9장. 문맥상의 File, Directory, Device File

목 차

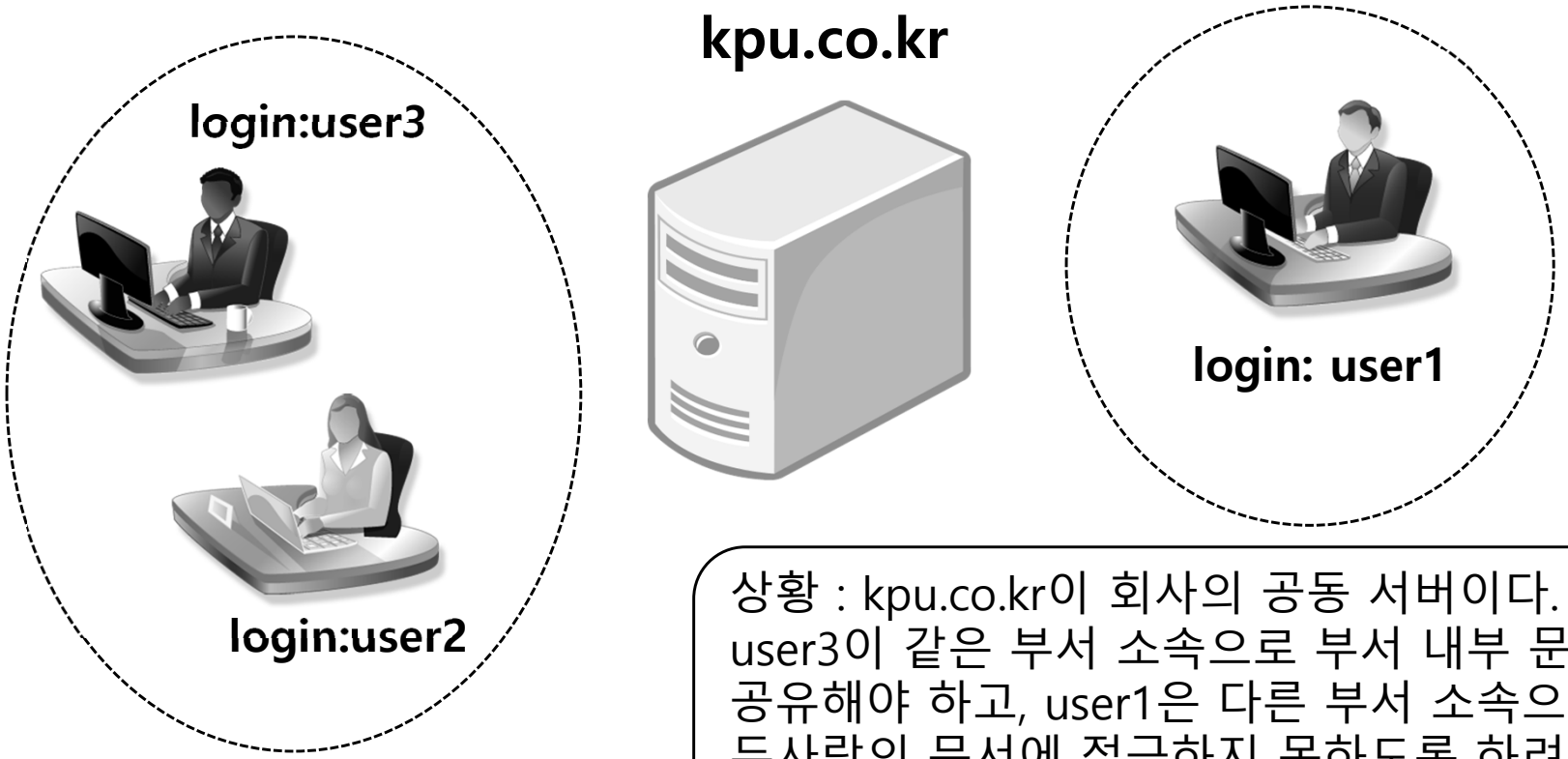
User id, group id

- File's ownership
  - Each file has a user id(uid) and a group id(gid)
- User id
  - /etc/passwd 파일에 정의

    bluewing:x:35:10:/usr/home/bluewing:bin/sh

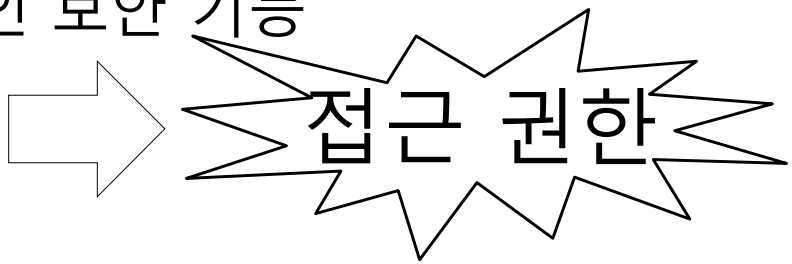
    (user name:passwd 표시:uid:gid:home directory:login shell)
  - Super user
    - user name = root
    - uid = 0
- Group id
  - /etc/group에 정의

다중 사용자 환경



상황 : kpu.co.kr이 회사의 공동 서버이다. user2와 user3이 같은 부서 소속으로 부서 내부 문서를 공유해야 하고, user1은 다른 부서 소속으로 다른 두사람의 문서에 접근하지 못하도록 하려면 어떻게 관리해야 할까?

- ❖ 파일을 읽고, 쓰고, 실행할 수 있는 권한
- ❖ 다중 사용자 시스템의 가장 기본적인 보안 기능



다중 사용자 환경

❖ 유닉스의 사용자 카테고리

- 소유자(Owner)
  일반적으로 파일을 생성한 사용자. 명령을 통해 변경할 수도 있음
- 그룹(Group)
  파일과 동일한 그룹에 속한 사용자들.
  파일이 속한 그룹?
    일반적으로 파일을 생성한 사용자의 기본 그룹
    상위 디렉토리에 특수한 권한(setgid)이 부여된 경우 다른 그룹으로 지정 명령을 이용하여 변경 가능
- 기타 사용자(Others)
  소유자나 그룹 카테고리에 속하지 않은 모든 사용자들

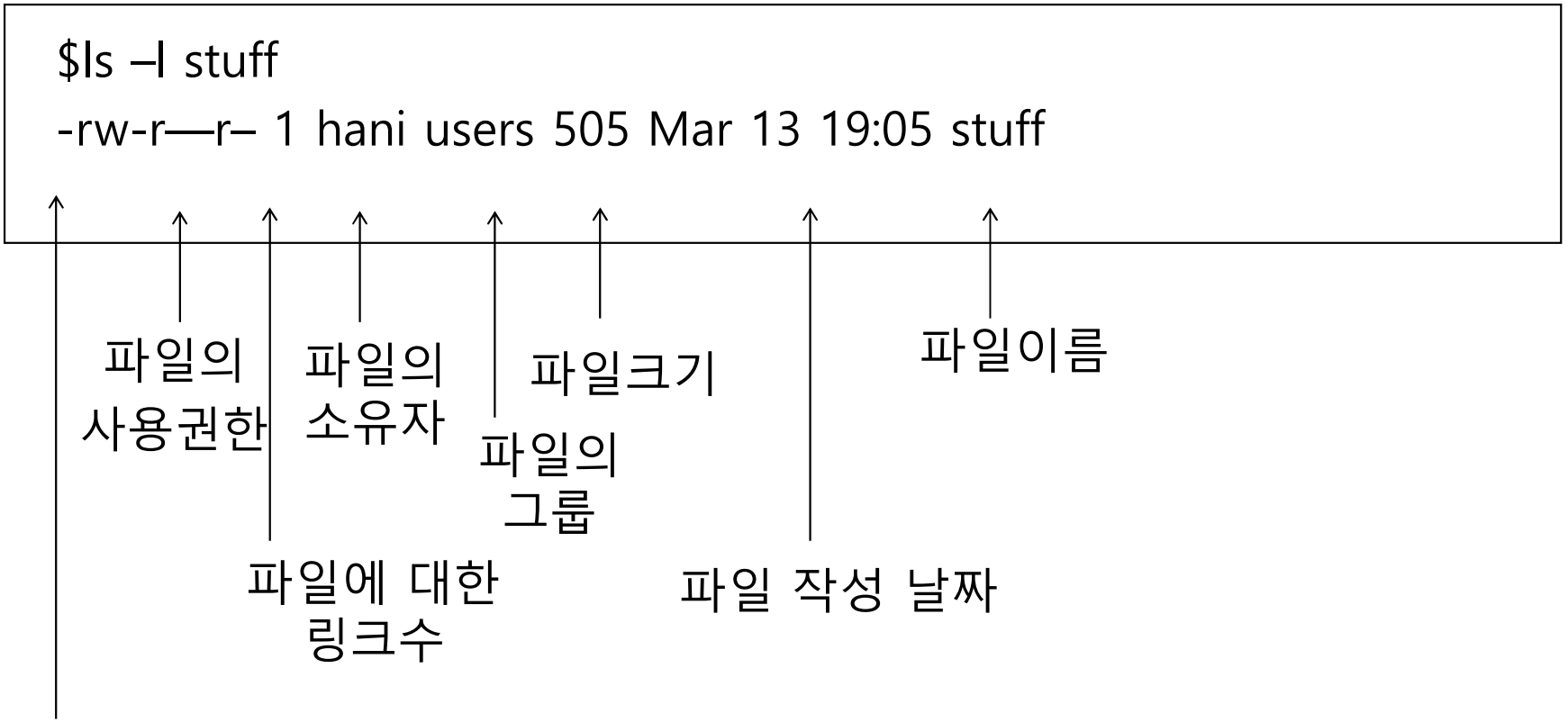
❖ 접근 권한은 카테고리 별로 다르게 부여

File Permissions

- UNIX 시스템은 멀티유저 시스템
- 한 사용자의 사용이 다른 사용자에게 피해가 되지 않도록 보호 장치를 마련 : 사용 권한 (permission)

사용 권한	파일의 경우	디렉토리의 경우
읽 기	파일 내용을 볼 수 있음	디렉토리 내용을 볼 수 있음
쓰 기	파일 내용 수정 가능	해당 디렉토리 내에서 새 파일 수정 가능
실행	파일 실행 가능	cd 명령으로 해당 디렉토리로 이동 가능

File Permissions



File Permissions

- 사용권한

r w x	r w x	r w x
소유자권한	그룹권한	다른 사용자 권한

r	읽기 권한
w	쓰기 권한
x	실행 권한
-	해당 권한 없음

예) -rwxr-x--x -rw------- -rwxrwxrwx

File Permissions

- 기호에 의한 파일 사용 권한 설정
- 형식 : chmod {a,u,g,o} {+-} {r,w,x} filenames
- 설명

  - a는 all을 의미
  - u는 user, 파일소유자
  - g는 group, 그룹
  - o는 other, 다른 사용자
  - +는 기능 설정
  - -는 기능 제거
  - r는 read 속성
  - w는 write 속성
  - x는 excute 속성
  - filenames는 속성이 설정될 파일 이름

File Permissions

- chmod 명령의 사용 예
  - $chmod a+r stuff
  - $chmod +r stuff
  - $chmod og-x stuff
  - $chmod u+rwx stuff
  - $chmod o-rwx stuff

- 8진수를 사용한 예
  - $chmod 444 stuff
  - $chmod 200 stuff

File Permissions

- euid, ruid, egid
  - effective user-id (euid, 유효 사용자 식별번호): 실제 소유권을 갖는 uid
  - real user-id (ruid, 진짜 사용자 식별번호): 프로세스를 실행시킨 uid

- Permissions and file modes in <sys/stat.h>

value	symbolic mode	permission
0400	S_IRUSR	read by owner
0200	S_IWUSR	write by owner
0100	S_IXUSR	execute by owner
0040	S_IRGRP	read by group
0020	S_IWGRP	write by group
0010	S_IXGRP	execute by group
0004	S_IROTH	read by other
0002	S_IWOTH	write by other
0001	S_IXOTH	execute by other

File Permissions

- Extra permission for executable files
  - 04000 S_ISUID set uid on execution (set user-id)
  - 02000 S_ISGID set gid on execution (set group-id)
  - 01000 S_ISVTX save text image (sticky bit)

cf. /bin/passwd : -r-sr-sr-x root sys

File Creation Mask and umask System Call

- Each process has a file creation mask

  fd=open(pathname, O_CREAT, mode);

  actually means

  fd=open(pathname, O_CREAT, (~mask)&mode);

  – 예: 현재 umask 값이 07인 경우
    - fd = open("/tmp/newfile", O_CREAT, 0644);
    - 실제 file mode는 0640이 된다.

- The umask system call

  #include <sys/types.h>

  #include <sys/stat.h>

  mode_t umask(mode_t newmask);

  – newmask: new mask to set
  – Return: old mask

umask의 예

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

int specialcreat (const char *pathname, mode_t mode)
{
  mode_t oldu;
  int filedes:
  /* 화일 생성 마스크를 0으로 설정 */
  if ( (oldu = umask(0)) == 1)
  {
    perror ("saving old mask");
    return (-1);
 }
 /* 화일을 생성한다 */
 if((filedes=open(pathname, O_WRONLY | O_CREAT | O_EXCL, mode))== -1)
     perror ("opening file");
 /* 비록 개방에 실패하더라도, 과거의 화일 모드를 복원한다. */
 if (umask (oldu) == -1)
     perror ("restoring old mask");
 /* 화일 기술자를 복귀한다. */
 return filedes;
}
```

Open과 file 허가 오류

- open 과 파일허가 오류 번호: errno
  - EACCESS error(허가가 거부됨)
  - EEXIST error(file이 이미 존재함)
  - 예

```
fd=open(pathname, O_WRONLY|O_CREAT|O_TRUNC, 0600);
fd=open(pathname, O_WRONLY|O_CREAT|O_EXCL, 0600);
```

Determining File Accessibility

- Check if a process can access the given file
  - Effective uid가 아닌 real uid에 준하여 process가 file에 접근할 수 있는가를 알려줌.

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

  - mode: R_OK, W_OK, X_OK
  - return:
    - success: 0
    - fail: -1

Determining File Accessibility

```
/*access의 사용 예 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
  char *filename = "afile";
  if (access (filename, R_OK) == -1)
  {
    fprintf (stderr, "User cannot read file %s₩n", filename);
    exit (1);
  }
 printf ("%s readable, proceeding₩n", filename);
  /* 프로그램의 나머지 부분... */
}
```

Changing Permission and Ownership

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int chmod(const char *pathname, mode_t newmode);
int chown(const char *pathname, uid_t uid, gid_t gid);
```

- newmode: permission mode
- uid: user id
- gid: group id
- return:
  - success: 0
  - fail: -1
- file의 소유자나 super user만 사용 가능

다수의 이름을 갖는 file(link)

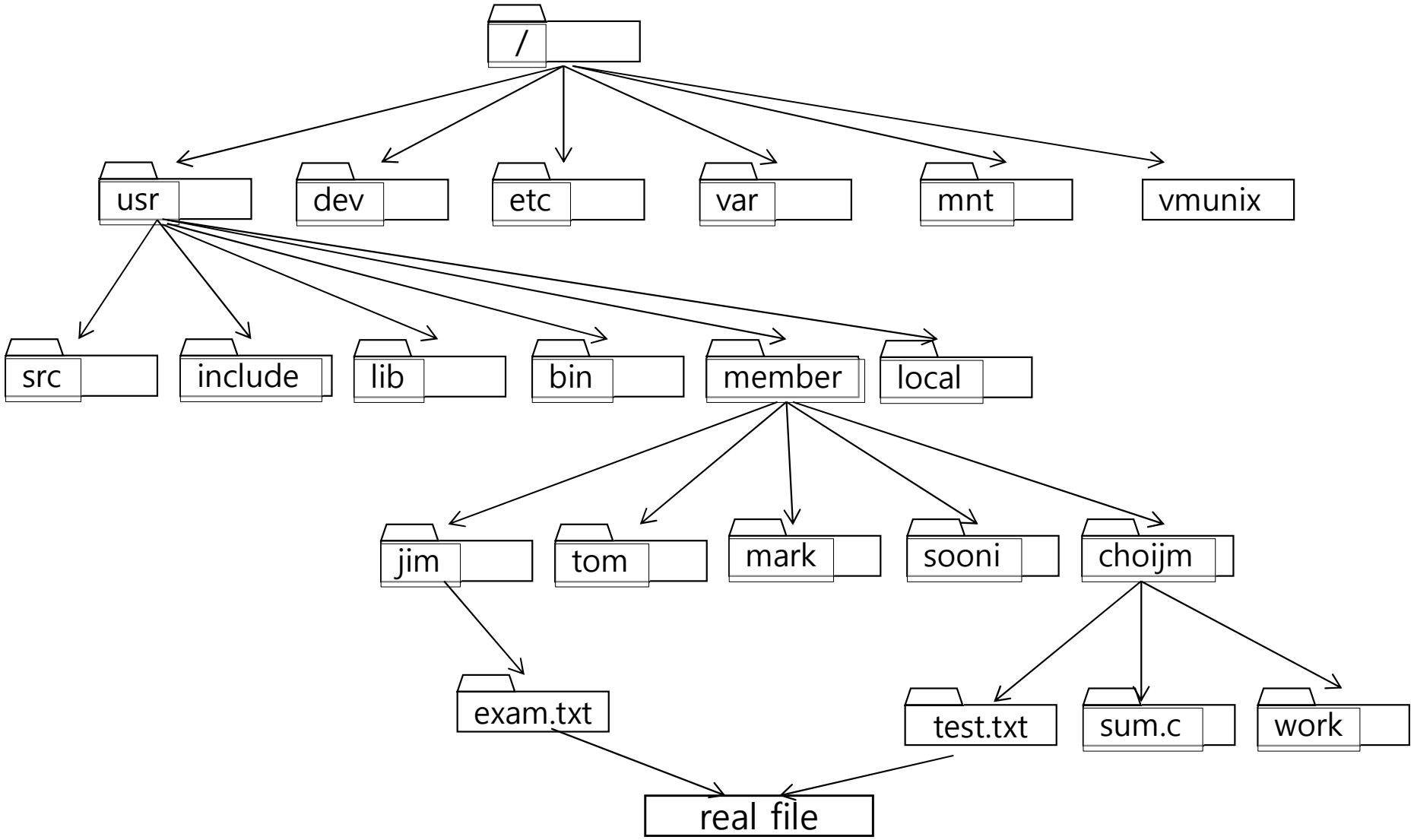
- Hard link and link count (링크 계수)
  - A file with multiple names can have many link counts
  - This saves disk space and ensures many people can access the same file
  - link and unlink system calls
- The link system call

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

  - return:
    - success: 0
    - fail: -1
- How to move a file
  - Use link and unlink system calls
  - Use rename system call

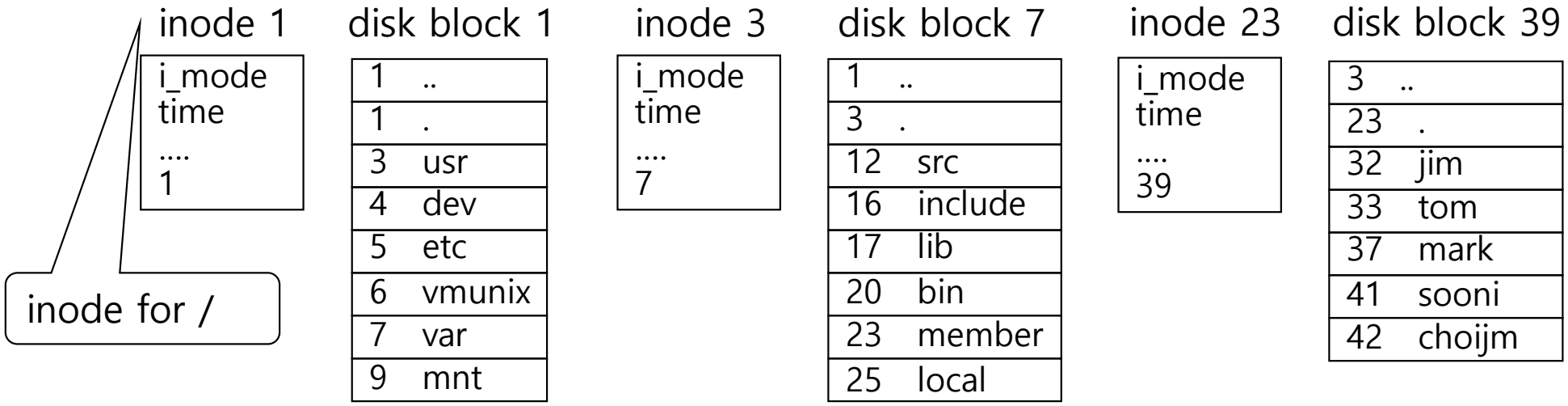
다수의 이름을 갖는 file(link)

- 파일 계층 구조 (hierarchical structure)

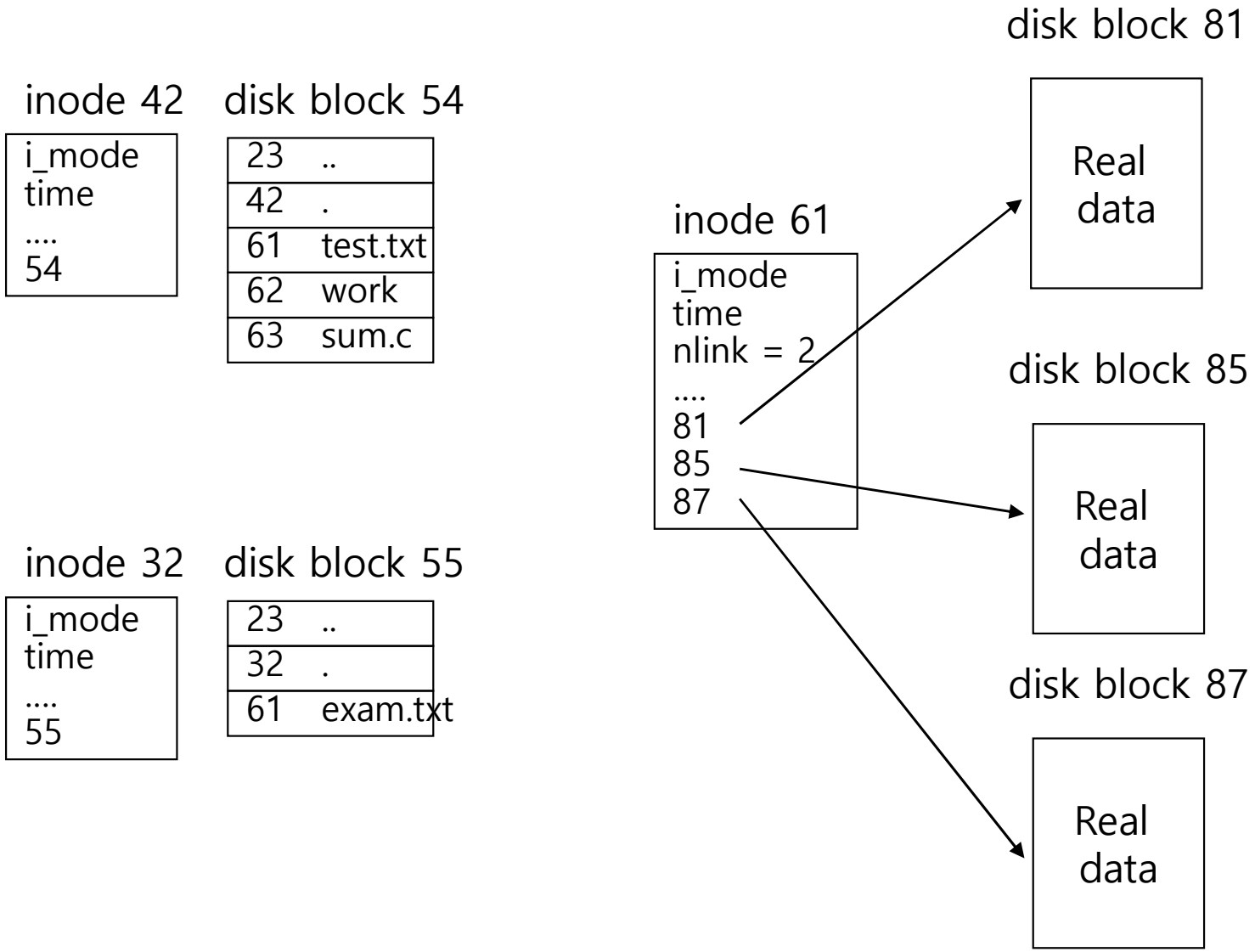


다수의 이름을 갖는 file(link)

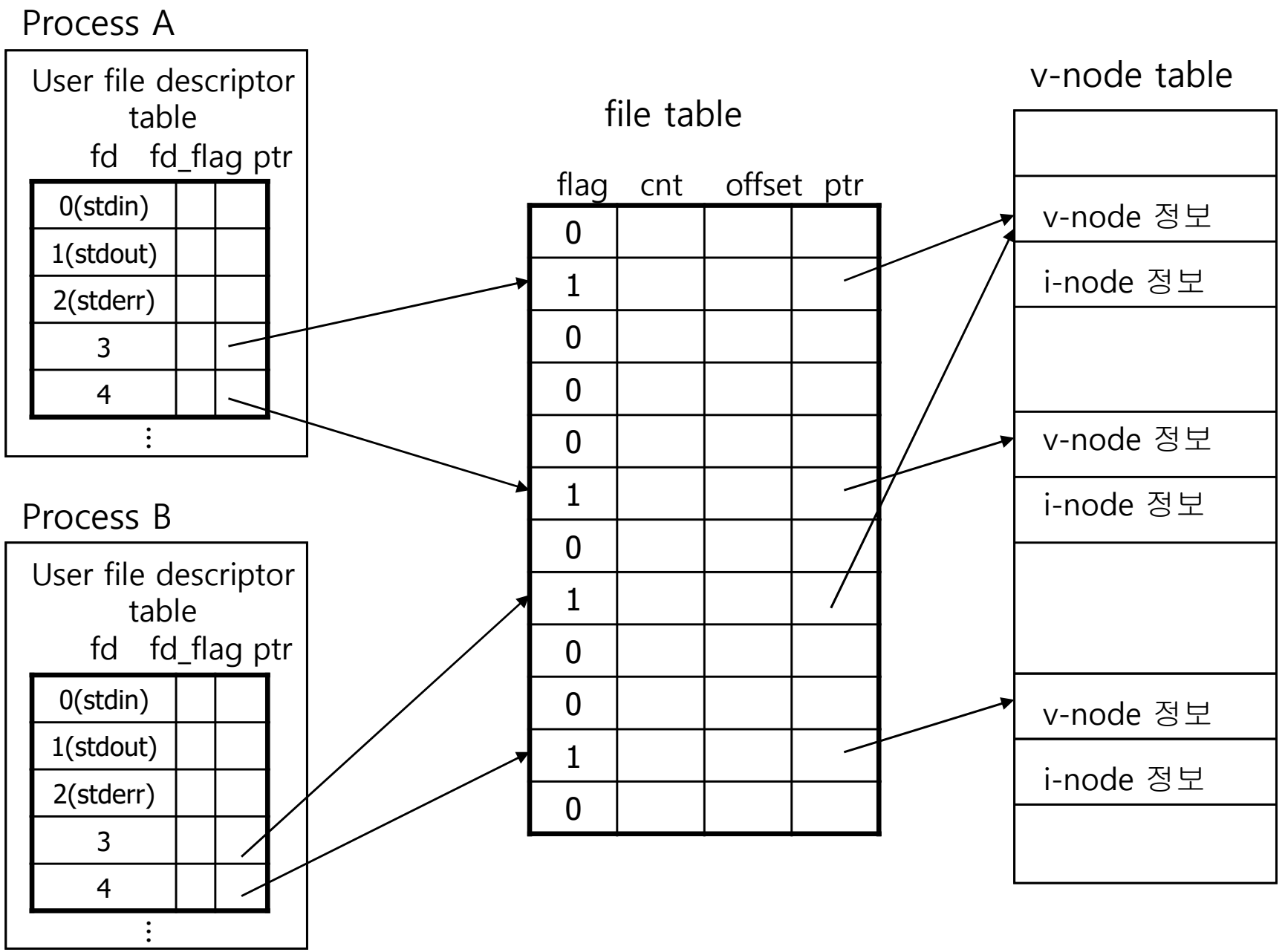
- link("/usr/member/choijm/test.txt","/usr/member/jim/exam.txt");



다수의 이름을 갖는 file(link)



다수의 이름을 갖는 file(link)



Program 예: move

```
/* move -- 한 화일을 하나의 경로이름으로부터 다른 경로이름으로 옮긴다. */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

char *usage = "usage: move file1 file2₩n";

/* main은 명령줄에 의해 표준적인 방법으로 전달된 인수를 사용한다. */
main (int argc, char **argv)
{
 if (argc != 3)
 {fprintf (stderr, usage); exit (1);}

if ( link (argv[1], argv[2] == -1)
 {  perror ("link failed");exit (1); }

 if (unlink (argv[1] == -1)
 {
    perror ("unlink failed"); unlink (argv[2]); exit (1);
 }
```

rename System Call

include <stdio.h>

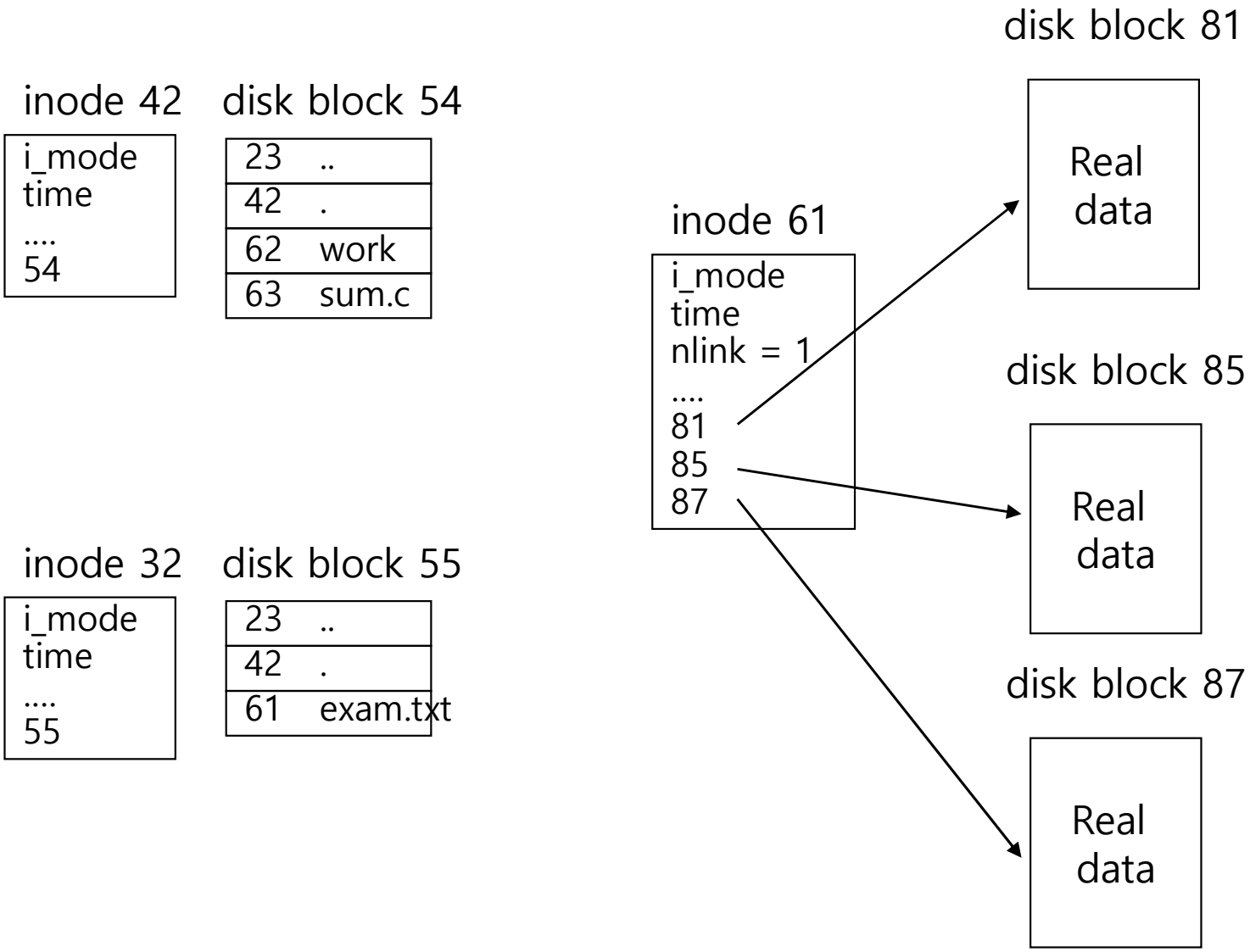
int rename(const char *oldpathname, const char *newpathname);

- return:
  - success: 0
  - fail: -1

- regular file과 directory 이름의 재지정

unlink System Call

unlink("/usr/member/choijm/test.txt");



Symbolic Link

- Hard link's limitations
  - directory link and link across file systems are not allowed
  - Symbolic link 는 그 자체가 하나의 file임(자신이 링크되어 있는 file에 대한 경로 수록)
  - Symbolic link에 의해 가르켜지고 있는 file 제거시: link가 끊어짐.
- The symlink and readlink system calls

```
#include <unistd.h>
int symlink(const char *realname, const char *symname);
int readlink(const char *sympath, char *buffer, size_t bufsize);
```

  - return of symlink:
    - success: 0, fail: -1
  - symname 그 자체에 들어있는 데이터를 볼 경우 readlink를 사용
  - buffer: place to put the result pathname (not NULL terminating)
  - bufsize: the size of buffer
  - return of readlink:
    - success: # of characters, fail: -1

File 정보의 획득: stat와 fstat

- The stat and fstat system calls

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *pathname, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

  - return:
    - success: 0
    - fail: -1

File 정보의 획득: stat와 fstat

- Each file's properties are in the

```
struct stat {
        dev_t           st_dev;     /* the logical device */
        ino_tst_ino;    /* inode number */
        mode_t          st_mode;  /* permission and file type */
        nlink_t         st_nlink;   /* # of hard links */
        uid_tst_uid;    /* user id */
        gid_tst_gid;    /* group id */
        dev_t           st_rdev;    /* device if file is device */
        off_t st_size;  /* logical file size */
        time_t          st_atime;  /* last data read time */
        time_t          st_mtime; /* last data write time */
        time_t          st_ctime;  /* last stat write time */
        long st_blksize;/* I/O block size */
        long st_blocks; /* # of physical blocks */
}
```

File 정보의 획득: stat와 fstat

- Stat와 fstat의 예

```
struct stat s;
int filedes, retval;

filedes = open("tmp/dina", O_RDWR);

/* s는 이제 아래의 명령이나 ... */
retval = stat("/tmp/dina", &s);

/* 또는 아래의 명령에 의해 채워질 수 있다. */
retval = fstat(filedes, &s);
```

stat와 fstat 예: filedata(1)

```
/* filedata -- 한 화일에 관한 정보를 출력 */

#include <stdio.h>
#include <sys/stat.h>
/* 허가 비트가 설정되어 있는지 결정하기 위해 octarray를 사용 * /
static short octarray[9] = { 0400, 0200, 0100, 0040, 0020, 0010, 0004, 0002, 0001};

/* 화일 허가에 대한 기호화 코드 끝부분의 null 때문에 길이가 10문자이다. */
static char perms[10] = "rwxrwxrwx";
int filedata (const char *pathname)
{
 struct stat statbuf;
 char descrip[10];
 int j:
 if(stat (pathname, &statbuf) == -1)
 {
      fprintf (stderr, "Couldn't stat %s₩n", pathname);
      return (-1);
 }
```

stat와 fstat 예: filedata(2)

```
/* 허가를 읽기 가능한 형태로 바꾼다. */
 for(j=0; j<9; j++)
 {
      / * 비트별 AND를 사용하여 허가가 설정되었는지 테스트 */
      if (statbuf.st_mode & octarray[j])
                descrip[j] = perms[j];
      else
                descrip[j] = '-';
 }
 descrip[9] = '₩0'; /* 하나의 문자열을 가지도록 확인 */

 /* 화일 정보를 출력한다. */
 printf ("₩nFile %s :₩n", pathname);
 printf ("Size %ld bytes₩n", statbuf.st_size);
 printf ("User-id %d, Group-id %d₩n₩n", statbuf.st_uid,
        statbuf.st_gid);
 printf ("Permissions: %s₩n", descrip);
 return (0);
}
```

stat와 fstat 예: lookout(1)

```
/* lookout -- 화일이 변경될 때 메시지를 프린트 */
        #include <stdlib.h>
        #include <stdio.h>
        #include <sys/stat.h>
        #define MFILE     10
        void cmp(const char *, time_t);
        struct stat sb;

        main (int argc, char **argv)
        {
         int j;
         time_t last_time[MFILE+1];
         if(argc < 2)
         {
             fprintf (stderr, "usage: lookout filename ...₩n"); exit (1);
         }
         if(argc > MFILE)
         {
             fprintf (stderr, "lookout: too many filenames₩n"); exit (1);
         }
```

stat와 fstat 예: lookout(2)

```
/* 초기화 */
 for (j=1; j<=argc; j++)
 {
     if (stat (argv[j], &sb) == -1)
     { fprintf (stderr, "lookout: couldn't stat %s₩n", argv[j]); exit (1);}
     last_time[j] = sb.st_mtime;
 }
 /* 화일이 변경될 때까지 루프 */
 for (;;)
 {
     for (j=1; j<=argc; j++)
         cmp (argv[j], last_time[j]);
     /*
      * 60초간 쉰다.
      * "sleep"는 표준 UNIX
      * 라이브러리 루틴이다.
      */
    sleep (60);
 }
}
```

stat와 fstat 예: lookout(3)

```
void cmp(const char *name, time_t last)
{
 /* 파일에 관한 통계를 읽을 수 있는 한 변경시간을 검사한다. */
 if (stat(name, &sb) == 1 || sb.st_mtime ! = last)
 {
     fprintf (stderr, "lookout: %s changed₩n", name);
     exit (0);
 }
}
```

stat와 fstat 예: addx

```
/* addx -- 화일에 수행허가를 추가 */
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#define XPERM  0100        / *  소유자에 대한 수행 허가 */
main(int argc, char **argv)
{
 int k; struct stat statbuf;
 /* 인수 리스트의 모든 화일에 대해 루프 */
 for (k=1; k<argc; k++)
 {
     /* 현행 화일 모드를 얻음 */
     if (stat (argv[k], &statbuf) == -1)
     { fprintf (stderr, "addx: couldn't stat %s₩n", argv[k]); continue;}
     /* 비트별 OR 연산을 사용하여 수행허가의 추가를 시도 */
     statbuf.st_mode | = XPERM;
     if (chmod (argv[k], statbuf.st_mode) == -1)
         fprintf (stderr, "addx: couldn't change mode for %s₩n", argv[k]);
 } /* 루프의 끝 */
 exit (0);
}
```

Directory – User's View

- *Home directory* - The directory where a user is placed at login.
- *Current working directory* - The directory where a user is currently working at.
- File's pathname
  - Absolute - from the root directory (eg: /home/john/book)
  - Relative - from the currently working directory (eg: ../john/book)

Directory

- A directory is a list of an unique inode number and a file/directory name
- An inode number represents an inode structure which contains file or directory's stat info (uid, gid, permission, size, date, ...) and address to actual data blocks
- A directory contains a current directory(.) and a parent directory(..).
- A directory can hold other directory as a subdirectory

Directory Permissions

- Directory permission consists of read/write/execute.
- Directory permission is interpreted differently from file permission.
  - Read - One can list file name or subdirectory name within the directory. (This does not mean one can read files)
  - Write - One can create new files and remove existing files within the directory. (This does not mean one can modify files)
  - Execute (search) - One can move into this directory using cd command or chdir system call. (To open a file or execute a file, one should have execute permissions on all directories leading to the file)

Directory의 생성 및 제거

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

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

- Return of mkdir:
  - Success: 0
  - Fail: -1
- Return of rmdir:
  - Success: 0
  - Fail: -1

Directory의 열기 및 닫기

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *dirname);

#include <dirent.h>
int closedir(DIR *dirptr);
```

- Return of opendir:
  - Success: DIR *
  - Fail: NULL
- Return of closedir:
  - Success: 0
  - Fail: -1

Directory의 열기 및 닫기

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

main()
{
DIR *dp;
  if ((dp = opendir("/tmp/dir1")) == NULL)
  {
     fprintf (stderr, "Error on opening directory /tmp/dir1₩n");
       exit (1);
   }
   /* 디렉토리에 대한 코드를 처리한다. */
  .
  .
  closedir(dp);
}
```

Directory 읽기

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dirptr);
void rewinddir(DIR *dirptr);
```

- Return of readdir:
  - Success: struct dirent *
  - Fail (if no directory entry): NULL
- Return of rewinddir:
  - None

- struct dirent consists of d_ino and d_name.

Directory 예: my_double_ls

```
#include <dirent.h>
int my_double_ls (const char *name)
{
  struct dirent *d;
  DIR *dp;
  /* 디렉토리를 개방하고, 실패여부를 점검함 */
  if ((dp=opendir(name)) == NULL)   return (-1);
  /* 디렉토리를 살피면서 루프를 계속한다. 이때 inode 번호가 유효하면 디렉토리항을 프린트한다.     */
  while (d = readdir(dp))  {
    if (d->d_ino !=0)
        printf ("%s₩n", d->d_name);
  }
  /*이제 디렉토리의 시작으로 되돌아간다 ... */
  rewinddir(dp);
  /* ... 그리고 디렉토리를 다시 프린트한다. */
  while (d = readdir(dp))  {
    if (d->d_ino != 0)
        printf ("%s₩n", d->d_name);
  }
  closedir (dp);
  return (0);
}
```

Directory 예: find_entry(1)

```
#include <stdio.h> /* NULL을 정의 */
#include <dirent.h>
#include <string.h> /* 스트링 함수를 정의 */

int match(const char *, const char *);

char *find_entry(char *dirname, char *suffix, int cont)
{
  static DIR *dp=NULL;
  struct dirent *d;

  if (dp == NULL || cont == 0){
    if (dp != NULL}
          closedir (dp);
     if ((dp = opendir (dirname)) == NULL)
          return (NULL);
  }
```

Directory 예: find_entry(2)

```
 while (d = readdir(dp)) {
     if (d->d_ino == 0)
         continue;
     if (match (d->d_name, suffix))
         return (d->d_name);
  }
  closedir (dp);
  dp = NULL;
  return (NULL);
}

int match (const char *s1, const char *s2){
  int diff = strlen(s1)- strlen(s2);
  if (strlen(s1) > strlen(s2))
       return (strcmp(&s1[diff], s2) == 0);
  else
       return (0);
}
```

Current Working Directory

```
#include <unistd.h>
int chdir(const char *path);
char *getcwd(char *name, size_t size);
```

- Return of chdir:
  - Success: 0
  - Fail: -1

- name: place where the current directory name is copied into
- Return of getcwd:
  - Success: name
  - Fail: NULL

Current Working Directory 예: my_pwd

```
/* my_pwd -- 작업 디렉토리를 프린트한다.*/
#include <stdio.h>
#include <unistd.h>
#define VERYBIG 200
void my_pwd (void);

main()
{
  my_pwd();
}

void my_pwd (void)
{
  char dirname[VERYBIG];

  if ( getcwd(dirname, VERYBIG) == NULL)
    perror("getcwd error");
else
    printf("%s₩n", dirname);
}
```

Walking a Directory Tree

```
#include <ftw.h>
int ftw(const char *path, int(*func)(), int depth);
```

- Return of ftw:
  - Success: 0
  - Fail: -1 or non-zero value returned by func
- func: a function called for each file/directory searched

```
int func (const char *name, const struct stat *sptr, int type) {
    /* 함수의 내용 */
}
```

  - type : 방문하는 객체의 type
    - FTW_F file
    - FTW_D directory
    - FTW_DNR directory that could not be read
    - FTW_SL symbolic link
    - FTW_NS not symbolic link, stat이 실행될 수 없는 객체
- depth: # of file descriptors used

Walking a Directory Tree 예: list(1)

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

int list(const char *name, const struct stat *status, int type)
{
   /* 만일 stat 호출이 실패하면, 그냥 복귀한다. */
   if (type == FTW_NS)
        return 0;
/* 아니면 객체 이름, 허가 그리고 만일 객체가 디렉토리이거나 상징형 링크이면 뒤에 "*"를 첨가한다.
    */
  if(type == FTW_F)
        printf("%-30s₩t0%3o₩n", name, status->st_mode&0777);
  else
        printf("%-30s*₩t0%3o₩n", name, status->st_mode&0777);

  return 0;
}
```

Walking a Directory Tree 예: list(2)

```
main (int argc, char **argv)
{
  int list(const char *, const struct stat *, int);

  if (argc == 1)
        ftw (".", list, 1);
  else
        ftw (argv[1], list, 1);
  exit (0);
}
```

UNIX Device Files

- *Special files* - UNIX extends the file concept to cover the peripheral devices connected to a system. These peripheral devices such as printers, disk and even memory, are represented by filenames in the file structure. These can be accessed using UNIX file I/O system calls.

UNIX Device Files

- Device files: each device has one file in /dev

  예: /dev/lp0, /dev/console, /dev/hda1

- Device files can be used as regular files

  예: $ cat file > /dev/lp0

- Block and character devices
  - Block(b) -disk, tape

  예: brw-rw---- 1 root disk 3,1 May 6 1998 /dev/hda1
  - Character(c) - terminal, modem, printer

  예: crw-rw---- 1 root daemon 6,0 May 6 1998 /dev/lp0

- Major and minor device numbers
  - Major - device driver number
  - Minor - device port number

UNIX Device Files: 예

```
#include<fcntl.h>

main()
{
 int i, fd;

 fd = open("/dev/tty00", O_WRONLY);

 for(i = 0; i< 100; i++)
        write(fd, "x", 1);

 close(fd);
}
```