파일과 소켓

뇌를 자극하는 TCP/IP 소켓 프로그래밍

5장 파일과 소켓

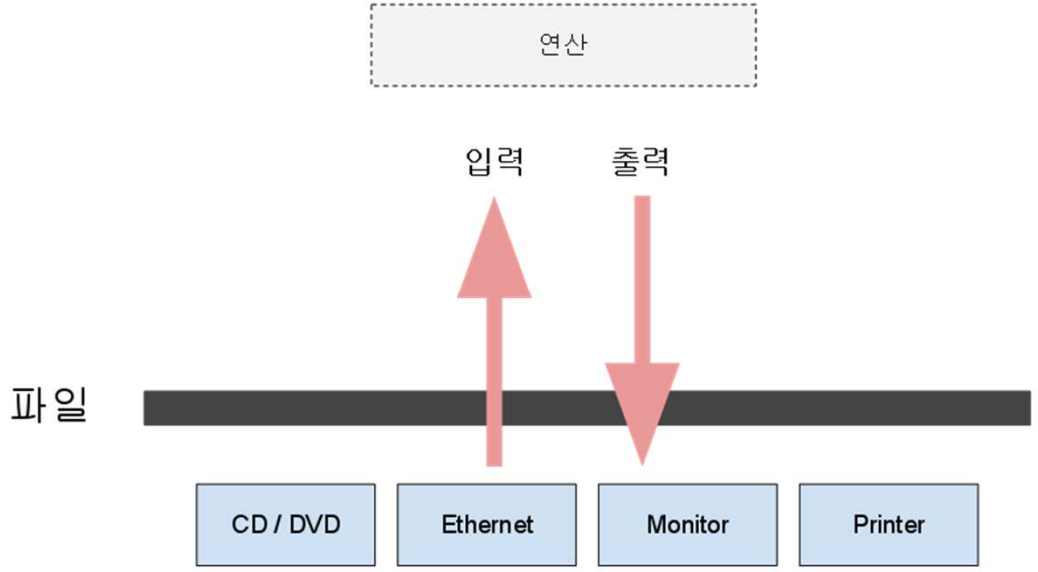
- 리눅스는 모든 자원(**file, directory, devices** such as video card, audio card, printer, and hard disks, and **socket**)을 파일로 본다.
  - 파일로 추상화(abstraction)
  - 개발자는 장치에 상관 없이 읽고/쓰는 동일한 인터페이스로 모든 장치를 제어

- 파일을 가리키는 지정 번호(file descriptor)로 소켓을 비롯한 파일을 제어

- 윈도우는 파일을 커널 객체로 다루지만 BSD Socket과의 호환을 유지하기 위해 소켓 지정 번호로 제어한다.

왜 모든 것을 파일로 다루나

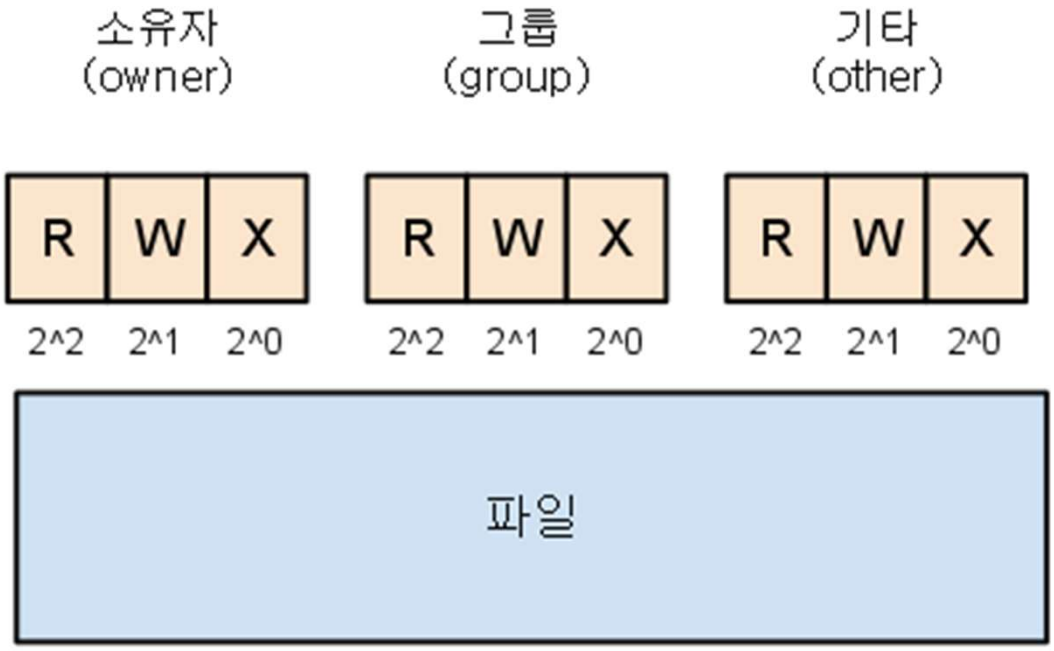
- 컴퓨터에서 모든 정보처리는 “입력”과 “출력” 그리고 “저장”의 무한 반복이다.
- 이는 파일의 기본적인 인터페이스로, 모든 자원을 파일과 마찬가지로 다룰 수 있다.
- 자원을 파일로 추상화 함으로써, 단일 인터페이스로 모든 자원에 접근할 수 있다.



파일 권한

Lab

- 유저 : 소유자, 그룹, 기타
- 권한 : 실행(X), 읽기(쓰기), 읽기(R)

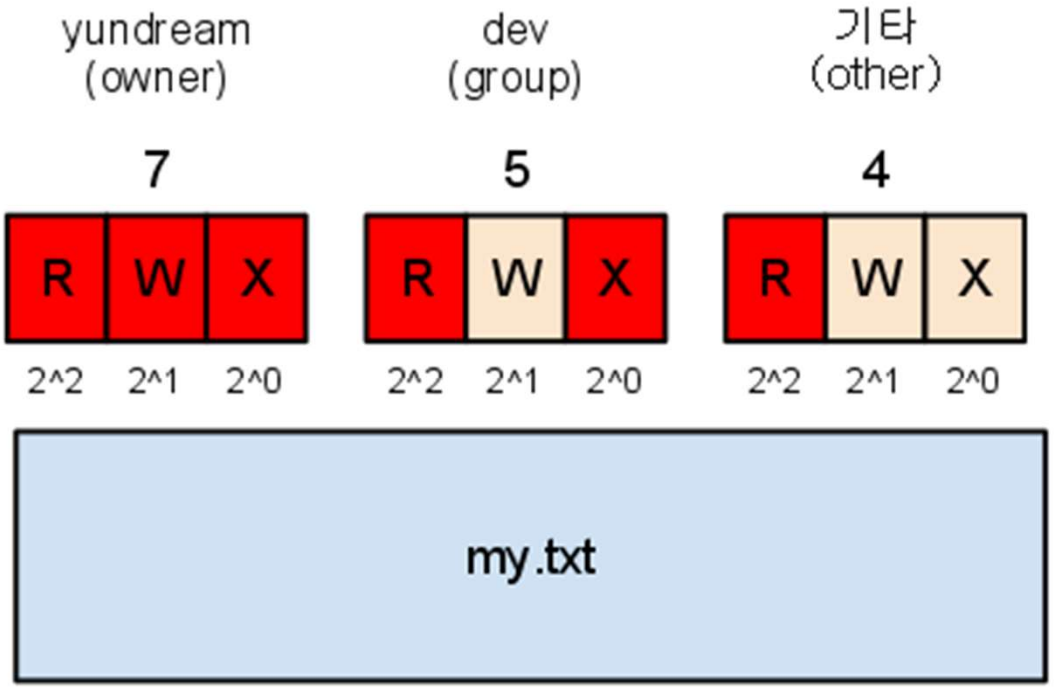


파일 권한

Lab

$chmod 0777 file

- 파일 권한 지정 예
  - owner : 읽기 / 쓰기 / 실행
  - dev 그룹 : 읽기 / 실행
  - other : 읽기



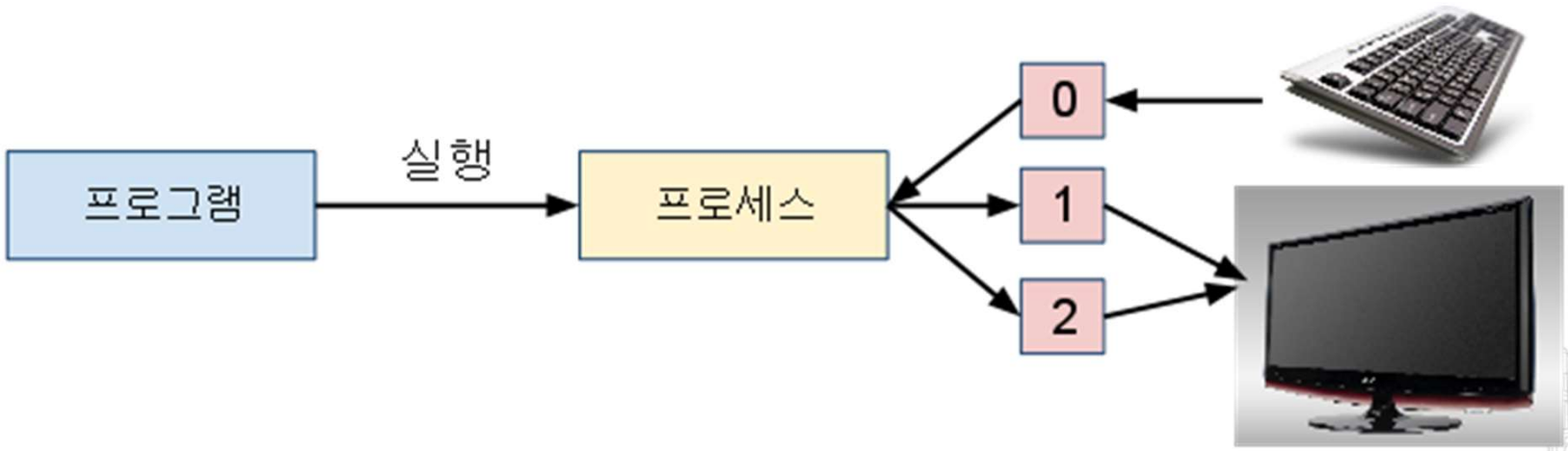
파일의 종류

- 모두 파일로 보지만 파일의 형식을 다르게 함으로써, 입출력 방식을 다르게 한다.
  - 일반 파일
  - 디렉토리
  - 파이프 : 프로세스와 프로세스간의 데이터 통신에 사용
  - 심볼릭 링크 : 파일의 정보를 가지고 있는 파일
  - 블록 장치 : 블록단위로 데이터에 접근하는 장치(HDD, CD-ROM)
  - 문자 장치 : 연속된 데이터의 흐름을 다루는 장치
  - 소켓

표준 입력, 표준 출력, 표준 에러

- 리눅스는 기본적인 입출력을 위한 장치를 가진다.
- 이들 장치를 위한 파일지정번호는 프로세스 생성과 함께 할당된다.
  따라서 open함수 필요 없음

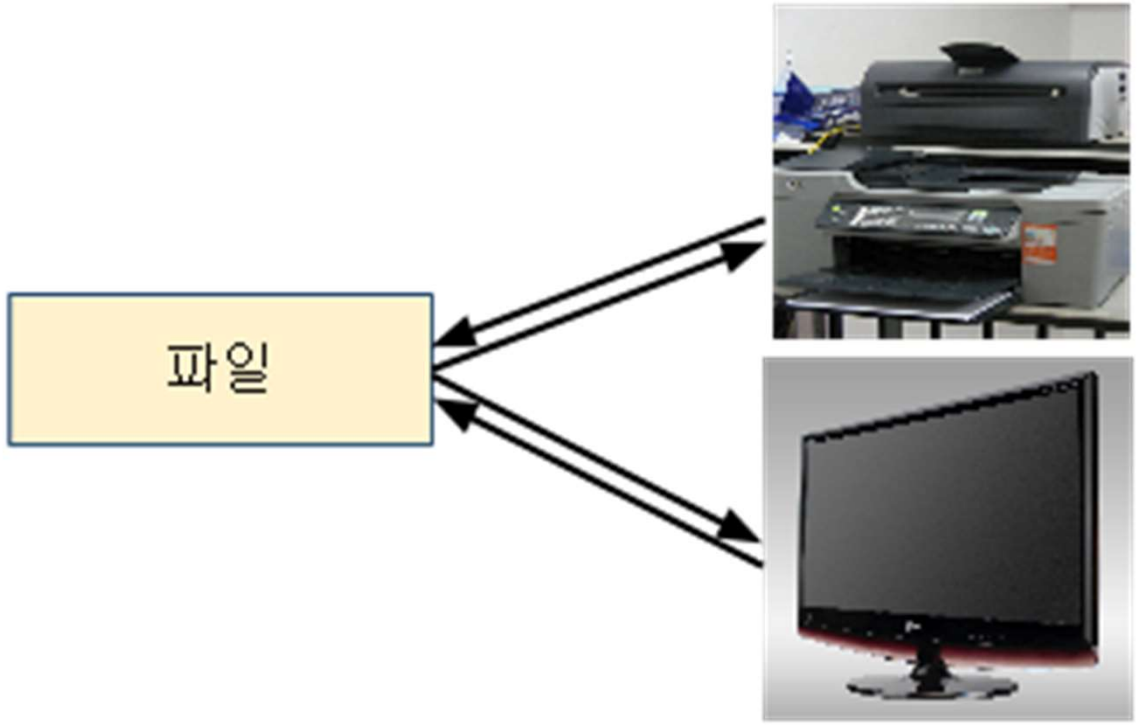
입력 장치	번호	설명
표준입력	0	키보드 입력
표준출력	1	모니터 출력
표준에러	2	모니터 출력



재지향(redirection)

재지향 : 리눅스는 장치로 출력된 데이터를 다른 장치로 보낼 수 있다.

- 파일 데이터를 프린터로 보내거나 혹은 모니터로 전송
- 모니터에 출력된 내용을 파일로 전송



재지향(redirection)

Lab

재지향 사용 예.

- 표준 출력 데이터를 파일로 재지향
  $ cat my.txt > ok.txt

- 표준 에러 데이터를 파일로 재지향
  $ ./stderr 2> errmsg.txt

파일관련 함수

- 저수준 함수 : 운영체제에서 제공하는 시스템 호출**(system call)**
  - open(), read(), write()

- 고수준 함수 : 자주 사용하는 기능들을 묶어서 좀 더 쉽게 사용할 수 있도록 도와주는 일종의 사용자 함수**(user function)**
  - fopen(), fread(), fwrite()
  - fscanf(), fprintf()
  - fgetc(), fgets(), fputc(), fputs(), …

파일 열기

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

 int open(const char *pathname, int flags);
 int open(const char *pathname, int flags, mode_t mode);

fd = open("my.txt", O_WRONLY|O_CREAT, 0755);
```

- pathname : 열고자 하는 파일의 이름
- flag : 파일 제어 방법 지정
- mode : 파일의 권한, 생성방식 지정

파일 열기

- flag에 사용할 수 있는 값들
  - O_RDONLY : 읽기 전용
  - O_WRONLY : 쓰기 전용
  - O_RDWR : 읽기/쓰기
  - O_CREATE : 파일을 생성
  - O_EXCL : 파일이 있는지 검사
  - O_APPEND : 추가 모드로 열기
  - O_NONBLOCK : 비 봉쇄 모드로 열기.

파일 열기

open 함수의 사용 예

- my.text 라는 이름의 파일을
- 쓰기 전용으로 open 한다.
- 만약 파일이 존재하지 않으면 새로 생성하고
- 파일이 존재하면 실패한다.

```
open("my.txt", O_WRONLY|O_CREATE|O_EXCL)
```

파일 열기

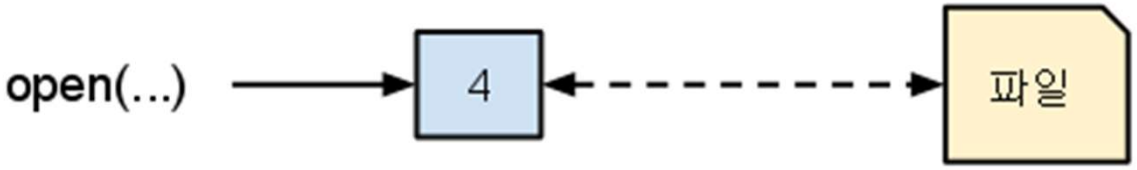
- 소유자/그룹/Other 단위로
- 읽기/쓰기/실행 권한 지정

쓰기 전용으로 파일을 생성하되 755의 권한을 가지게 한다.

- open("my.txt", O_CREATE|O_WRONLY, 0755);

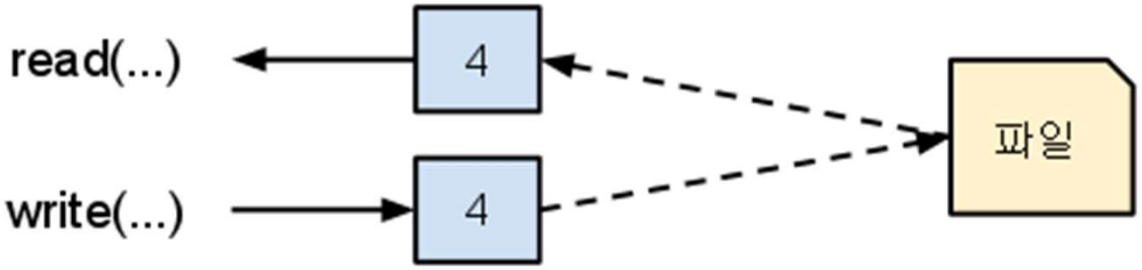
파일 지정 번호

- 파일 지정 번호 : 파일을 가리키는 int 타입의 객체
- 파일의 종류를 막론하고 성공적으로 열리면 파일 지정 번호를 반환 한다.
- 파일에 대한 모든 제어는 파일 지정 번호로 이루어진다.

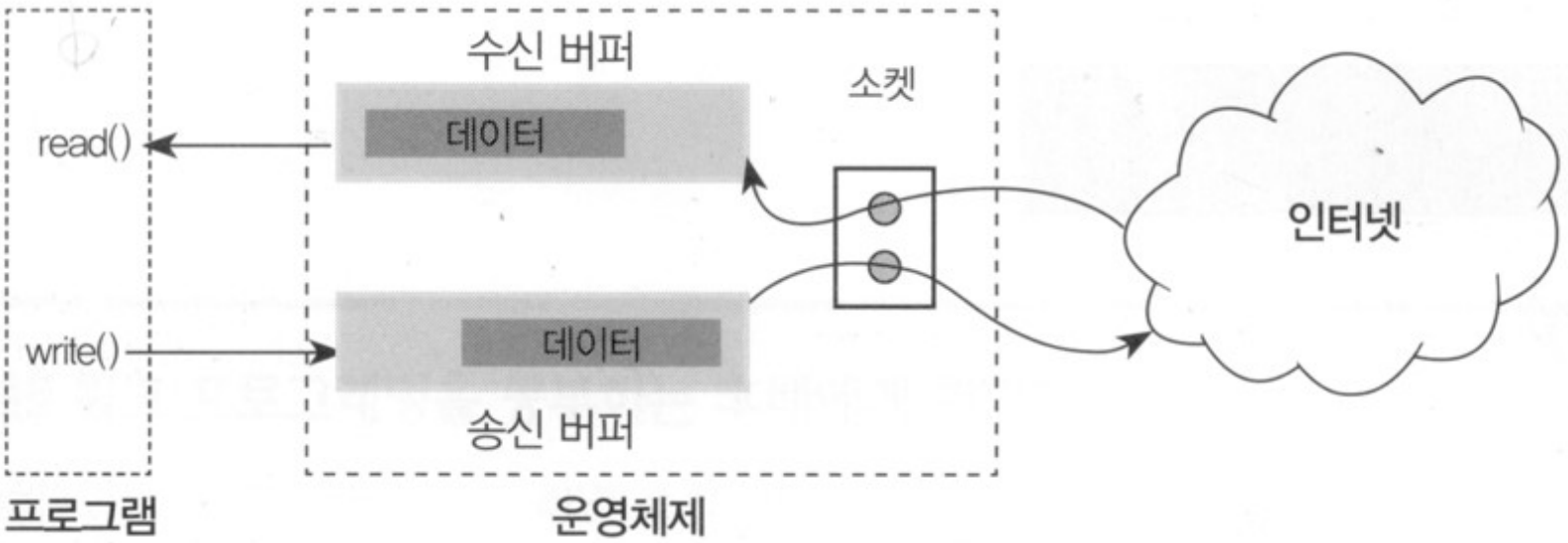


파일 읽기와 쓰기

- 파일 지정 번호를 이용 읽고 쓰기
  - 읽기 : read 함수
  - 쓰기 : write 함수
- 읽기/쓰기 함수 모두 파일 지정 번호를 이용 파일을 제어한다.



송신 버퍼 / 수신 버퍼



- 응용 프로그램내의 user buffer(사용자 버퍼)

```
#define MAXLINE 1024
char buf[MAXLINE]

memset(buf, 0x00,MAXLINE)
read(fd, buf, MAXLINE)
```

- 운영체제내 송수신 버퍼

파일 닫기

- 더 이상 쓰지 않는 파일은 닫아야 한다.
  - close(int fd);
- close 함수를 호출하지 않으면, 파일은 열린 채로 남는다 - 자원낭비
- 프로세스는 열 수 있는 파일 개수에 한계가 있으며, 파일을 닫지 않으면 한계를 초과 해서 더 이상 파일을 열 수 없게 된다.

- 생성 가능한 파일(소켓) 개수 제한
  - 리눅스는 프로세스 하나당 1,024개의 파일(소켓) 생성 가능
  - ulimit 명령어로 조정 가능, 예 ulimit –n 4096

Thank You !

뇌를 자극하는 TCP/IP 소켓 프로그래밍