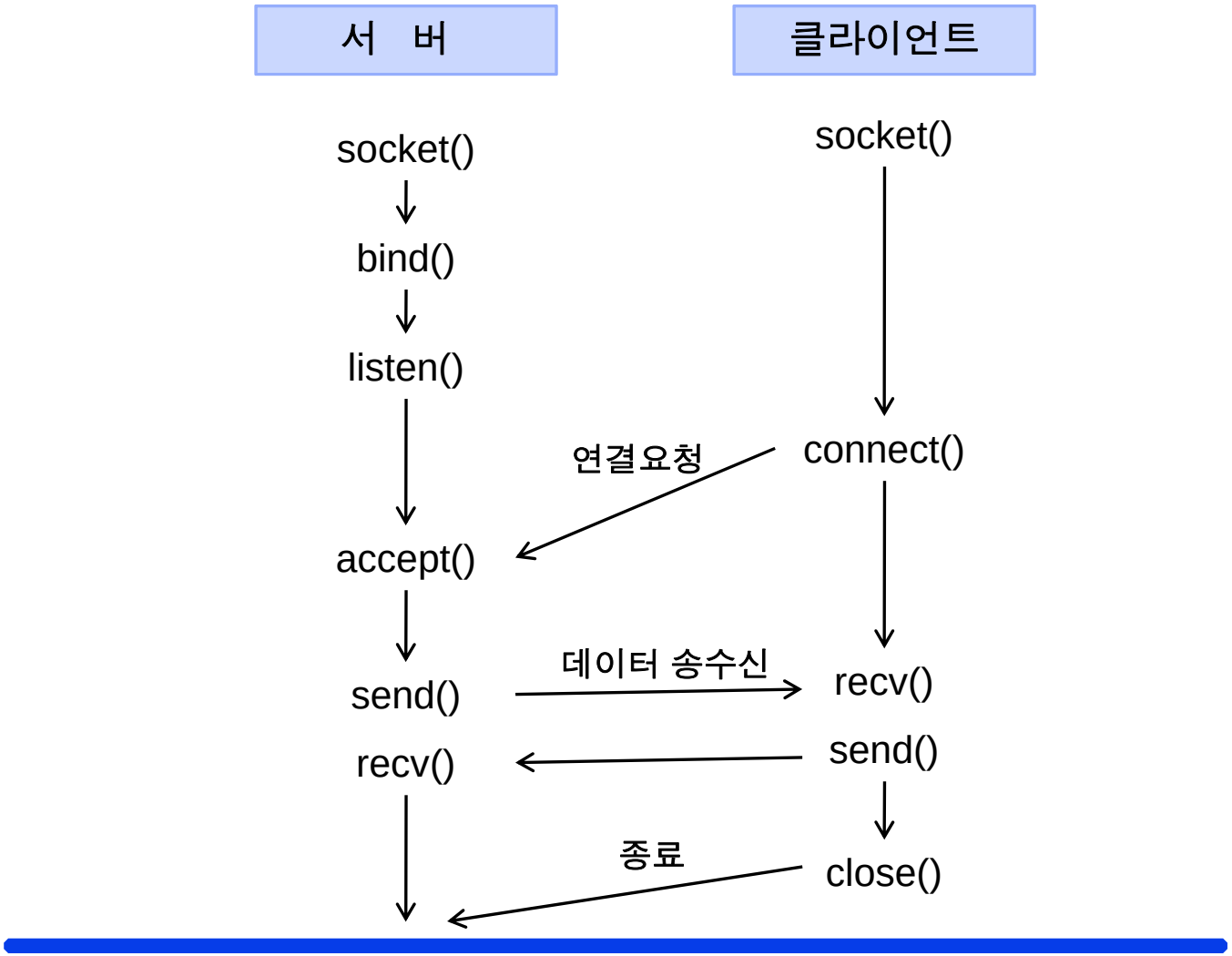
```
PF_INET          // 인터넷프로토콜 체계
PF_INET6         // IPv6 프로토콜 체계
PF_UNIX          // 유닉스 방식의 프로토콜 체계
PF_NS            // XEROX 네트워크 시스템의 프로토콜 체계
PF_PACKET        // 리눅스에서 패킷 캡쳐를 위해 사용
```

  - 서비스 타입

```
SOCK_STREAM      // TCP 소켓
SOCK_DGRAM       // UDP 소켓
SOCK_RAW         // RAW 소켓
```

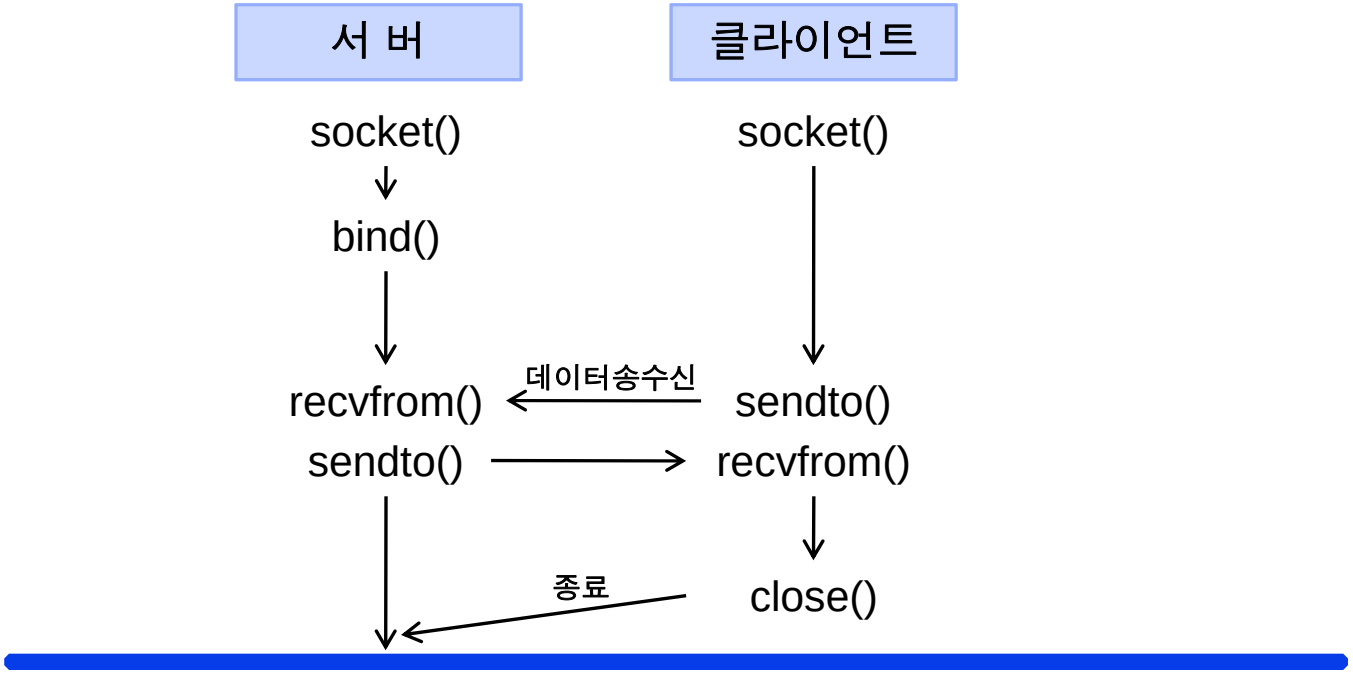
소켓 사용 절차

- TCP (연결형) 소켓 프로그래밍 절차



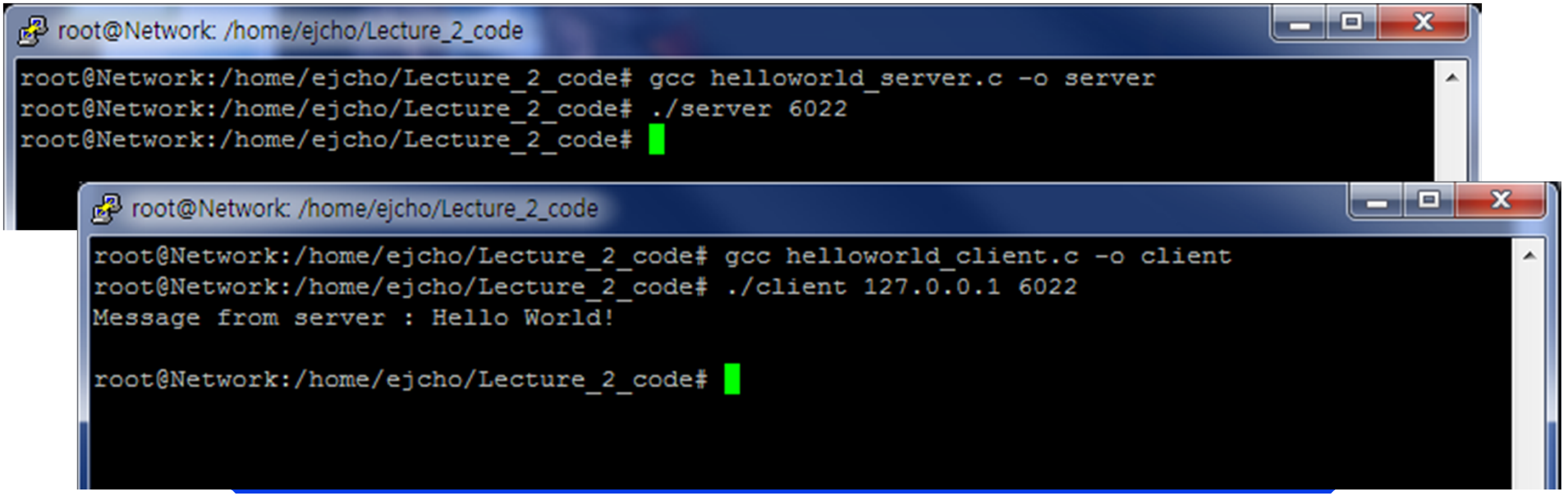
소켓 사용 절차

- UDP (비연결형) 소켓 프로그래밍 절차
  - TCP와 달리 일 대 일 통신에만 사용되지 않음
  - 비연결형 소켓
    - Connect() 시스템 콜을 사용할 필요가 없음
    - 소켓 개설 후 바로 상대방과 데이터를 송수신



“Hello World!” 서버 / 클라이언트

- 프로그램 예제
  - helloworld_server.c, helloworld_client.c
- 실행하기
  - 포트 번호는 123+각자 아이디
    - Ex) ID : 25 = 12325



파일의 조작

저 수준 파일 입출력(Low-Level File Access)

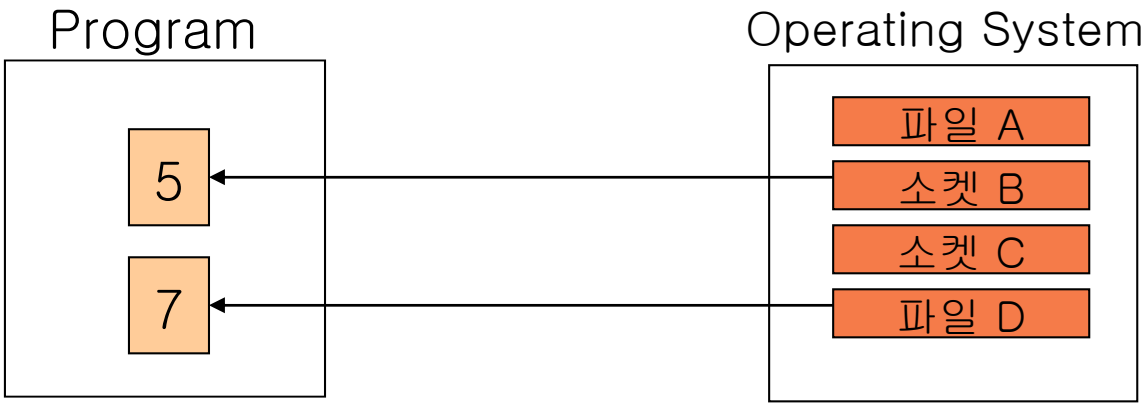
- 리눅스 혹은 윈도우즈 자체에서 제공해 주는 파일 입출력 함수를 사용하여 파일을 관리(파일의 생성 및 삭제, 데이터 입력 및 출력)하는 것을 의미함.

- 리눅스에서는 모든 것을 파일로 관리한다.
  - 파일, 소켓, 표준 입력, 표준 출력.
- 파일에 파일 디스크립터를 할당해서 관리(파일 디스크립터는 정수)

파일 디스크립터(File Descriptor)

- 파일을 관리하기 위해서 모든 파일(파일, 소켓 표준 입력, 표준 출력)에 파일 디스크립터를 할당 해 준다.

파일 디스크립터	대 상
0	표준 입력
1	표준 출력
2	표준 에러 출력



File open 및 close

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

int open(const char *path, int flag);
```

성공 시 파일 디스크립터, 실패 시 -1 리턴

- path : 파일의 경로를 포함한 이름을 나타내는 문자열의 포인터
- flag : 파일 오픈 모드

MODE	의 미
O_CREAT	필요한 경우 파일을 생성
O_TRUNC	존재하던 데이터를 모두 삭제
O_APPEND	존재하던 데이터 보존하고 뒤에 이어서 저장
O_RDONLY	읽기 전용 모드로 파일을 오픈
O_WRONLY	쓰기 전용 모드로 파일을 오픈
O_RDWR	읽기 쓰기 겸용 모드로 파일을 오픈

```
#include <unistd.h>

int close(int fildes);
```

성공 시 0, 실패 시 -1 리턴

- Fildes : 닫아줄 파일의 파일 디스크립터

Data read & write

```
#include <unistd.h>

ssize_t write(int fildes, const void * buf, size_t nbytes);
```

성공 시 전달한 바이트 수, 실패 시 -1 리턴

- fildes : 데이터 전송 영역을 나타내는 파일 디스크립터
- buf : 전송할 데이터를 가지고 있는 버퍼의 포인터
- nbytes : 전송할 데이터의 바이트 수

```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbytes);
```

성공 시 수신한 바이트 수(단, EOF 만나면 0), 실패 시 -1 리턴

- fildes : 데이터를 전송해 주는 대상을 가리키는 파일 디스크립터
- buf : 수신한 데이터를 저장할 버퍼를 가리키는 포인터
- nbytes : 수신할 최대 바이트 수

예제 확인

- 프로그램 예제
  - low_open.c, low_read.c, fd_seri.c
- 실행하기