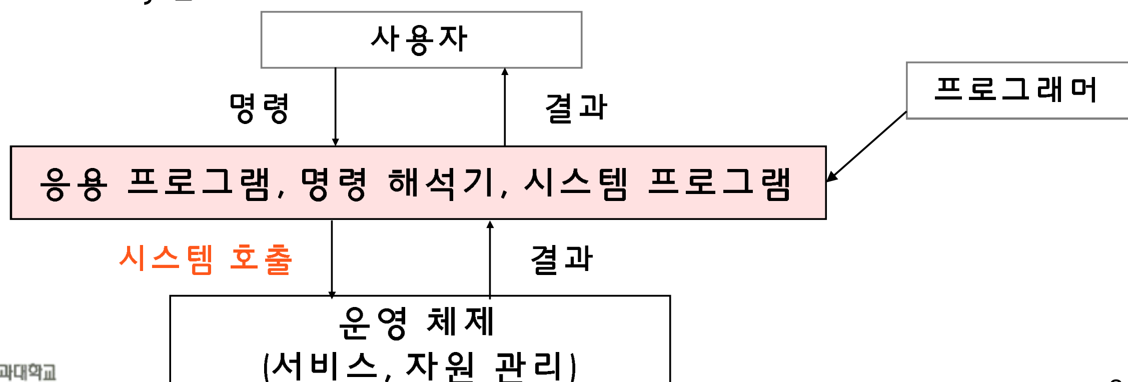
파일 시스템 프로그래밍
File System Programming

- 파일 시스템 내부 구조
- File Descriptor
- File 기본 작업 시스템 호출
- File 정보 관리 시스템 호출
- Directory 관리 시스템 호출

이제부터는…

- 시스템 사용에만 머물지 않는다.
  - 내부로 들어가서 건드려 보고 나만의 시스템을 만들자.
- 시스템 호출(또는 표준 API)을 이용하여 운영 체제 기능을 프로그램에서 사용
- 시스템 호출 (System Calls)
  - 프로세스와 운영체제 간의 인터페이스 제공
    - 운영체제마다 다른 것이 원칙
    - Library 함수와 다르다.



파일 관련 시스템 호출

❑ 파일 기본 동작

함수	의미
open	이미 존재하는 파일을 읽기 또는 쓰기용으로 열거나, 새로운 파일을 생성하여 연다.
creat	새로운 파일을 생성하여 연다.
close	open 또는 creat로 열려진 파일을 닫는다.
read	열려진 파일로부터 데이터를 읽어 들인다.
write	열려진 파일에 데이터를 쓴다.
lseek	파일 안에서 읽기/쓰기 포인터를 지정한 바이트 위치로 이동한다.
unlink/remove	파일을 삭제한다.

File System Programming

파일 관련 시스템 호출

❑ 파일 정보 관리 작업

함수	의미
umask	파일 생성 마스크를 설정한다.
access	파일에 대한 사용자의 접근 권한을 확인한다.
chmod/fchmod	파일에 대한 접근 권한을 변경한다.
chown/fchown	파일의 소유주와 그룹을 변경한다.
link	파일의 새로운 이름을 생성한다. (hard-link)
rename	파일의 이름이나 위치를 변경한다.
symlink	파일의 새로운 이름을 생성한다. (soft-link, symbolic link)
readlink	심볼형 링크의 값(실제 내용)을 읽어온다.
stat/fstat	파일의 상태 정보를 가져온다.

파일 관련 시스템 호출

❑ 디렉터리 작업

함수	의미
mkdir	새로운 디렉터리를 작성한다.
rmdir	디렉터리를 삭제한다.
opendir	디렉터리를 파일처럼 개방한다.
closedir	개방한 디렉터리를 닫는다.
readdir	개방된 디렉터리로부터 디렉터리 항목을 읽어온다.
rewinddir	개방된 디렉터리 스트림을 초기화한다.
chdir	디렉터리 경로를 변경한다.
getcwd	현재 작업 디렉터리를 구한다.

파일 시스템

❑ 파일

- 정보의 논리적 저장 단위
- FCB (File Control Block) - 파일에 대한 정보를 구성하는 저장 구조로서 운영체제에서 사용

❑ 운영체제에서 파일 시스템 제공

- 파일의 물리적 의미, 구조, 속성, 연산 정의
- 논리적 파일 시스템을 물리적 보조 저장 장치에 매핑하는 알고리즘과 자료 구조
- 기본적으로 디스크 기반 파일 시스템 제공
  - UFS, VFS, FAT, NTFS, ext3, JFS, ReiserFS, XFS, …

❑ 계층적 구조 사용

파일 시스템

❑ 디스크 내 기본 정보

- 운영체제 부트 방법, 블록의 수, 자유 블록의 수와 위치, 디렉터리 구조, 개별 파일 정보 등

❑ 디스크 구조

- 부트 제어 블록
  - 시스템 부팅에 필요한 정보
  - UFS – boot block, NTFS – partition boot sector
- 파티션 제어 블록 (또는 Volume Control Block)
  - 파티션의 블록 수, 크기, 자유 블록 수와 포인터, 자유 FCB 수와 포인터 등의 파티션 정보
  - UFS – Superblock, NTFS – Master File Table
- 디렉터리 구조
- 파일 제어 블록 (FCB)
  - 파일 허가, 소유, 크기 등 자세한 파일 정보
  - UFS – inode, NTFS – Master File Table 안에 저장

유닉스 파일 시스템

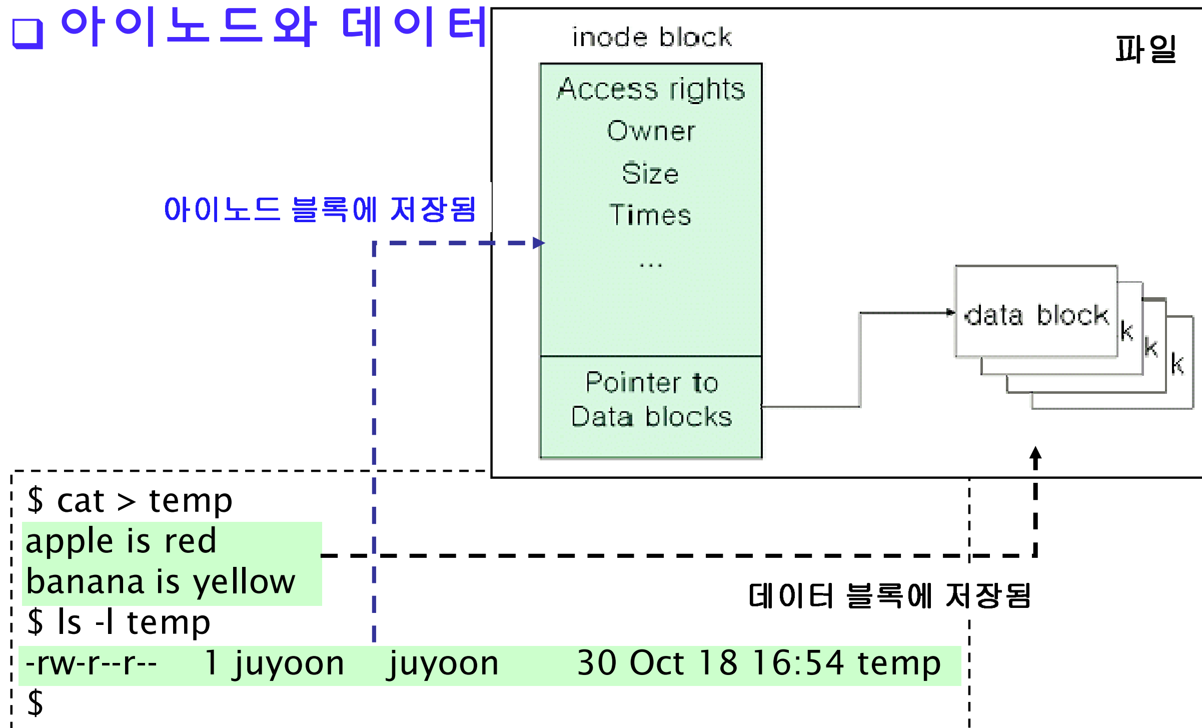
❑ 유닉스 파일 시스템 구조

부트 블록 (Boot Block)	수퍼 블록 (Super Block)	아이노드 블록 (Inode Blocks)	데이터 블록 (Data Blocks)

- 부트 블록 (boot block) – 운영체제 당 하나
  - 운영체제를 부팅시키기 위한 코드가 저장되어 있다.
- 수퍼 블록 (super block) – 파일 시스템(파티션) 당 하나
  - 파일 시스템과 관련된 정보를 저장하고 있다.
- 아이노드 블록 (inode blocks) – 파일 당 하나
  - 파일에 대한 정보를 저장하고 있다.
  - 소유자, 크기, 접근권한, 접근시간 등
- 데이터 블록 (data blocks)
  - 파일이 보관해야 하는 데이터를 저장하고 있다.
  - 보관하는 데이터의 크기에 따라 여러 개일 수 있다.

유닉스 파일 시스템

❑ 아이노드와 데이터



유닉스 파일 시스템

❑ 수퍼 블록 정보 보기

텔넷 ce.kumoh.ac.kr

```
[juyoon:juyoon/6]$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda1              6048320    169864   5571216   3% /
/dev/sdc1             34985444   1335192  31873064   5% /home
/dev/sdd1             34985444     50200  33158056   1% /home1
/dev/sde1             34985444     32840  33175416   1% /home2
/dev/sdf1             34985444     32828  33175428   1% /home3
none                   1549612         0   1549612   0% /dev/shm
/dev/sda5              6048320   1742716   3998364  31% /usr
/dev/sdb1             17488652     32944  16567316   1% /usr/local
/dev/sda2              3012236    144256   2714964   6% /var
[juyoon:juyoon/7]$
```

❑ 디렉터리 정보는 어디?

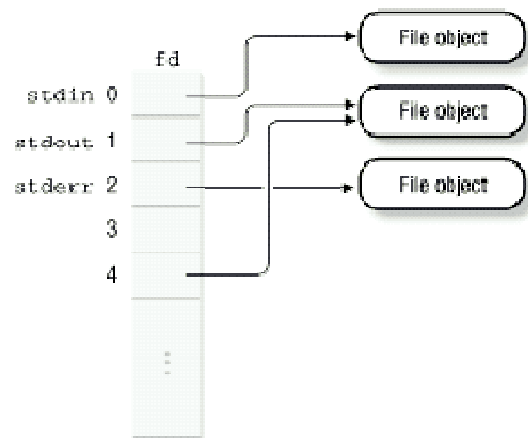
- 디렉터리도 파일이다.
- 아이노드 번호(ID)와 파일명으로 구성된 목록 파일
  - 추가 정보 필요시 아이노드 번호를 이용해 해당 파일의 아이노드 정보를 가져 온다.
- 항상 .(자기 자신)와 ..(부모 디렉터리)을 포함한다.

프로세스와 파일

- 프로그램 전에 프로그램 수행 시 파일 관련 정보는 어떻게 관리되는지 알아 보자.
  - File Descriptor
  - Read/Write Pointer

프로세스와 파일

- 파일 기술자 (File Descriptor)
  - 실행중인 프로그램(프로세스)가 관리하는 파일들의 포인터 배열에 대한 인덱스
    - 음수가 아닌 정수 값
      - 시스템(커널)이 결정
    - 파일 개방이 실패하면 -1
    - 여러 개의 프로그램이 동시에 하나의 파일을 개방할 수 있다.



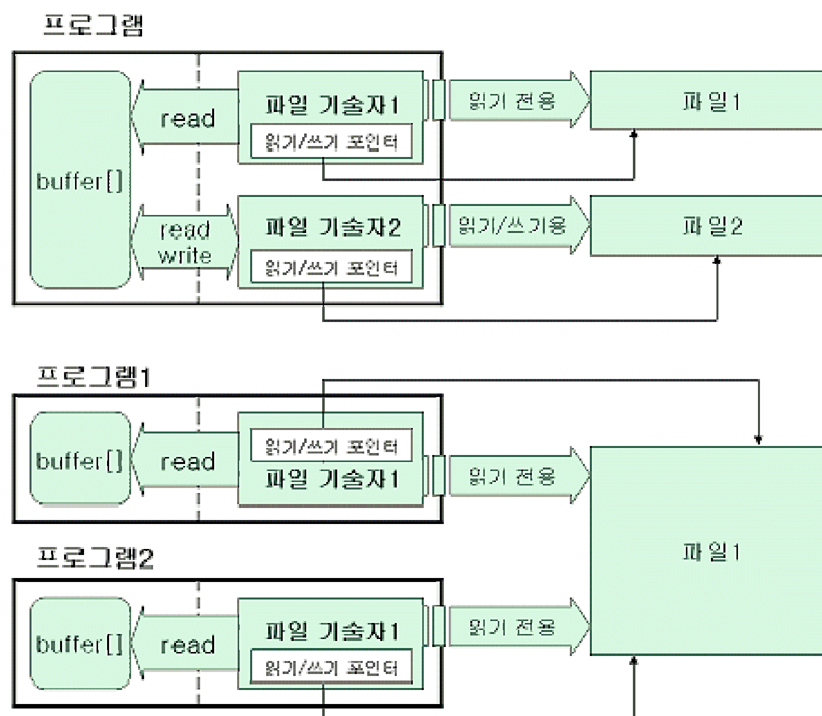
프로세스와 파일

❑ 읽기/쓰기 포인터

- 개방된 파일 내에서 읽기나 쓰기 작업을 수행할 바이트 단위의 위치
- 특정 위치를 기준으로 한 상대적인 위치를 의미
  → 오프셋(offset)
- 파일을 개방한 직후에 읽기/쓰기 포인터는 0
  - 파일의 첫 번째 바이트를 가리킨다.
  - 파일의 내용을 읽거나 파일에 새로운 데이터를 작성하면 그만큼 증가한다.
- 파일 기술자마다 하나씩 존재
  - 한 파일이 여러 프로세스에서 사용되어도 무방

프로세스와 파일

❑ 프로세스와 파일 기술자



시스템 호출

❑ 시스템 호출 사용 시 필수 요소

- 기능
- 이름
- 전달 인자
- 반환 타입
- 자세한 사용법은 ‘man’을 활용하자!

open

❑ 파일을 개방

- 이미 존재하는 파일 개방
- 새로운 파일 개방 - open 또는 creat
- 오류가 발생해도 그대로 진행 - 프로그래머가 체크

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

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

pathname	개방할 파일의 경로 이름을 가지고 있는 문자열의 포인터
flags	파일의 개방 방식 지정: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_APPEND, …
mode	대부분의 경우 생략할 수 있는 값으로 새롭게 생성하는 파일의 초기 접근 권한을 지정
반환값	정상적으로 파일을 개방하게 되면 파일 기술자 반환 파일 개방이 실패할 경우 -1 반환

open

❑ 주요 플래그

O_RDONLY	읽기 전용
O_WRONLY	쓰기 전용
O_RDWR	읽기와 쓰기가 동시에 가능한 상태로 개방
O_CREAT	지정한 경로의 파일이 존재하지 않으면 새롭게 생성한 후 개방한다. 지정한 경로의 파일이 존재하면 지정한 상태로 개방한다.
O_EXCL	지정한 경로의 파일이 존재하지 않으면 새롭게 생성하나, 지정한 경로의 파일이 존재하면 open 호출을 실패한다. (※O_CREAT 플래그와 함께 사용해야 한다.)
O_APPEND	파일을 개방한 직후에 읽기/쓰기 포인터의 위치를 파일 내용의 마지막 바로 뒤로 이동
O_TRUNC	파일을 개방한 직후에 읽기/쓰기 포인터의 위치를 파일 내용의 첫 부분으로 이동

open

❑ open 사용 예

- 여러 플래그를 동시에 사용할 때는 '|' (bitwise OR) 사용

```
int fd1, fd2, fd3, fd4;
char *filename = “data.txt”;

fd1 = open(“data1.txt”, O_RDONLY);
fd2 = open(“data2.txt”, O_WRONLY | O_APPEND);
fd3 = open(“data3.txt”, O_RDWR | O_CREAT);
fd4 = open(filename, O_RDWR | O_CREAT | O_EXCL, 0644);
```

creat

❑ 새로운 파일 생성

- O_WRONLY | O_CREAT | O_TRUNC 설정으로 개방하는 것과 같은 효과
  - 기존 파일이 있으면 데이터를 모두 삭제(O_TRUNC)하고 개방

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

int creat (const char *pathname, mode_t mode);
```

pathname	개방할 파일의 경로 이름을 가지고 있는 문자열의 포인터
mode	새롭게 생성하는 파일의 초기 접근 권한을 지정. open과 달리 생략할 수 없다.
반환값	정상적으로 파일을 개방하게 되면 파일 기술자 반환 파일 개방이 실패할 경우 -1 반환

File System Programming

close

❑ 개방 상태의 파일 닫기

- 사용이 끝나면 반드시 닫아 주어야 한다.
  - 하나의 프로세스가 동시에 개방할 수 있는 파일의 수와 전체 시스템에서 동시에 개방할 수 있는 파일의 수가 제한되어 있음.
  - 프로세스 정상 종료 시 자동 폐쇄 – 프로그램 내에서 처리하는 습관이 좋다!

```
#include <unistd.h>

int close (int filedes);
```

filedes	이전에 open이나 creat에 의해 개방된 파일의 파일 기술자
반환값	작업이 성공할 경우 0이 반환되며, 실패할 경우 -1 반환

read/write

❑ 파일에서 데이터 읽기/쓰기

```
#include <unistd.h>

ssize_t read (int filedes, void *buf, size_t count);
ssize_t write (int filedes, const void *buf, size_t count);
```

filedes	읽기/쓰기 작업을 수행할 파일의 파일 기술자
buf	읽거나 쓸 내용을 저장하는 공간. 일반적으로 배열을 사용하게 되는데 배열의 데이터 형식은 어느 것이라도 상관없다.
count	읽거나 쓸 내용의 크기를 지정. 바이트 단위. 버퍼(buf) 크기와 상관없다.
반환값	파일로부터 읽기/쓰기 작업이 성공할 경우 1) 읽거나 쓴 내용의 바이트 크기 반환 (1 이상의 값) (일반적으로 count 값과 같다.) 2) 읽거나 쓴 내용이 없을 경우 (EOF일 경우) 0을 반환 읽기 작업이 실패한 경우: -1을 반환 쓰기 작업이 실패한 경우: count 값과 반환값이 다르다.

read/write

❑ read/write 사용 예

```
ssize_t nread, nwrite;

if ((nread = read(fd, buf, BUFSIZE)) > 0)   // 읽기가 정상적으로 수행되면
   ……

if ((write(fd, buf, nwrite) < nwrite)        // 쓰기가 정상적으로 수행되면
   ……
```

lseek

❑ 읽기/쓰기 포인터 위치 변경

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

off_t lseek (int filedes, off_t offset, int whence);
```

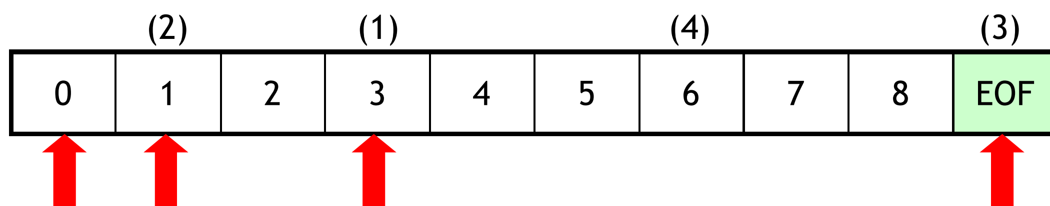
filedes	읽기/쓰기 포인터를 변경할 파일을 지정
offset	새롭게 지정할 읽기/쓰기 포인터의 위치. 오프셋이기 때문에 기준에 따라 음수가 될 수도 있으며, 바이트 단위로 이동한다.
whence	offset의 기준. 파일의 맨 처음(SEEK_SET), 현재 포인터의 위치(SEEK_CUR) 또는 파일의 맨 마지막(SEEK_END)
반환값	작업이 성공하면 파일의 첫 부분을 기준으로 한 포인터의 오프셋을 반환. 작업이 실패할 경우 (off_t)-1 반환.

lseek

❑ lseek 사용 예

```
off_t newpos;

newpos = lseek (fd, (off_t) 3, SEEK_SET);   --- (1)
newpos = lseek (fd, (off_t) -2, SEEK_CUR); --- (2)
newpos = lseek (fd, (off_t) 0, SEEK_END);   --- (3)
newpos = lseek (fd, (off_t) -3, SEEK_CUR); --- (4)
```



unlink/remove

❑ 경로명으로 지정한 파일 삭제

- 디렉터리 삭제
  - 비어 있지 않으면 삭제할 수 없다.
  - 비어 있는 디렉터리는 remove로 삭제

```
#include <unistd.h>
int unlink (const char *pathname);

#include <stdio.h>
int remove (const char *pathname);
```

pathname	삭제할 파일의 경로 이름
반환값	작업이 성공할 경우 0이 반환되며, 실패할 경우 -1이 반환된다.

예제 1

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

int main (void)
{
    int fd, fd1, fd2;
    ssize_t nread;
    off_t newpos;

    char buffer[1024];
    char content[] = "Hello!₩n";

    fd = open("data.txt", O_RDWR);

    nread = read(fd, buffer, 1024);
    printf("%s₩n", buffer);

    write(fd, content, strlen(content));

    newpos = lseek(fd, (off_t)5, SEEK_SET);
    nread = read(fd, buffer, 1024);
    printf("%s₩n", buffer);

    close(fd);

    fd1 = open("data1.txt", O_RDWR | O_CREAT, 0644);
    fd2 = creat("data2.txt", 0644);

    close(fd1);
    close(fd2);
    unlink("data2.txt");

    return 0;
}
```

실행결과:

```
juyoon@ce:~/system/programming/file
[juyoon:file/51]$ cat data.txt
babo computer

[juyoon:file/52]$ ex1
babo computer


computer

Hello!

[juyoon:file/53]$
```

umask

❑ 새로운 파일 생성 시 적용하는 접근권한 중 일부를 제한

❑ 명령어로 존재
- 파일 생성 시 접근권한을 설정할 때 디폴트로 사용할 마스크 지정

```
$ umask 077
```

  - mask XOR 777 (또는 ~mask & 777) → 새 파일의 접근권한
  - 시스템에서 022로 지정되어 있다.
  - 항상 적용하고 싶으면 .bashrc 등의 설정 파일에 기록해서 사용

File System Programming

umask

❑ umask 시스템 호출

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

mode_t umask (mode_t mask);
```

mask	새로 생성하는 파일의 접근권한 설정을 위한 마스크. (open의 mode & ~mask)로 실제 접근권한이 설정됨.
반환값	umask의 실행은 항상 성공하며 이전 mask 값을 반환한다.

- 프로그램에서 설정한 umask는 프로세스 실행 중일 때만 유지됨.
- 예

```
mode_t oldmask;
oldmask = umask (022);
fd = open (“data.txt”, O_CREAT, 0777);
```

  - 실제로는 (777 & ~022 = 055)로 접근권한이 설정됨.

access

- 지정한 파일에 대해 특정 접근권한을 가지고 있는지 검사

```
#include <unistd.h>

int access (const char *pathname, int mode);
```

pathname	파일의 경로이름
mode	검사하려는 접근 권한: R_OK, W_OK, X_OK, F_OK 등
반환값	성공하면 0을 반환, 실패하면 -1을 반환

- 사용 예

```
if (access("data.txt", R_OK) == -1) {
   printf("User cannot read file data.txt\n");
   exit(1);
}
```


chmod/fchmod

- 파일의 접근 권한 변경
  - chmod: 파일 경로명 이용
  - fchmod: 파일 기술자 이용
  - 8진수 또는 매크로 상수로 이루어진 mode를 이용해 접근 권한 변경

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

int chmod (const char *path, mode_t mode);
int fchmod (int filedes, mode_t mode);
```

path	파일의 경로 이름
filedes	개방된 파일의 파일 기술자
mode	파일에 새롭게 적용하려는 접근 권한
반환값	성공하면 0을 반환, 실패하면 -1을 반환

chmod/fchmod

❑ 접근권한을 나타내는 매크로 상수

8진수 값	상수이름	의미
0400	S_IRUSR	소유자에 대한 읽기 권한
0200	S_IWUSR	소유자에 대한 쓰기 권한
0100	S_IXUSR	소유자에 대한 실행 권한
0040	S_IRGRP	그룹 사용자에 대한 읽기 권한
0020	S_IWGRP	그룹 사용자에 대한 쓰기 권한
0010	S_IXGRP	그룹 사용자에 대한 실행 권한
0004	S_IROTH	기타 사용자에 대한 읽기 권한
0002	S_IWOTH	기타 사용자에 대한 쓰기 권한
0001	S_IXOTH	기타 사용자에 대한 실행 권한

- **사용 예:**

```
mode_t mode;
mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
chmod("test.txt", mode);
```


chown/fchown

❑ 지정한 경로의 파일이나 이미 개방된 파일의 소유주 변경

- 시스템 관리자만 수행할 수 있다.

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

int chown (const char *path, uid_t owner, gid_t group);
int fchown (int fd, uid_t owner, gid_t group);
```

path	파일의 경로 이름
fd	개방된 파일의 파일 기술자
owner	새로운 소유주의 사용자 식별 번호(UID)
group	새로운 소유주의 그룹 식별 번호(GID)
반환값	성공하면 0을 반환, 실패하면 -1을 반환

- UID, GID는 'id' 명령으로 알 수 있다.

link/symlink

❑ 파일의 하드/심볼릭 링크 생성

```
#include <unistd.h>

int link (const char *oldpath, const char *newpath);
int symlink (const char *oldpath, const char *newpath);
```

oldpath	원본 파일의 경로 이름
newpath	하드 링크/소프트 링크의 경로 이름
반환값	성공하면 0을 반환, 실패하면 -1을 반환

readlink

❑ 소프트링크 파일의 내용을 읽는다.

- 내용 == 원본 파일의 경로 이름

```
#include <unistd.h>

int readlink (const char *path, char *buf, size_t bufsize);
```

path	소프트 링크의 경로 이름
buf	소프트 링크의 실제 내용을 담을 공간
bufsize	buf의 크기
반환값	읽기가 성공하면 buf에 저장한 바이트 수를 반환하며, 실패할 경우 -1을 반환

- 사용 예

```
char buffer[1024];
int nread;

nread = readlink(“link”, buffer, 1024);
write = (1, buffer, nread);
```

rename

❑ 지정한 경로의 파일 이름을 새로운 이름으로 변경

```
#include <stdio.h>

int rename (const char *oldpath, const char *newpath);
```

oldpath	이름을 바꾸려는 파일의 경로 이름
newpath	파일의 새로운 이름
반환값	성공하면 0을 반환하고, 실패하면 -1을 반환

- oldpath == newpath 이면 성공으로 간주
- newpath와 같은 이름의 파일이 이미 존재할 경우, 이를 삭제한다. 즉, overwrite한다.
- newpath와 같은 이름의 디렉터리가 존재할 경우, 비어 있으면 overwrite하고 비어 있지 않으면 실패한다.

stat/fstat

❑ 파일의 상세 정보를 읽어 온다.

- 아이노드에 저장된 메타정보

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

int stat (const char *filename, struct stat *buf);
int fstat (int filedes, struct stat *buf);
```

filename	파일의 경로 이름
filedes	개방된 파일의 파일 기술자
buf	파일의 정보를 담기 위한 struct stat 타입 구조체의 포인터
반환값	성공하면 0을 반환하고, 실패하면 -1을 반환

- 사용 예

```
struct stat fileinfo;

if (stat(“data.txt”, &fileinfo) == -1) {
   printf(“파일 정보 읽기 실패\n”);
   exit(1);
}
```

stat/fstat

❑ struct stat 구조체

```
struct stat {
    dev_t         st_dev;                    // 장치 식별 번호
    ino_t         st_ino;                    // 아이노드 블록 번호
    mode_t        st_mode;                   // 접근 권한
    nlink_t       st_nlink;                  // 하드 링크 계수
    uid_t         st_uid;                    // 소유자 ID
    gid_t         st_gid;                    // 소유자 그룹 ID
    dev_t         st_rdev;                   // 장치 타입
    off_t         st_size;                   // 바이트 단위 파일 크기
    timestruc_t   st_atim;                   // 마지막 접근 시간
    timestruc_t   st_mtim;                   // 마지막 수정 시간
    timestruc_t   st_ctim;                   // 마지막 상태 변경 시간
    blksize_t     st_blksize;                // 할당된 블록 크기
    blkcnt_t      st_blocks;                 // 할당된 데이터 블록 수
    char          st_fstype[_ST_FSTYPSZ];    // 파일 시스템의 종류
};
```


예제 2

```
int main (void)
{
    char *originalName = "manage.txt";
    char *hardfileName = "manage.txt.hard";
    char *softfileName = "manage.txt.soft";

    int fd, retval;
    mode_t oldmask;
    char buffer[1024];
    int nread;
    struct stat fileinfo;

    oldmask = umask(0377);

    fd = open(originalName, O_CREAT, 0755);
    close(fd);

    if (retval = access(originalName, W_OK) == -1) {
        printf("%s is not writable₩n", originalName);
        chmod(originalName, 0644);
    }

    link(originalName, hardfileName);
    symlink(originalName, softfileName);
    rename (hardfileName, "manage.hard.new");

    nread = readlink(softfileName, buffer, 1024);
    write(1, buffer, nread);

    stat(originalName, &fileinfo);
    printf("₩n%s₩n", originalName);
    printf("File mode    : %o₩n", fileinfo.st_mode);
    printf("File size    : %d bytes₩n", fileinfo.st_size);
    printf("Number of Blocks: %d₩n", fileinfo.st_blocks);
    return 0;
}
```

실행결과:

```
juyoon@ce:~/system/program...
[juyoon:file/73]$ ex2
manage.txt is not writable
manage.txt
manage.txt
File mode        : 100644
File size        : 0 bytes
Number of Blocks: 0
[juyoon:file/74]$ ls
합계 68
   4 b
   4 data.txt
  16 ex1*
   4 ex1.c
  16 ex2*
   4 ex2.c
  12 lseek*
   4 lseek.c
   4 lseek.txt
   0 manage.hard.new
   0 manage.txt
   0 manage.txt.soft@
[juyoon:file/75]$
```

mkdir/rmdir

❑ 새 디렉터리 생성 또는 삭제

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir (const char *pathname, mode_t mode);

#include <unistd.h>
int rmdir (const char *pathname);
```

pathname	디렉터리의 경로 이름
mode	생성하려는 디렉터리의 초기 접근 권한
반환값	성공하면 0을 반환하고, 실패하면 -1을 반환

- rmdir은 비어 있는 디렉터리만 삭제 가능

opendir/closedir

❑ 디렉터리 개방 폐쇄

- 디렉터리도 파일 - 특수 형태
  - open으로 개방할 수 있으나 내용을 읽기 어렵다.
  - 별도의 시스템 호출 사용

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

DIR *opendir (const char *name);
int closedir (DIR *dir);
```

name	개방하려는 디렉터리의 경로 이름
dir	닫으려고 하는 개방된 디렉터리에 대한 포인터
반환값	[opendir] 성공하면 디렉터리 스트림에 대한 DIR형 포인터를 반환, 실패하면 NULL을 반환 [closedir] 호출이 성공하면 0을 반환, 실패하면 -1을 반환

readdir

- 개방된 디렉터리(파일)에서 하나의 디렉터리 항(directory entry)을 읽어 온다.

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

struct dirent *readdir(DIR *dirp);
```

dirp	opendir로 개방한 디렉터리에 대한 포인터
반환값	호출이 성공하면 struct dirent 포인터형의 디렉터리 항을 반환하고, 호출이 실패하면 NULL을 반환한다. 더 이상 읽을 디렉터리 항이 없을 경우에도 NULL을 반환한다.

- 사용 예

```
DIR *dirp;
struct dirent *dentry;

if ((dirp = opendir(".")) == NULL)
   exit (1);
while (dentry = readdir(dirp)) {      // 항이 남아 있는 동안
        if (dentry->d_ino != 0)          // 0은 삭제된 파일
           printf("%s\n", dentry->d_name);
}
```


readdir

- 디렉터리 항(directory entry) 구조
  - dirent 자료 구조

```
struct dirent {
    long d_ino;
    char d_name[NAME_MAX + 1];
}
```

  - 디렉터리 파일의 내용

1020	.	\0				
907	.	.	\0			
1507	t	e	s	t	1	\0
0	t	e	m	p	\0	
1347	a	p	p	l	e	\0

↑ 아이노드 블록 번호 — 파일 이름

rewinddir

❑ 디렉터리 파일에서 읽기 포인터의 위치 초기화

- lseek(fd, 0, SEEK_SET)과 같은 효과
- 디렉터리 파일의 읽기 포인터는 항 단위로 이동

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

void rewinddir (DIR *dir);
```

dir	읽기 포인터를 초기화하려는 개방된 파일의 포인터
반환값	없음

chdir

❑ 작업 디렉터리 변경

```
#include <unistd.h>

int chdir (const char *path);
```

path	변경하려는 새로운 디렉터리 경로
반환값	성공하면 0을 반환, 실패하면 -1을 반환한

- 프로그램 내 디렉터리 변경은 프로세스 실행 동안에만 적용되며, 프로세스 종료 후 쉘에 영향을 주지 않는다.

getcwd

❑ 현재 작업 디렉터리를 알아 본다.

- Current Working Directory
- 절대 경로가 버퍼에 저장된다.

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

buf	현재 작업 디렉터리의 경로를 저장할 버퍼
size	버퍼의 최대 크기
반환값	성공하면 buf의 포인터를 반환하고, 실패할 경우 NULL을 반환

- 사용 예

```
char buffer[256];

if (getcwd(buffer, 256) == NULL)
   exit(1);
printf("%s\n", buffer);
```


예제 3

```
int main(void)
{
    char buffer[256];
    DIR *dirp;
    struct dirent *dentry;

    getcwd(buffer, 256);
    printf("I am in %s.₩n", buffer);

    mkdir ("apple", 0755);
    mkdir ("banana", 0755);
    chdir ("apple");
    getcwd(buffer, 256);
    printf("I am in %s now.₩n₩n", buffer);

    close(open("data1.txt", O_CREAT|O_RDWR, 0644));
    close(open("data2txt", O_CREAT|O_RDWR, 0644));

    chdir("..");
    rmdir("apple");
    rmdir("banana");

    dirp = opendir("apple");
    while (dentry = readdir(dirp))
        if (dentry->d_ino != 0)
            printf("%s₩n", dentry->d_name);
    rewinddir(dirp);
    if (dentry = readdir(dirp))
        printf("₩n%s₩n", dentry->d_name);
    closedir(dirp);
}
```

실행결과:

```
juyoon@ce:~/system/program...
[juyoon:file/135]$ ex3
I am in /home/juyoon/system/p
rogramming/file.
I am in /home/juyoon/system/p
rogramming/file/apple now.

.
..
data1.txt
data2txt

.
[juyoon:file/136]$ ls apple
합계 0
   0 data1.txt
   0 data2txt
[juyoon:file/137]$
```