File System

Jo, Heeseung

리눅스 파일의 특징[1]

파일

- 파일은 데이터 저장, 장치구동, 프로세스 간 통신 등에 사용
- 일반파일, 디렉토리, 특수파일

일반파일

- 텍스트 파일, 실행파일, 라이브러리, 이미지 등 사용하는 대부분의 파일

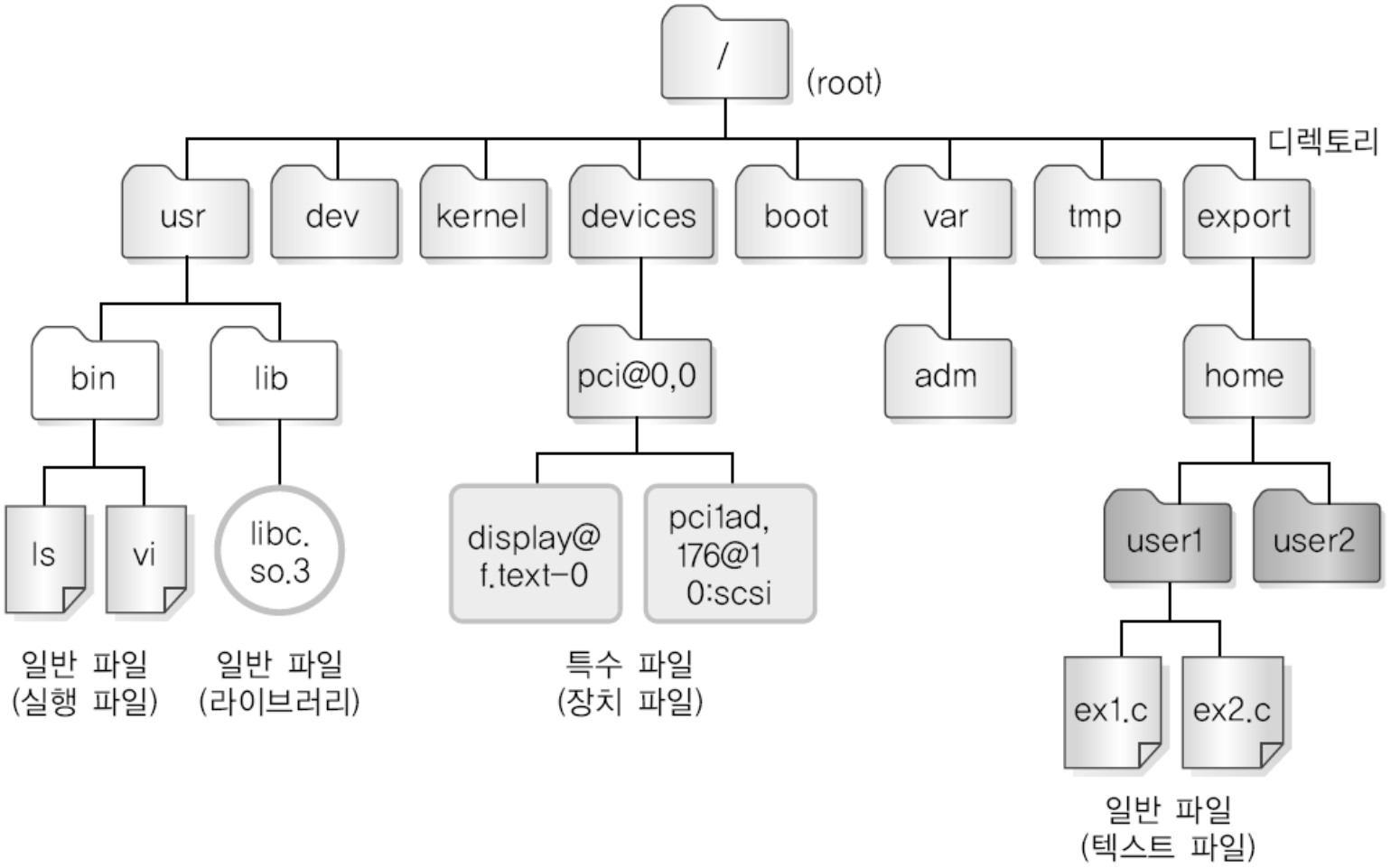
디렉토리

- 디렉토리도 파일로 취급
- 디렉토리에 속한 파일의 목록과 inode를 가진 파일

특수파일 - 장치파일

- 장치를 파일로 표현
- 예 : /dev/sda1

리눅스 파일의 특징[1]



[그림 3-1] 유닉스 파일의 종류

유닉스 파일의 특징[2]

파일의 종류 구분

- ls –l 명령으로 파일의 종류 확인
  - 결과의 맨 앞 글자로 구분

```
# ls -l /usr/bin/vi
-r-xr-xr-x   5 root     bin        193968 2007  9월 14일 /usr/bin/vi
```

- 파일 종류 식별 문자

문자	파일의 종류
–	일반 파일
d	디렉토리
b	블록 장치 특수 파일
c	문자 장치 특수 파일
l	심볼릭 링크

유닉스 파일의 특징[3]

파일의 구성 요소

- 파일명, inode, 데이터블록

파일명

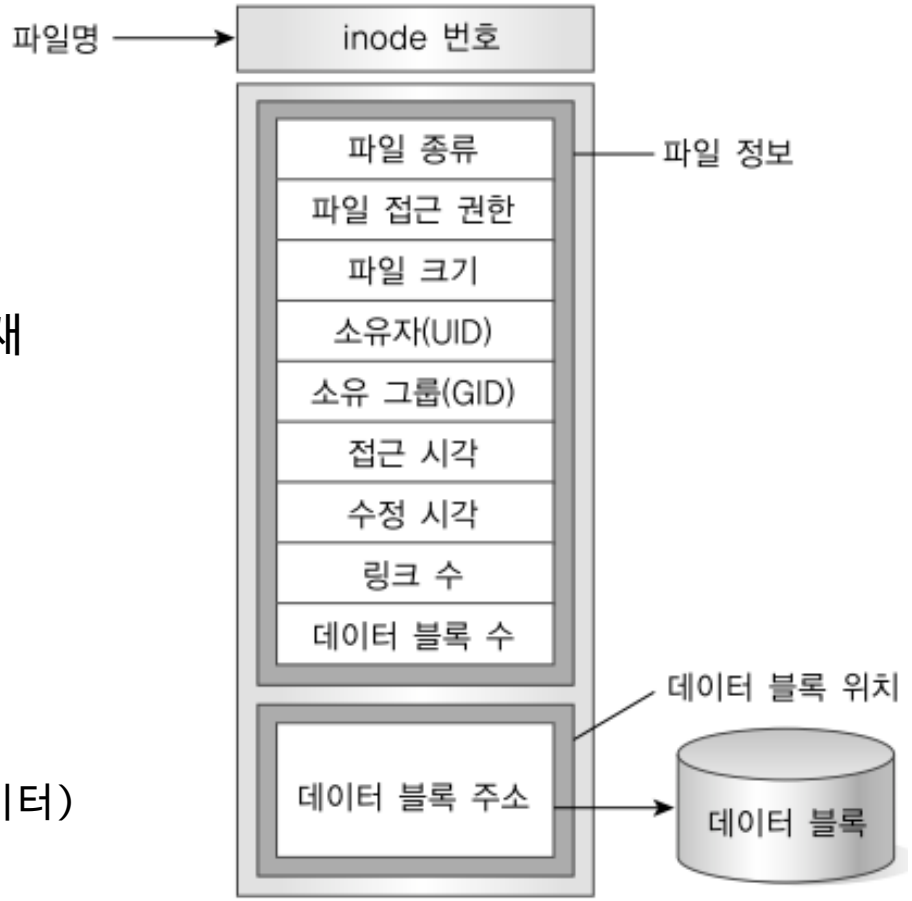
- 사용자가 파일에 접근할 때 사용
- 파일명과 관련된 inode가 반드시 존재
- 최대 255자까지 가능
- 대소문자를 구분
- '.' 으로 시작하면 숨김 파일

inode

- 번호로 표시
- 두 부분으로 나누어 정보 저장
  - 파일 정보를 저장하는 부분 (메타데이터)
  - 데이터 블록의 주소 저장하는 부분
- ls –i 명령으로 inode 번호 확인

데이터 블록

- 실제로 데이터가 저장되는 부분



파일 정보 검색[1]

파일명으로 파일 정보 검색 : stat(2)

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *restrict path, struct stat *buf);
```

- inode에 저장된 파일정보 검색
- path에 검색할 파일의 경로 지정
  - 검색한 정보를 buf에 저장
- stat 구조체

```
struct stat {
    dev_t       st_dev;
    ino_t       st_ino;
    mode_t      st_mode;
    nlink_t     st_nlink;
    uid_t       st_uid;
    gid_t       st_gid;
    dev_t       st_rdev;
    off_t       st_size;
    time_t      st_atime;
    time_t      st_mtime;
    time_t      st_ctime;
    blksize_t   st_blksize;
    blkcnt_t    st_blocks;
    char        st_fstype[_ST_FSTYPSZ];
};
```

파일명으로 inode 정보 검색하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <stdio.h>
04
05  int main(void) {
06     struct stat buf;
07
08     stat("unix.txt", &buf);
09
10     printf("Inode = %d\n", (int)buf.st_ino);
11     printf("Mode = %o\n", (unsigned int)buf.st_mode);
12     printf("Nlink = %o\n",(unsigned int) buf.st_nlink);
13     printf("UID = %d\n", (int)buf.st_uid);
14     printf("GID = %d\n", (int)buf.st_gid);
15     printf("SIZE = %d\n", (int)buf.st_size);
16     printf("Atime = %d\n", (int)buf.st_atime);
17     printf("Mtime = %d\n", (int)buf.st_mtime);
18     printf("Ctime = %d\n", (int)buf.st_ctime);
19     printf("Blksize = %d\n", (int)buf.st_blksize);
20     printf("Blocks = %d\n", (int)buf.st_blocks);
21     //printf("FStype = %s\n", buf.st_fstype);
22
23     return 0;
24  }
```

```
# ex3_1.out
Inode = 192
Mode = 100644
Nlink = 1
UID = 0
GID = 1
SIZE = 24
Atime = 1231397228
Mtime = 1231397228
Ctime = 1231397228
Blksize = 8192
Blocks = 2
```

UNIX time:
1970/1/1 00:00:00 (UTC)
부터 경과한 초

파일 정보 검색[2]

파일 기술자로 파일 정보 검색 : fstat(2)

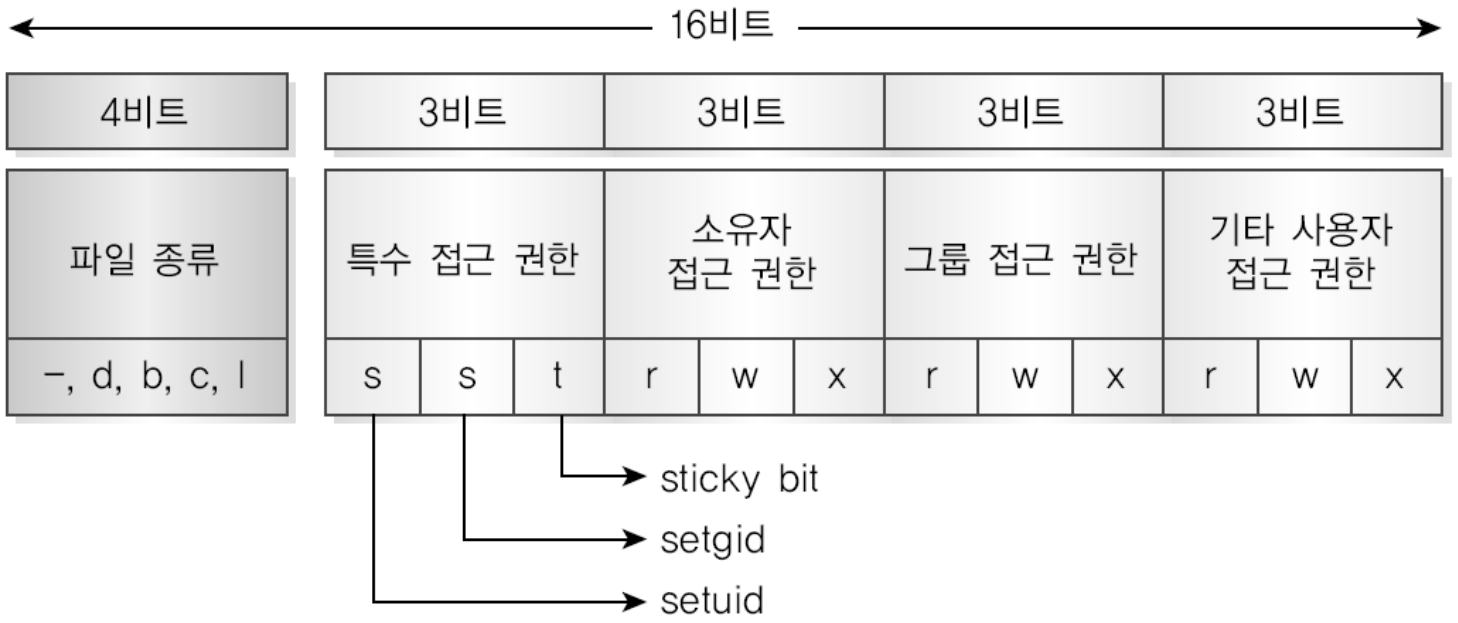
```
#include <sys/types.h>
#include <sys/stat.h>
int fstat(int fd, struct stat *buf);
```

- fd로 지정한 파일의 정보를 검색하여 buf에 저장

파일 접근권한 제어

stat구조체의 st_mode항목에 파일의 종류와 접근권한 저장

st_mode 값의 구조

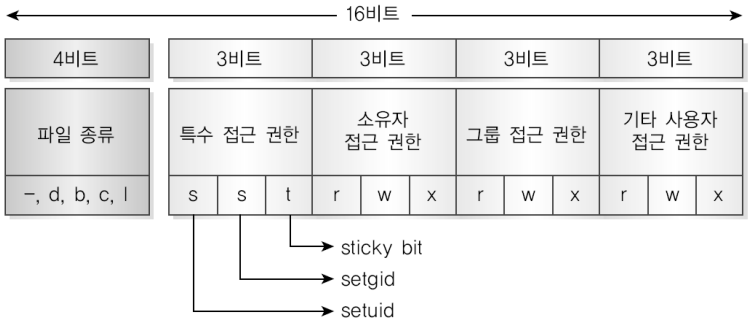


[그림 3-3] st_mode의 비트 구조

파일 접근권한 제어

chmod 명령어

- 파일/디렉토리의 접근 권한을 변경



[그림 3-3] st_mode의 비트 구조

- chmod 755 test.out
  - st_mode의 하위 9비트를 111 101 101 로 변경
- chmod -R 644 testdir1
  - testdir1을 포함한 모든 하위 파일/디렉토리를 644로 변경

파일 접근권한 제어

파일 확장자

- Linux/Unix에서는 파일의 확장자는 아무런 의미가 없음
- a.exe b.jpg 등 확장자는 이름의 일부일 뿐임

실행 파일? or not?

- 접근 권한의 x bit으로만 구분됨

./a.out

- 현재 디렉토리 밑의 a.out 파일을 실행하는 의미
- a.out에 x bit 권한이 없으면 실행 안됨

파일 종류 검색[1]

상수를 이용한 파일 종류 검색

- 파일의 종류 검색 관련 상수

상수명	상수값(16진수)	기능
S_IFMT	0xF000	st_mode 값에서 파일의 종류를 정의한 부분을 가져옴
S_IFIFO	0x1000	FIFO 파일
S_IFCHR	0x2000	문자 장치 특수 파일
S_IFDIR	0x4000	디렉토리
S_IFBLK	0x6000	블록 장치 특수 파일
S_IFREG	0x8000	일반 파일
S_IFLNK	0xA000	심볼릭 링크 파일
S_IFSOCK	0xC000	소켓 파일

- st_mode 값과 S_IFMT을 AND(&) 연산하면 파일의 종류 부분만 남음

상수를 이용해 파일 종류 검색하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <stdio.h>
04
05  int main(void) {
06      struct stat buf;
07      int kind;
08
09      stat("unix.txt", &buf);
10
11      printf("Mode = %o (16진수: %x)\n", (unsigned int)buf.st_mode,
(unsigned int)buf.st_mode);
12
13      kind = buf.st_mode & S_IFMT;
14      printf("Kind = %x\n", kind);
15
16      switch (kind) {
17          case S_IFIFO:
18              printf("unix.txt : FIFO\n");
19              break;
20          case S_IFDIR:
21              printf("unix.txt : Directory\n");
22              break;
```

상수를 이용해 파일 종류 검색하기

```
23          case S_IFREG:
24              printf("unix.txt : Regular File\n");
25              break;
26      }
27
28      return 0;
29  }
```

```
# ex3_3.out
Mode = 100644 (16진수: 81a4)
Kind = 8000
unix.txt : Regular File
```

파일 종류 검색[2]

매크로를 이용한 파일 종류 검색

매크로명	매크로 정의	기 능
S_ISFIFO(mode)	(((mode)&0xF000) == 0x1000)	참이면 FIFO 파일
S_ISCHR(mode)	(((mode)&0xF000) == 0x2000)	참이면 문자 장치 특수 파일
S_ISDIR(mode)	(((mode)&0xF000) == 0x4000)	참이면 디렉토리
S_ISBLK(mode)	(((mode)&0xF000) == 0x6000)	참이면 블록 장치 특수 파일
S_ISREG(mode)	(((mode)&0xF000) == 0x8000)	참이면 일반 파일
S_ISLNK(mode)	(((mode)&0xF000) == 0xa000)	참이면 심볼릭 링크 파일
S_ISSOCK(mode)	(((mode)&0xF000) == 0xc000)	참이면 소켓 파일

- 각 매크로는 mode 값을 0xF000과 AND연산 수행
- AND연산의 결과를 파일의 종류별로 정해진 값과 비교

매크로를 이용해 파일 종류 검색하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <stdio.h>
04
05  int main(void) {
06      struct stat buf;
07
08       stat("unix.txt", &buf);
09       printf("Mode  =  %o  (16 진 수 :  %x)\n",(unsigned  int)buf.st_mode,
                  (unsigned int)buf.st_mode);
11
12       if(S_ISFIFO(buf.st_mode)) printf("unix.txt : FIFO\n");
13       if(S_ISDIR(buf.st_mode)) printf("unix.txt : Directory\n");
14       if(S_ISREG(buf.st_mode)) printf("unix.txt : Regular File\n");
15
16      return 0;
17  }
```

```
# ex3_4.out
Mode = 100644 (16진수: 81a4)
unix.txt : Regular File
```

파일 접근 권한 검색[3]

함수를 사용한 파일 접근 권한 검색 : access(2)

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

- path에 지정된 파일이 amode로 지정한 권한을 가졌는지 확인
- 접근권한이 있으면 0을, 오류가 있으면 -1을 리턴
- 오류메시지
  - ENOENT : 파일이 없음
  - EACCESS : 접근권한이 없음
- amode
  - R_OK : 읽기 권한 확인
  - W_OK : 쓰기 권한 확인
  - X_OK : 실행 권한 확인
  - F_OK : 파일이 존재하는지 확인

access 함수를 이용해 접근 권한 검색하기

```
01  #include <sys/errno.h>
02  #include <unistd.h>
03  #include <stdio.h>
04
05  extern int errno;
06
07  int main(void) {
08      int per;
09
10      if (access("unix.bak", F_OK) == -1 && errno == ENOENT)
11          printf("unix.bak: File not exist.\n");
12
13      per = access("unix.txt", R_OK);
14      if (per == 0)
15          printf("unix.txt: Read permission is permitted.\n");
16      else if (per == -1 && errno == EACCES)
17          printf("unix.txt: Read permission is not permitted.\n");
18
19      return 0;
20  }
```

```
# ls -l unix*
-rw-r--r--   1 root other 24  1월   8일   15:47 unix.txt
# ex3_6.out
unix.bak: File not exist.
unix.txt: Read permission is permitted.
```

파일 접근권한 변경

파일명으로 접근권한 변경 : chmod(2)

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

- path에 지정한 파일의 접근권한을 mode값으로 변경
- 접근권한 추가 - OR연산자
  - chmod(path, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
  - mode |= S_IWGRP;
- 접근권한 제거 - NOT연산 후 AND 연산자
  - mode &= ~(S_IROTH);

파일 기술자로 접근 권한 변경 : fchmod(2)

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

파일 접근권한 변경

POSIX에서 정의한 접근권한 관련 상수

상수명	상수값	기능
S_IRWXU	00700	소유자 읽기/쓰기/실행 권한
S_IRUSR	00400	소유자 읽기 권한
S_IWUSR	00200	소유자 쓰기 권한
S_IXUSR	00100	소유자 실행 권한
S_IRWXG	00070	그룹 읽기/쓰기/실행 권한
S_IRGRP	00040	그룹 읽기 권한
S_IWGRP	00020	그룹 쓰기 권한
S_IXGRP	00010	그룹 실행 권한
S_IRWXO	00007	기타 사용자 읽기/쓰기/실행 권한
S_IROTH	00004	기타 사용자 읽기 권한
S_IWOTH	00002	기타 사용자 쓰기 권한
S_IXOTH	00001	기타 사용자 실행 권한

시프트 연산 없이 직접
AND 연산이 가능한 상수 정의

chmod 함수 사용하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <stdio.h>
04
05  int main(void) {
06      struct stat buf;
07
08      chmod("unix.txt", S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
09      stat("unix.txt", &buf);
10      printf("1.Mode = %o\n", (unsigned int)buf.st_mode);
11
12      chmod("unix.txt", 0770);
13      stat("unix.txt", &buf);
14      printf("2.Mode = %o\n", (unsigned int)buf.st_mode);
15
16      return 0;
17  }
```

```
# ls -l unix.txt
-rw-r--r--   1 root    other 24  1월   8일   15:47 unix.txt
# ex3_7.out
1.Mode = 100754
2.Mode = 100770
# ls -l unix.txt
-rwxrwx---   1 root    other 24  1월   8일   15:47 unix.txt
```

0770 ?

링크 파일 생성[1]

링크

- 이미 있는 파일이나 디렉토리에 접근할 수 있는 새로운 이름
- 같은 파일/디렉토리지만 여러 이름으로 접근할 수 있게 한다
- 하드 링크 : 기존 파일과 동일한 inode 사용, inode에 저장된 링크 개수 증가
  - `ln original.txt link.txt`
- 심볼릭 링크 : 기존 파일에 접근하는 다른 파일 생성(다른 inode 사용)
  - `ln -s original.txt link.txt`

하드링크 생성 : link(2)

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

- 두 경로는 같은 파일시스템에 존재해야 함

link 함수 사용하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <unistd.h>
04  #include <stdio.h>
05
06  int main(void) {
07      struct stat buf;
08
09      stat("unix.txt", &buf);
10      printf("Before Link Count = %d\n", (int)buf.st_nlink);
11
12      link("unix.txt", "unix.ln");
13
14      stat("unix.txt", &buf);
15      printf("After Link Count = %d\n", (int)buf.st_nlink);
16
17      return 0;
18  }
```

```
# ls -l unix*
-rwxrwx---   1 root    other  24  1월   8일   15:47 unix.txt
# ex3_8.out
Before Link Count = 1
After Link Count = 2
# ls -l unix*
-rwxrwx---   2 root    other  24  1월   8일   15:47 unix.ln
-rwxrwx---   2 root    other  24  1월   8일   15:47 unix.txt
```

링크 파일 생성[2]

심볼릭 링크 생성 : symlink(2)

```
#include <unistd.h>
int symlink(const char *name1, const char *name2);
```

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <unistd.h>
04
05  int main(void) {
06      symlink("unix.txt", "unix.sym");
07
08      return 0;
09  }
```

```
# ls -l unix*
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.ln
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.txt
# ex3_9.out
# ls -l unix*
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.ln
lrwxrwxrwx   1 root     other          8  1월 11일   18:48 unix.sym ->
unix.txt
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.txt
```

심볼릭 링크 정보 검색

심볼릭 링크 정보 검색 : lstat(2)

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

- lstat : 심볼릭 링크 자체의 파일 정보 검색
- 심볼릭 링크를 stat 함수로 검색하면 원본 파일에 대한 정보를 검색

심볼릭 링크의 내용 읽기 : readlink(2)

```
#include <unistd.h>
ssize_t readlink(const char *restrict path, char  *restrict buf, size_t bufsiz);
```

- 심볼릭 링크의 데이터 블록에 저장된 내용 읽기

원본 파일의 경로 읽기 : realpath(3)

```
#include <stdlib.h>
char *realpath(const char *restrict  file_name, char *restrict resolved_name);
```

- 심볼릭 링크가 가리키는 원본 파일의 실제 경로명 출력

Exercise

Ex.

- 특정 파일/디렉토리 명을 주면 모든 정보를 출력하고,
- 권한을 700으로 변경하는 프로그램을 작성

```
$ a.out /tmp/test
Is file = Y
Is directory = N
Inode = 192
Mode = 100644
Nlink = 1
UID = 0
GID = 1
SIZE = 24
Atime = 1231397228
Mtime = 1231397228
Ctime = 1231397228
Blksize = 8192
Blocks = 2
$ ls -aFl /tmp/test
-rwx------ 1 root root  429 2015-02-12 00:11 /tmp/test
```

디렉토리 관련 함수[1]

디렉토리 생성: mkdir(2)

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

- path에 지정한 디렉토리를 mode 권한에 따라 생성

디렉토리 삭제: rmdir(2)

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

디렉토리명 변경: rename(2)

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

디렉토리 생성/삭제/이름 변경하기

```
01  #include <sys/stat.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      if (mkdir("han", 0755) == -1) {
08          perror("han");
09          exit(1);
10      }
11
12      if (mkdir("bit", 0755) == -1) {
13          perror("bit");
14          exit(1);
15      }
16
17      if (rename("han", "hanbit") == -1) {
18          perror("hanbit");
19          exit(1);
20  }
21
```

han -> hanbit로 변경

디렉토리 생성/삭제/이름 변경하기

```
22      if (rmdir("bit") == -1) {
23          perror("bit");
24          exit(1);
25      }
26
27      return 0;
28  }
```

bit는 생성했다 삭제

```
# ex3_13.out
# ls -l
drwxr-xr-x   2 root other 512  1월 12일   18:06 hanbit
```

디렉토리 관련 함수[2]

현재 작업 디렉토리 위치 : getcwd(3)

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

- size는 buf의 크기
- 현재 작업 디렉토리 위치를 알려주는 명령은 pwd, 함수는 getcwd

디렉토리 이동: chdir(2)

```
#include <unistd.h>
int chdir(const char *path);
```

작업 디렉토리 위치 검색/디렉토리 이동하기

```
01  #include <unistd.h>
02  #include <stdio.h>
03
04  int main(void) {
05      char *cwd;
06      char wd[BUFSIZ];
07
08      cwd = getcwd(NULL, BUFSIZ);
09      printf("1.Current Directory : %s\n", cwd);
10
11      chdir("hanbit");
12
13      getcwd(wd, BUFSIZ);
14      printf("2.Current Directory : %s\n", wd);
15
16      return 0;
17  }
```

```
# ex3_14.out
1.Current Directory : /export/home/jw/syspro/ch3
2.Current Directory : /export/home/jw/syspro/ch3/hanbit
```

디렉토리 정보 검색[1]

디렉토리 열기: opendir(3)

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

```
typedef struct dirent {
    ino_t          d_ino;
    off_t          d_off;
    unsigned short d_reclen;
    char           d_name[NAME_MAX+1];
} dirent_t;
```

- 성공하면 열린 디렉토리를 가리키는 DIR 포인터를 리턴

디렉토리 닫기: closedir(3)

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dirp);
```

디렉토리 정보 읽기: readdir(3)

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

- 디렉토리의 내용을 한 번에 하나씩 읽어옴

디렉토리 열고 정보 읽기

```
01  #include <dirent.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      DIR *dp;
07      struct dirent *dent;
08
09      if ((dp = opendir("hanbit")) == NULL) {
10          perror("opendir: hanbit");
11          exit(1);
12      }
13
14      while ((dent = readdir(dp))) {
15          printf("Name : %s  ", dent->d_name);
16          printf("Inode : %d\n", (int)dent->d_ino);
17      }
18
19      closedir(dp);
20
21      return 0;
22  }
```

```
# ex3_15.out
Name : .  Inode : 208
Name : ..  Inode : 189
```

디렉토리 항목의 상세 정보 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <dirent.h>
04 #include <stdlib.h>
05 #include <stdio.h>
06
07  int main(void) {
08      DIR *dp;
09      struct dirent *dent;
10      struct stat sbuf;
11      char path[BUFSIZ];
12
13      if ((dp = opendir("hanbit")) == NULL) {
14          perror("opendir: hanbit");
15          exit(1);
16      }
18      while ((dent = readdir(dp))) {
19         if (dent->d_name[0] == '.') continue;
20         else break;
21      }
22
```

디렉토리 항목의 상세 정보 검색하기

```
23      sprintf(path, "hanbit/%s", dent->d_name);
24      stat(path, &sbuf);
25
26      printf("Name : %s\n", dent->d_name);
27      printf("Inode(dirent) : %d\n", (int)dent->d_ino);
28      printf("Inode(stat) : %d\n", (int)sbuf.st_ino);
29      printf("Mode : %o\n", (unsigned int)sbuf.st_mode);
30      printf("Uid : %d\n", (int)sbuf.st_uid);
31
32      closedir(dp);
33
34      return 0;
35  }
```

디렉토리의 항목을 읽고
다시 stat 함수로 상세 정보 검색

```
# ls -ai hanbit
208 .   189 ..   213 han.c
# ex3_16.out
Name : han.c
Inode(dirent) : 213
Inode(stat) : 213
Mode : 100644
Uid : 0
```

디렉토리 정보 검색[2]

디렉토리 오프셋: telldir(3), seekdir(3), rewinddir(3)

```
#include <dirent.h>
long telldir(DIR *dirp);
void seekdir(DIR *dirp, long loc);
void rewinddir(DIR *dirp);
```

- telldir : 디렉토리 오프셋의 현재 위치를 알려줌
- seekdir : 디렉토리 오프셋을 loc에 지정한 위치로 이동
- rewinddir : 디렉토리 오프셋을 디렉토의 시작인 0으로 이동

Exercise

Ex.

- 특정 디렉토리 내의 모든 파일 이름과 inode 번호를 출력하는 프로그램을 작성

```
./a.out /tmp/dirtest
25067538 .
25067521 ..
25067540 a.out
25067542 debug
25067543 hello
25067539 hello.c
25067541 perr.c
25067546 winscp437.zip
```

Exercise - Extra

Ex.

- 특정 디렉토리 내의 모든 파일 이름과 inode 번호를 출력하는 프로그램을 작성
- 하위에 디렉토리가 있을 경우 recursive로 출력

```
./a.out /tmp/dirtest
25067538 .
25067521 ..
25067540 a.out
25067542 debug
25067543 dir1
25067539 dir1/hello.c
25067541 dir1/perr.c
25067546 winscp437.zip
```