

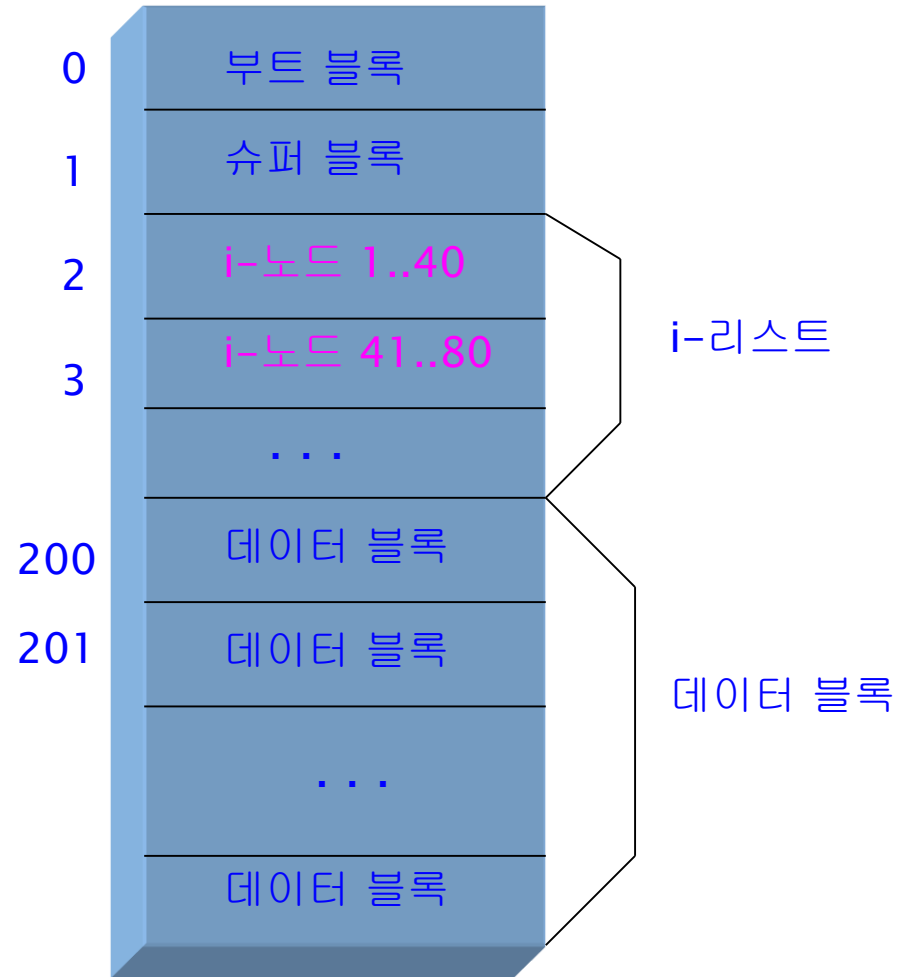
# 시스템 프로그래밍

**Linux system programming**

**File System**



## 파일 시스템 구조



## 파일 시스템 구조

- 부트 블록(Boot block)
  - 파일 시스템 시작부에 위치하고 보통 첫 번째 섹터를 차지
  - 부트스트랩 코드가 저장되는 블록
- 슈퍼 블록(Super block)
  - 전체 파일 시스템에 대한 정보를 저장
    - 총 블록 수, 사용 가능한 i-노드 개수, 사용 가능한 블록 비트 맵, 블록의 크기, 사용 중인 블록 수, 사용 가능한 블록 수 등
- i-리스트(i-list)
  - 각 파일을 나타내는 모든 i-노드들의 리스트
  - 한 블록은 약 40개 정도의 i-노드를 포함
- 데이터 블록(Data block)
  - 파일의 내용(데이터)을 저장하기 위한 블록들

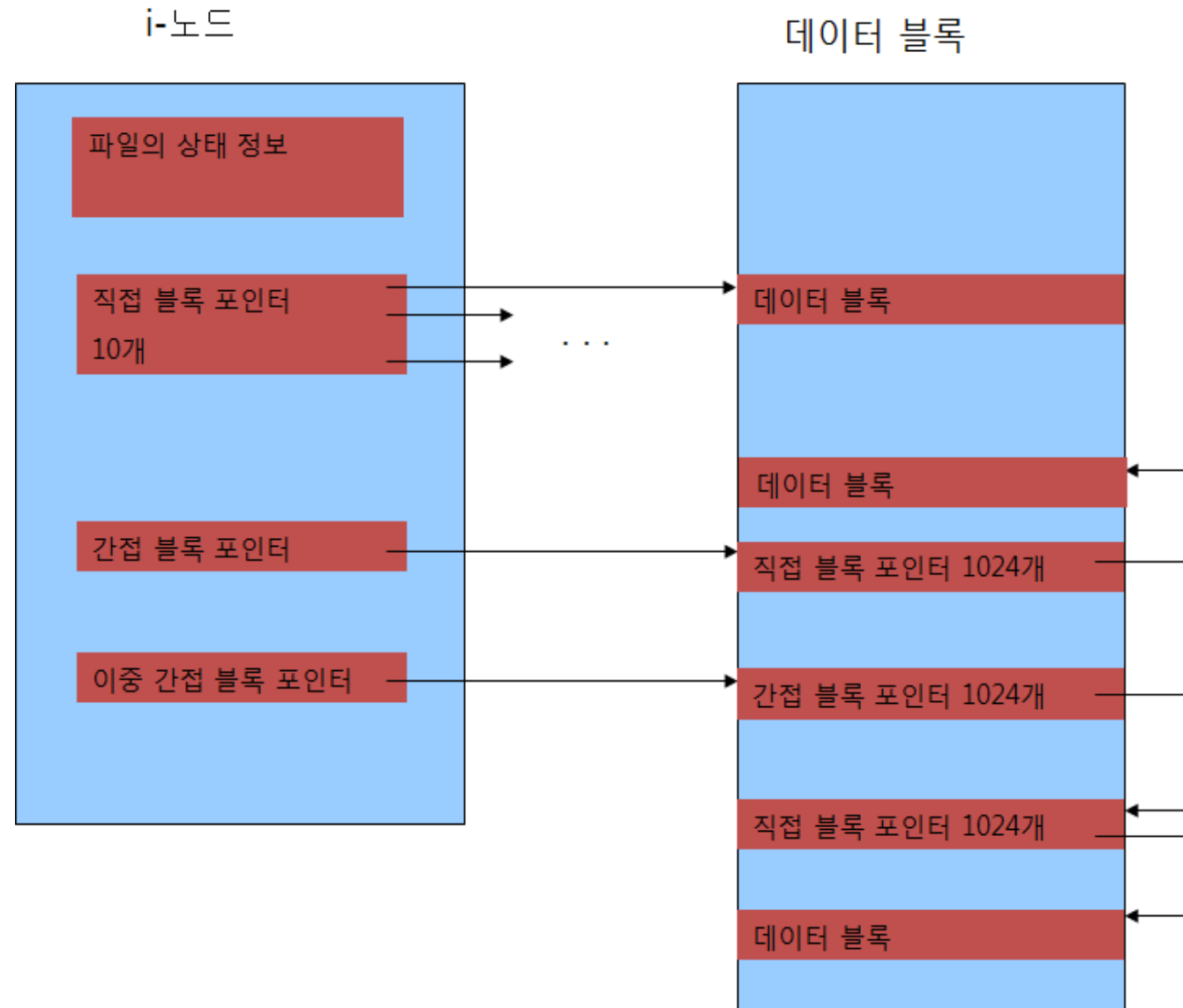


## i-노드(i-Node)

- 한 파일은 하나의 i-노드를 갖는다.
- 파일에 대한 모든 정보를 가지고 있음
  - 파일 타입: 일반 파일, 디렉터리, 블록 장치, 문자 장치 등
  - 파일 크기
  - 사용권한
  - 파일 소유자 및 그룹
  - 접근 및 갱신 시간
  - 데이터 블록에 대한 포인터(주소) 등



# 블록 포인터



## 블록 포인터

- 데이터 블록에 대한 포인터
  - 파일의 내용을 저장하기 위해 할당된 데이터 블록의 주소
- 하나의 i-노드 내의 블록 포인터
  - 직접 블록 포인터 10개
  - 간접 블록 포인터 1개
  - 이중 간접 블록 포인터 1개
- 최대 몇 개의 데이터 블록을 가리킬 수 있을까?



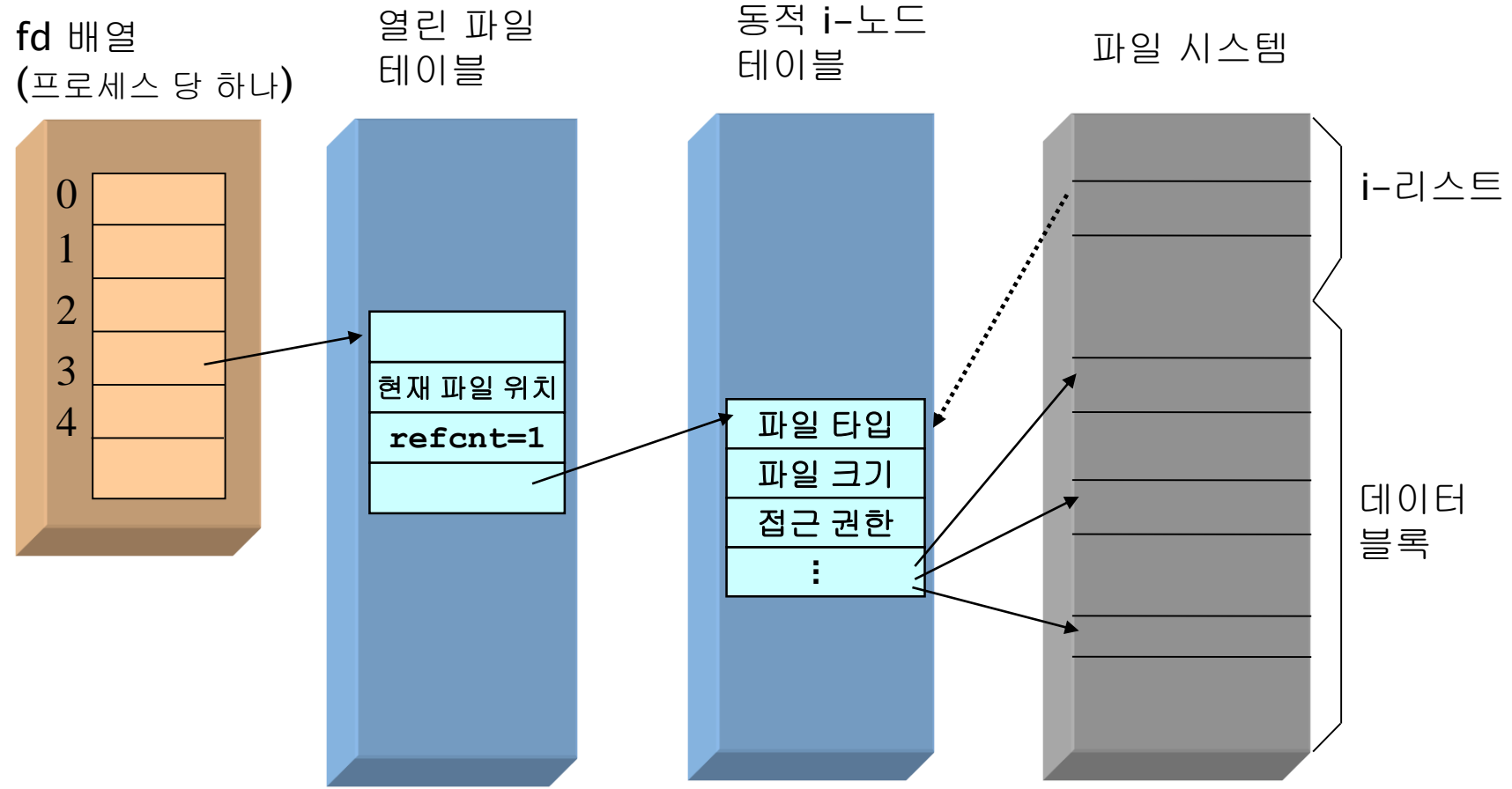
## 파일 입출력 구현

- 파일 입출력 구현을 위한 커널 내 자료구조
  - 파일 디스크립터 배열(Fd array)
  - 열린 파일 테이블(Open File Table)
  - 동적 i-노드 테이블(Active i-node table)



# 파일을 위한 커널 자료구조

- `fd = open("file", O_RDONLY);`





## 파일 디스크립터 배열(Fd Array)

- 프로세스 당 하나씩 갖는다.
- 파일 디스크립터 배열
  - 열린 파일 테이블의 엔트리를 가리킨다.
- 파일 디스크립터
  - 파일 디스크립터 배열의 인덱스
  - 열린 파일을 나타내는 번호



## 열린 파일 테이블(Open File Table)

- 파일 테이블 (file table)
  - 커널 자료구조
  - **열려진 모든 파일 목록**
  - 열려진 파일 → 파일 테이블의 항목
- 파일 테이블 항목 (file table entry)
  - **파일 상태 플래그**  
(read, write, append, sync, nonblocking,...)
  - **파일의 현재 위치** (current file offset)
  - i-node에 대한 포인터



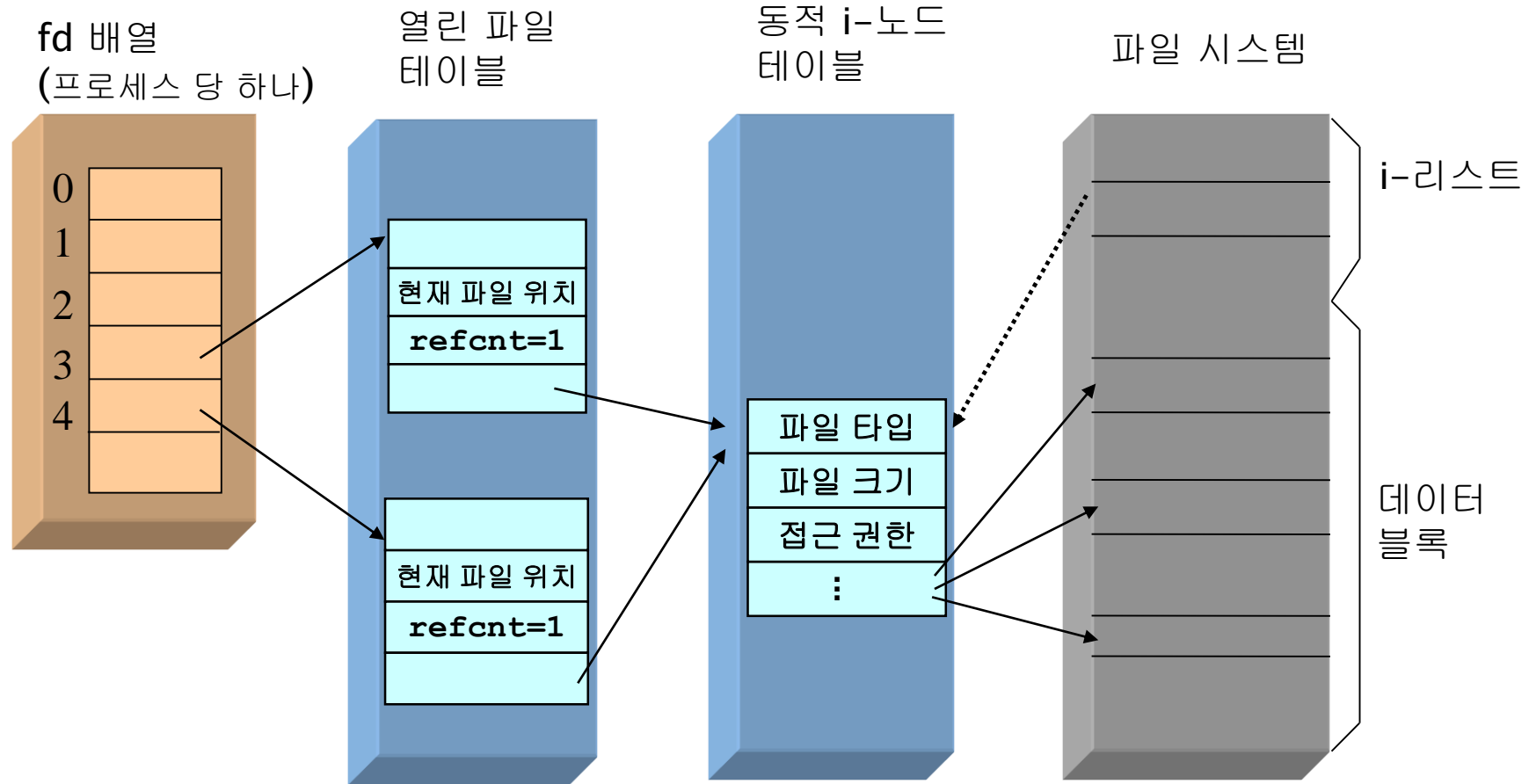
## 동적 i-노드 테이블(Active i-node Table)

- 동적 i-노드 테이블
  - 커널 내의 자료 구조
  - Open 된 파일들의 i-node를 저장하는 테이블
- i-노드
  - 하드 디스크에 저장되어 있는 파일에 대한 자료구조
  - 한 파일에 하나의 i-node
  - 하나의 파일에 대한 정보 저장
    - 소유자, 크기
    - 파일이 위치한 장치
    - 파일 내용 디스크 블록에 대한 포인터
- i-node table vs. i-node



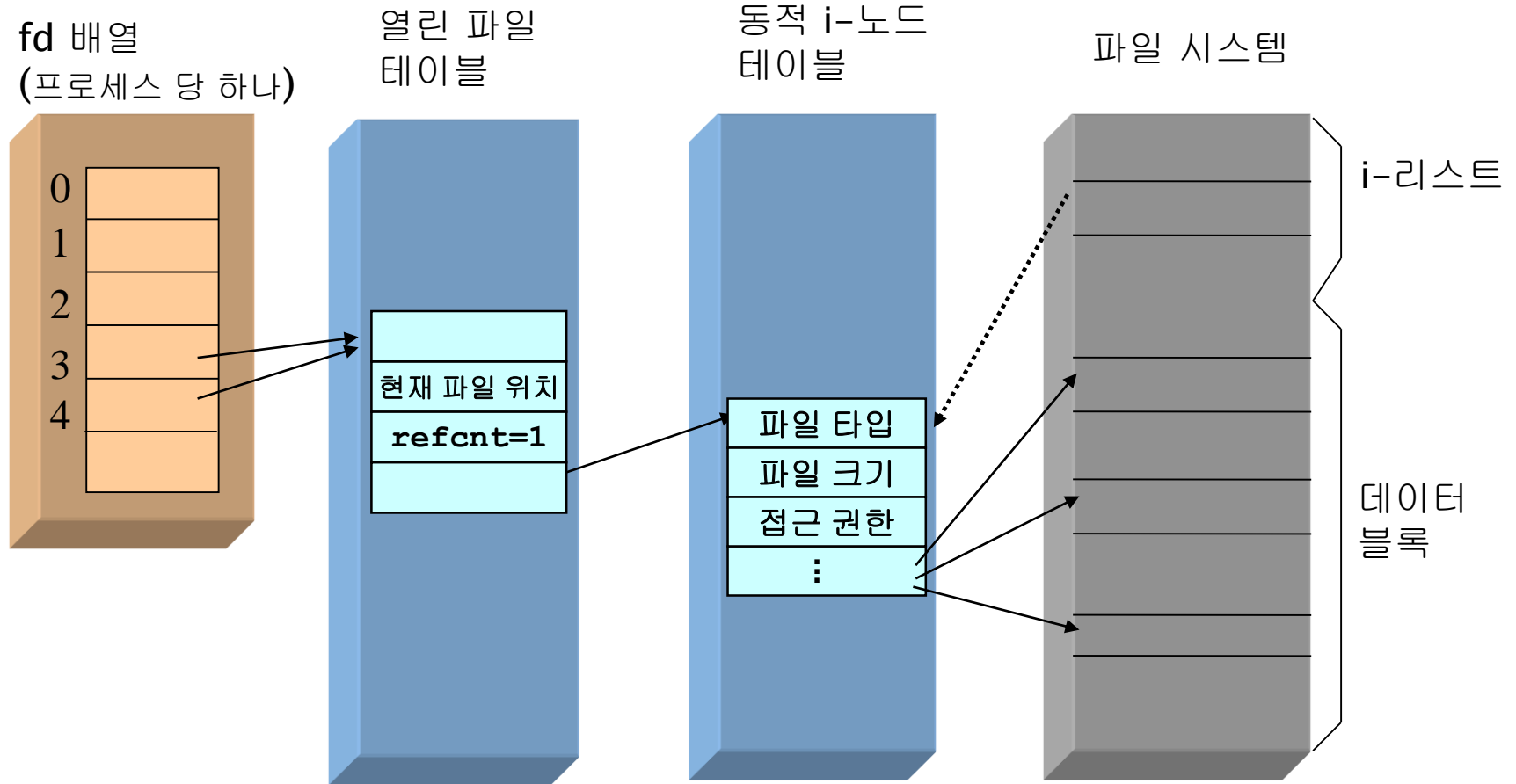
# 파일을 위한 커널 자료구조

- `fd = open("file", O_RDONLY);` //두 번 open



# 파일을 위한 커널 자료구조

- `fd = dup(3);` 혹은 `fd = dup2(3,4);`



## 파일 상태(file status)

- 파일 상태

- 파일에 대한 모든 정보
- 블록수, 파일 타입, 사용 권한, 링크수, 파일 소유자의 사용자 ID,
- 그룹 ID, 파일 크기, 최종 수정 시간 등

- 예

```
$ ls -l hello.c
```

```
2    -rw-r--r-- 1    chang  cs  617    11월 17일 15:53 hello.c
```

블록수    사용권한   링크수   사용자ID   그룹ID   파일 크기   최종 수정 시간    파일이름

↑  
파일 타입



## 상태 정보 : stat()

- 파일 하나당 하나의 i-노드가 있으며
- i-노드 내에 파일에 대한 모든 상태 정보가 저장되어 있다.

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

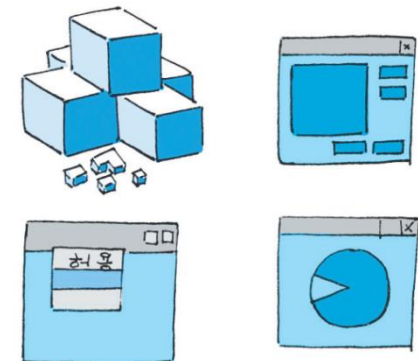
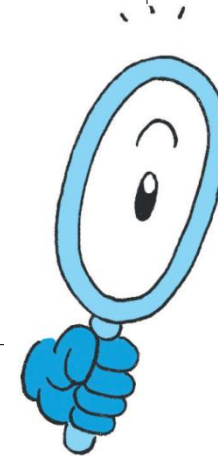
```
#include <sys/stat.h>
```

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

```
int fstat (int fd, struct stat *buf);
```

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

파일의 상태 정보를 가져와서 stat 구조체 buf에  
저장한다. 성공하면 0, 실패하면 -1을 리턴한다.



## stat 구조체

```
struct stat {  
    mode_t st_mode;        // 파일 타입과 사용권한  
    ino_t st_ino;          // i-노드 번호  
    dev_t st_dev;          // 장치 번호  
    dev_t st_rdev;         // 특수 파일 장치 번호  
    nlink_t st_nlink;      // 링크 수  
    uid_t st_uid;          // 소유자의 사용자 ID  
    gid_t st_gid;          // 소유자의 그룹 ID  
    off_t st_size;         // 파일 크기  
    time_t st_atime;       // 최종 접근 시간  
    time_t st_mtime;       // 최종 수정 시간  
    time_t st_ctime;       // 최종 상태 변경 시간  
    long st_blksize;       // 최적 블록 크기  
    long st_blocks;        // 파일의 블록 수  
};
```





# 파일 타입

| 파일 타입    | 설명                              |
|----------|---------------------------------|
| 일반 파일    | 데이터를 갖고 있는 텍스트 파일 또는 이진 화일      |
| 디렉터리 파일  | 파일의 이름들과 파일 정보에 대한 포인터를 포함하는 파일 |
| 문자 장치 파일 | 문자 단위로 데이터를 전송하는 장치를 나타내는 파일    |
| 블록 장치 파일 | 블록 단위로 데이터를 전송하는 장치를 나타내는 파일    |
| FIFO 파일  | 프로세스 간 통신에 사용되는 파일로 이름 있는 파이프   |
| 소켓       | 네트워크를 통한 프로세스 간 통신에 사용되는 파일     |
| 심볼릭 링크   | 다른 파일을 가리키는 포인터 역할을 하는 파일       |



## 파일 타입 검사 함수

- 파일 타입을 검사하기 위한 매크로 함수

| 파일 타입    | 파일 타입을 검사하기 위한 매크로 함수 |
|----------|-----------------------|
| 일반 파일    | S_ISREG()             |
| 디렉터리 파일  | S_ISDIR()             |
| 문자 장치 파일 | S_ISCHR()             |
| 블록 장치 파일 | S_ISBLK()             |
| FIFO 파일  | S_ISFIFO()            |
| 소켓       | S_ISSOCK()            |
| 심볼릭 링크   | S_ISLNK()             |



## ftype.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/* 파일 타입을 검사한다. */
int main(int argc, char *argv[])
{
    int i;
    struct stat buf;
    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            perror("lstat()");
            continue;
        }
    }
```

```
        if (S_ISREG(buf.st_mode))
            printf("%s \n", "일반 파일");
        if (S_ISDIR(buf.st_mode))
            printf("%s \n", "디렉터리");
        if (S_ISCHR(buf.st_mode))
            printf("%s \n", "문자 장치 파일");
        if (S_ISBLK(buf.st_mode))
            printf("%s \n", "블록 장치 파일");
        if (S_ISFIFO(buf.st_mode))
            printf("%s \n", "FIFO 파일");
        if (S_ISLNK(buf.st_mode))
            printf("%s \n", "심볼릭 링크");
        if (S_ISSOCK(buf.st_mode))
            printf("%s \n", "소켓");
    }
    exit(0);
}
```



## 파일 사용 권한(File permissions)

- 각 파일에 대한 권한 관리
  - 각 파일마다 사용권한이 있다.
  - 소유자(owner)/그룹(group)/기타(others)로 구분해서 관리한다.
- 파일에 대한 권한
  - 읽기 r
  - 쓰기 w
  - 실행 x



## 사용권한

- read 권한이 있어야
  - O\_RDONLY O\_RDWR 을 사용하여 파일을 열 수 있다
- write 권한이 있어야
  - O\_WRONLY O\_RDWR O\_TRUNC 을 사용하여 파일을 열 수 있다
- 디렉토리에 write 권한과 execute 권한이 있어야
  - 그 디렉토리에 파일을 생성할 수 있고
  - 그 디렉토리의 파일을 삭제할 수 있다
  - 삭제할 때 그 파일에 대한 read write 권한은 없어도 됨



# 파일 사용 권한

- 파일 사용권한(file access permission)
- stat 구조체의 st\_mode 의 값

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

st\_mode

사용권한



| st_mode mask | Meaning       |
|--------------|---------------|
| S_IRUSR      | user-read     |
| S_IWUSR      | user-write    |
| S_IXUSR      | user-execute  |
| S_IRGRP      | group-read    |
| S_IWGRP      | group-write   |
| S_IXGRP      | group-execute |
| S_IROTH      | other-read    |
| S_IWOTH      | other-write   |
| S_IXOTH      | other-execute |



## chmod(), fchmod()

```
#include <sys/stat.h>
#include <sys/types.h>
int  chmod (const char *path, mode_t mode );
int  fchmod (int fd, mode_t mode );
```

- 파일의 사용 권한(access permission)을 변경한다
- 리턴 값
  - 성공하면 0, 실패하면 -1
- *mode* : bitwise OR
  - S\_ISUID, S\_ISGID
  - S\_IRUSR, S\_IWUSR, S\_IXUSR
  - S\_IRGRP, S\_IWGRP, S\_IXGRP
  - S\_IROTH, S\_IWOTH, S\_IXOTH



## fchmod.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
/* 파일 사용권한을 변경한다. */
main(int argc, char *argv[])
{
    long strtol( );
    int newmode;
    newmode = (int) strtol(argv[1], (char **) NULL, 8);
    if (chmod(argv[2], newmode) == -1) {
        perror(argv[2]);
        exit(1);
    }
    exit(0);
}
```





## chown()

```
#include <sys/types.h>
#include <unistd.h>
int  chown (const char *path, uid_t owner, gid_t group );
int  fchown (int filedes, uid_t owner, gid_t group );
int  lchown (const char *path, uid_t owner, gid_t group );
```

- 파일의 user ID와 group ID를 변경한다.
- 리턴
  - 성공하면 0, 실패하면 -1
- lchown()은 심볼릭 링크 자체를 변경한다
- super-user만 변환 가능



## utime()

```
#include <sys/types.h>
#include <utime.h>
int utime (const char *filename, const struct utimbuf *times );
```

- 파일의 최종 접근 시간과 최종 변경 시간을 조정한다.
- *times*가 NULL 이면, 현재시간으로 설정된다.
- 리턴 값
  - 성공하면 0, 실패하면 -1
- UNIX 명령어 touch 참고



## utime()

```
struct utimbuf {  
    time_t  actime;    /* access time */  
    time_t  modtime;   /* modification time */  
}
```

- 각 필드는 1970-1-1 00:00 부터 현재까지의 경과 시간을 초로 환산한 값



## cptime.c

```
#include <sys/types.h>      #include <sys/stat.h>
#include <sys/time.h>        #include <utime.h>
#include <stdio.h>           #include <stdlib.h>

int main(int argc, char *argv[])
{
    struct stat buf;         // 파일 상태 저장을 위한 변수
    struct utimbuf time;

    if (argc < 3) {
        fprintf(stderr, "사용법: cptime file1 file2\n");
        exit(1);
    }

    if (stat(argv[1], &buf) < 0) { // 상태 가져오기
        perror("stat()");
        exit(-1);
    }

    time.actime = buf.st_atime;
    time.modtime = buf.st_mtime;

    if (utime(argv[2], &time)) // 접근, 수정 시간 복사
        perror("utime");
    else exit(0);
}
```

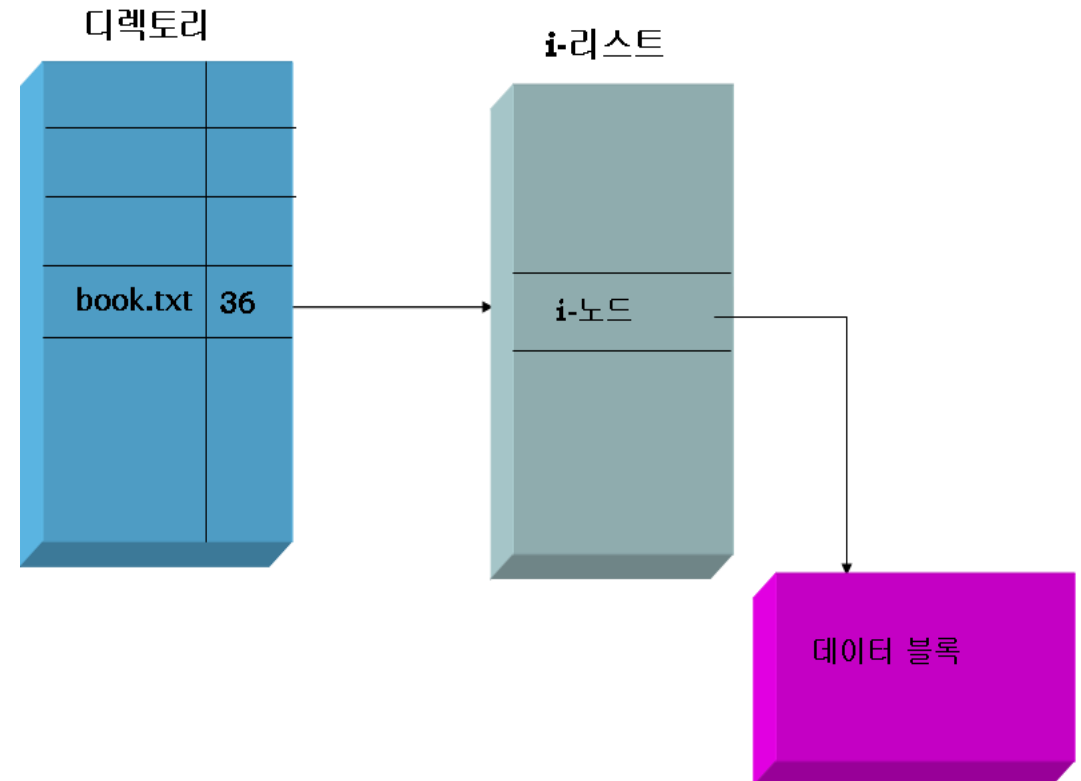


## 디렉토리 구현

- 디렉터리 내에는 무엇이 저장되어 있을까?
- 디렉터리 엔트리

■

```
#include <dirent.h>
struct dirent {
    ino_t d_ino; // i-노드 번호
    char d_name[NAME_MAX + 1];
    // 파일 이름
}
```



## 디렉토리 리스트

- opendir()
  - 디렉터리 열기 함수
  - DIR 포인터(열린 디렉터리를 가리키는 포인터) 리턴
- readdir()
  - 디렉터리 읽기 함수

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

```
#include <dirent.h>
```

```
DIR *opendir (const char *path);
```

path 디렉터리를 열고 성공하면 DIR 구조체 포인터를, 실패하면 NULL을 리턴

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

한 번에 디렉터리 엔트리를 하나씩 읽어서 리턴한다.



## list1.c

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <dirent.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 /* 디렉터리 내의 파일 이름들을 리스트한다. */
8 int main(int argc, char **argv)
9 {
10     DIR *dp;
11     char *dir;
12     struct dirent *d;
13     struct stat st;
14     char path[BUFSIZ+1];
15
```

```
16     if (argc == 1)
17         dir = "."; // 현재 디렉터리를 대상으로
18     else dir = argv[1];
19
20     if ((dp = opendir(dir)) == NULL) // 디렉터리 열기
21         perror(dir);
22
23     while ((d = readdir(dp)) != NULL) // 각 디렉터리 엔트리에 대해
24         printf("%s\n", d->d_name); // 파일 이름 프린트
25
26     closedir(dp);
27     exit(0);
28 }
```



## 파일 이름/크기 출력

- 디렉터리 내에 있는 파일 이름과 그 파일의 크기(블록의 수)를 출력하도록 확장

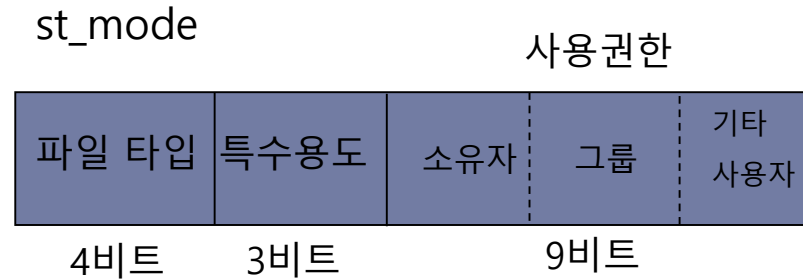
```
while ((d = readdir(dp)) != NULL) {           //디렉터리 내의 각 파일
    sprintf(path, "%s/%s", dir, d->d_name);    // 파일경로명 만들기
    if (lstat(path, &st) < 0)                 // 파일 상태 정보 가져오기
        perror(path);
    printf("%5d %s", st->st_blocks, d->name);    // 블록 수, 파일 이름 출력
    putchar('\n');
}
```





## st\_mode

- lstat() 시스템 호출
  - 파일 타입과 사용권한 정보는 st->st\_mode 필드에 함께 저장됨.
- st\_mode 필드
  - 4비트: 파일 타입
  - 3비트: 특수용도
  - 9비트: 사용권한
    - 3비트: 파일 소유자의 사용권한
    - 3비트: 그룹의 사용권한
    - 3비트: 기타 사용자의 사용권한



## 디렉토리 리스트 : 예

- list2.c
  - ls -l 명령어처럼 파일의 모든 상태 정보를 프린트
- 프로그램 구성
  - main()                   메인 프로그램
  - printStat()           파일 상태 정보 프린트
  - type()                 파일 타입 리턴
  - perm()                파일 사용권한 리턴



### list2.c

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <dirent.h>
4 #include <pwd.h>
5 #include <grp.h>
6 #include <stdio.h>
7
8 char type(mode_t);
9 char *perm(mode_t);
10 void printStat(char*, char*, struct stat*);
11
12 /* 디렉터리 내용을 자세히 리스팅한다. */
13 int main(int argc, char **argv)
14 {
15     DIR *dp;
16     char *dir;
17     struct stat st;
18     struct dirent *d;
19     char path[BUFSIZ+1];
```

```
21     if (argc == 1)
22         dir = ".";
23     else dir = argv[1];
24
25     if ((dp = opendir(dir)) == NULL) // 디렉터리 열기
26         perror(dir);
27
28     while ((d = readdir(dp)) != NULL) { // 디렉터리의 각 파일에 대해
29         sprintf(path, "%s/%s", dir, d->d_name); // 파일경로명 만들기
30         if (lstat(path, &st) < 0) // 파일 상태 정보 가져오기
31             perror(path);
32         printStat(path, d->d_name, &st); // 상태 정보 출력
33         putchar('\n');
34     }
35
36     closedir(dp);
37     exit(0);
38 }
```



```
40 /* 파일 상태 정보를 출력 */
41 void printStat(char *pathname, char *file, struct stat *st) {
42
43     printf("%5d ", st->st_blocks);
44     printf("%c%s ", type(st->st_mode), perm(st->st_mode));
45     printf("%3d ", st->st_nlink);
46     printf("%s %s ", getpwuid(st->st_uid)->pw_name,
47             getgrgid(st->st_gid)->gr_name);
48     printf("%9d ", st->st_size);
49     printf("%.12s ", ctime(&st->st_mtime)+4);
50     printf("%s", file);
51 }
```

```
52 /* 파일 타입을 리턴 */
53 char type(mode_t mode) {
54
55     if (S_ISREG(mode))
56         return('-');
57     if (S_ISDIR(mode))
58         return('d');
59     if (S_ISCHR(mode))
60         return('c');
61     if (S_ISBLK(mode))
62         return('b');
63     if (S_ISLNK(mode))
64         return('l');
65     if (S_ISFIFO(mode))
66         return('p');
67     if (S_ISSOCK(mode))
68         return('s');
69 }
```



## list2.c

```
71 /* 파일 사용권한을 리턴 */
72 char* perm(mode_t mode) {
73     int i;
74     static char perms[10] = "-----";
75
76     for (i=0; i < 3; i++) {
77         if (mode & (S_IREAD >> i*3))
78             perms[i*3] = 'r';
79         if (mode & (S_IWRITE >> i*3))
80             perms[i*3+1] = 'w';
81         if (mode & (S_IEXEC >> i*3))
82             perms[i*3+2] = 'x';
83     }
84     return(perms);
85 }
```

st\_mode

사용 권한



## 디렉토리 만들기

- mkdir() 시스템 호출
  - path가 나타내는 새로운 디렉터리를 만든다.
  - "." 와 ".." 파일은 자동적으로 만들어진다

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

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

```
int mkdir (const char *path, mode_t mode );
```

새로운 디렉터리 만들기에 성공하면 0, 실패하면 -1을 리턴한다.



## 디렉토리 삭제

- rmdir() 시스템 호출
  - path가 나타내는 디렉터리가 비어 있으면 삭제한다.

```
#include <unistd.h>
```

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

디렉터리가 비어 있으면 삭제한다. 성공하면 0, 실패하면 -1을 리턴



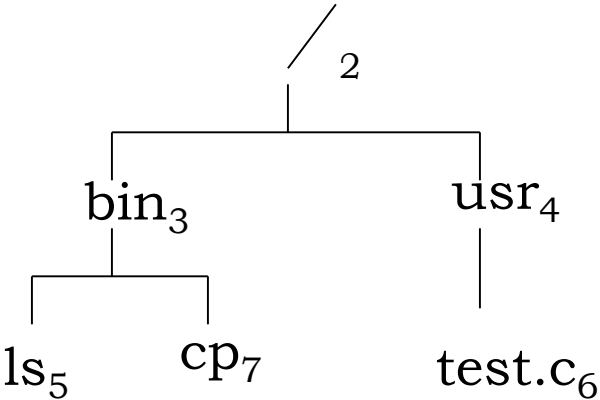
## 디렉토리 구현

- 파일 시스템 구조를 보자.
  - 디렉터리를 위한 구조는 따로 없다.
- 파일 시스템 내에서 디렉터리를 어떻게 구현할 수 있을까?
  - 디렉터리도 일종의 파일로 다른 파일처럼 구현된다.
  - 디렉터리도 다른 파일처럼 하나의 i-노드로 표현된다.
  - 디렉터리의 내용은 디렉터리 엔트리(파일이름, i-노드 번호)





디렉토리 구현



/usr/test.c

| i-노드<br>번호 | 블록<br>번호 | 블록<br>포인터        | 사용권한           |
|------------|----------|------------------|----------------|
| 1          |          |                  |                |
| 2          |          | 200              | dr-xr-xr-x ... |
| 3          |          | 201              | dr-xr-xr-x ... |
| 4          |          | 202              | dr-xr-xr-x ... |
| 5          |          | 203              | -r-xr-xr-x ... |
| 6          |          | 204,206          | -r-xr-xr-x ... |
|            |          |                  |                |
|            | 200      | .                | 2              |
|            |          | bin              | 3              |
|            |          | usr              | 4              |
|            | 201      | .                | 3              |
|            |          | ls               | 5              |
|            |          | cp               | 7              |
|            | 202      | .                | 4              |
|            |          | test.c           | 6              |
|            | 203      | ls 실행파일          |                |
|            | 204      | test.c : 첫 번째 블록 |                |
|            | 205      | cp 실행파일          |                |
|            | 206      | test.c : 두 번째 블록 |                |



## 링크

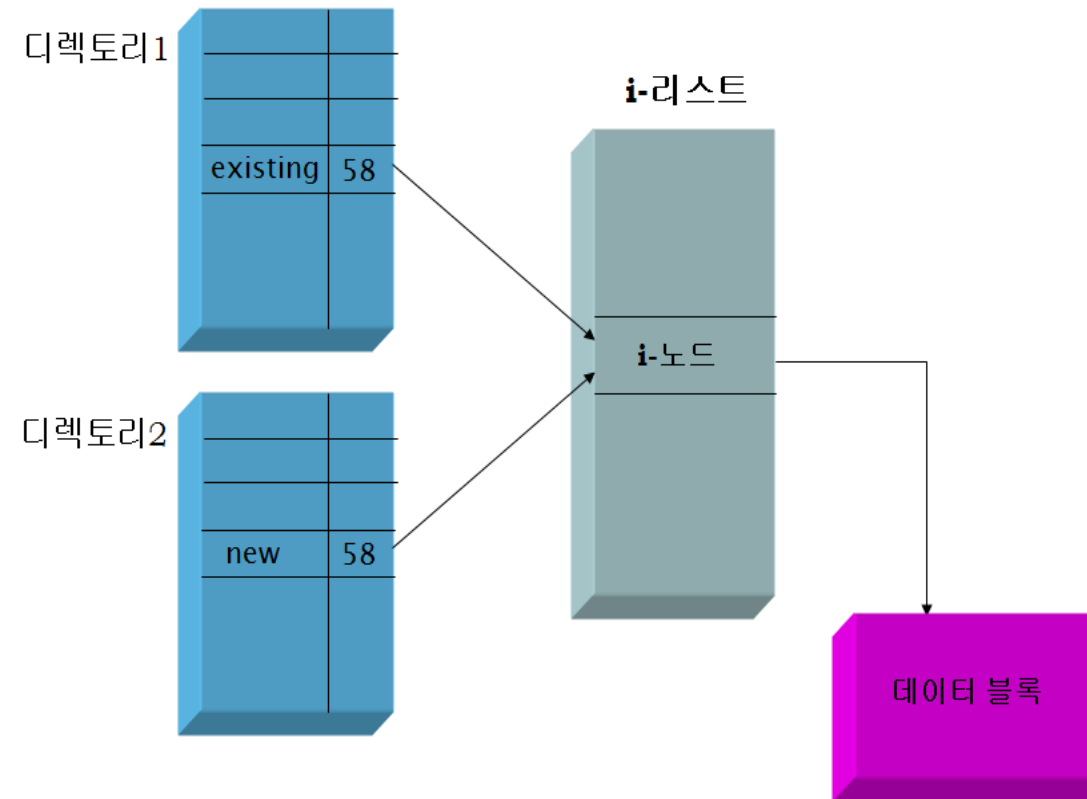
- 링크는 기존 파일에 대한 또 다른 이름으로
- 하드 링크와 심볼릭(소프트) 링크가 있다.

```
#include <unistd.h>  
  
int link(char *existing, char *new);  
  
int unlink(char *path);
```



## 링크의 구현

- link() 시스템 호출
  - 기존 파일 existing에 대한 새로운 이름 new 즉 링크를 만든다.



## link.c

```
#include <unistd.h>
int main(int argc, char *argv[ ])
{
    if (link(argv[1], argv[2]) == -1) {
        exit(1);
    }
    exit(0);
}
```



## unlink.c

```
#include <unistd.h>
main(int argc, char *argv[ ])
{
    int unlink( );
    if (unlink(argv[1]) == -1 {
        perror(argv[1]);
        exit(1);
    }
    exit(0);
}
```



## 하드 링크 vs 심볼릭 링크

- 하드 링크(hard link)
  - 지금까지 살펴본 링크
  - 파일 시스템 내의 i-노드를 가리키므로
  - 같은 파일 시스템 내에서만 사용될 수 있다
- 심볼릭 링크(symbolic link)
  - 소프트 링크(soft link)
  - 실제 파일의 경로명 저장하고 있는 링크
  - 파일에 대한 간접적인 포인터 역할을 한다.
  - 다른 파일 시스템에 있는 파일도 링크할 수 있다.



## 심볼릭 링크

```
int symlink (const char *actualpath, const char *sympath );
```

심볼릭 링크를 만드는데 성공하면 0, 실패하면 -1을 리턴한다.

```
#include <unistd.h>
int main(int argc, char *argv[ ])
{
    if (symlink(argv[1], argv[2]) == -1) {
        exit(1);
    }
    exit(0);
}
```



## 심볼릭 링크 내용

```
#include <unistd.h>
```

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

path 심볼릭 링크의 실제 내용을 읽어서 buf에 저장한다.

성공하면 buf에 저장한 바이트 수를 반환하며 실패하면 -1을 반환한다.





## rlink.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[ ])
{
    char buffer[1024];
    int nread;
    nread = readlink(argv[1], buffer, 1024);
    if (nread > 0) {
        write(1, buffer, nread);
        exit(0);
    } else {
        fprintf(stderr, "오류 : 해당 링크 없음\n");
        exit(1);
    }
}
```



## 핵심 개념

- 표준 유닉스 파일 시스템은 부트 블록, 슈퍼 블록, i-리스트, 데이터 블록 부분으로 구성된다
- 파일 입출력 구현을 위해서 커널 내에 파일 디스크립터 배열, 파일 테이블, 동적 i-노드 테이블 등의 자료구조를 사용한다.
- 파일 하나당 하나의 i-노드가 있으며 i-노드 내에 파일에 대한 모든 상태 정보가 저장되어 있다.
- 디렉터리는 일련의 디렉터리 엔트리들을 포함하고 각 디렉터리 엔트리는 파일 이름과 그 파일의 i-노드 번호로 구성된다.
- 링크는 기존 파일에 대한 또 다른 이름으로 하드 링크와 심볼릭(소프트) 링크가 있다.



## Reference

- ✓ 리눅스 프로그래밍, 창병모, 생능출판
- ✓ <https://www.44bits.io/ko/post/wsl2-install-and-basic-usage>

