

리눅스시스템 및 응용

Week 12

File & Concurrency Programming

학습목표

- 파일 정보 검색 시스템호출
 - stat(), fstat(), access()
- 파일 접근 권한 제어
 - chmod()
- 파일 링크 생성, 검색, 제어
 - link(), symlink(), lstat(), readlink(), realpath()
 - unlink()
- 파일 잠금(locking) 및 동시성 제어(Binary Semaphore)

파일 정보 검색

■ 파일명으로 파일 정보 검색 : stat(2)

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

[함수 원형]

```
int stat(const char *pathname, struct stat *statbuf);
```

- pathname : 정보를 알고자 하는 파일명
- statbuf : 검색한 파일 정보를 저장할 구조체 주소
- stat() 함수의 특징
 - pathname에 지정한 파일의 정보를 검색
 - stat() 함수로 파일의 정보를 검색할 때 파일에 대한 읽기/쓰기/실행 권한이 반드시 있어야 하는 것은 아니지만 파일에
이르는 경로의 각 디렉터리에 대한 읽기 권한은 있어야 함
 - 성공하면 0을 반환하고 stat 구조체에 파일 정보를 저장, 오류가 발생하면 -1을 리턴

■ 파일명으로 파일 정보 검색 : stat(2)

■ stat 구조체

- stat() 함수로 검색한 inode 정보는 stat 구조체에 저장되어 리턴
- stat 구조체의 세부 구조는 man -s 2 stat으로 확인

```
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;  
    blksize_t  st_blksize;  
    blkcnt_t   st_blocks;  
  
    struct timespec st_atim;  
    struct timespec st_mtim;  
    struct timespec st_ctim;  
  
    #define st_atime st_atim.tv_sec  
    #define st_mtime st_mtim.tv_sec  
    #define st_ctime st_ctim.tv_sec  
};
```

- **st_dev** : 파일이 저장되어 있는 장치의 번호를 저장
- **st_ino** : 파일의 inode 번호를 저장
- **st_mode** : 파일의 종류와 접근 권한을 저장
- **st_nlink** : 하드 링크의 개수를 저장
- **st_uid** : 파일 소유자의 UID를 저장
- **st_gid** : 파일 소유 그룹의 GID를 저장
- **st_rdev** : 장치 파일이면 주 장치 번호와 부 장치 번호를 저장
장치 파일이 아니면 아무 의미가 없음
- **st_blksize** : 파일 내용을 입출력할 때 사용하는 버퍼의 크기를 저장
- **st_blocks** : 파일에 할당된 파일 시스템의 블록 수로, 블록의 크기는 512바이트

■ 파일명으로 파일 정보 검색 : stat(2)

▪ stat 구조체

– timespec

» 초와 나노초를 저장하기 위한 구조체, 리눅스 커널 2.6부터 나노초 수준의 정밀도를 지원

```
struct timespec {  
    __kernel_time_t tv_sec;          /* 초 (seconds) */  
    long            tv_nsec;        /* 나노초(nanoseconds) */  
};
```

- **st_atime** : 마지막으로 파일을 읽거나 실행한 시각을 timespec 구조체로 저장
- **st_mtime** : 마지막으로 파일의 내용을 변경(쓰기)한 시각을 저장
- **st_ctime** : 마지막으로 inode의 내용을 변경한 시각을 저장
- 리눅스 커널 2.6 이전 버전과의 호환성을 위해 #define으로 타임스탬프 값을 초 단위 값으로 매핑해 정의

```
#define st_atime st_atim.tv_sec  
#define st_mtime st_mtim.tv_sec  
#define st_ctime st_ctim.tv_sec
```

■ 파일명으로 inode 정보 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <unistd.h>
04 #include <stdio.h>
05
06 int main() {
07     struct stat statbuf;
08
09     stat("linux.txt", &statbuf);
10
11     printf("Inode = %d\n", (int)statbuf.st_ino);
12     printf("Mode = %o\n", (unsigned int)statbuf.st_mode);
13     printf("Nlink = %o\n", (unsigned int)statbuf.st_nlink);
14     printf("UID = %d\n", (int)statbuf.st_uid);
15     printf("GID = %d\n", (int)statbuf.st_gid);
16     printf("SIZE = %d\n", (int)statbuf.st_size);
17     printf("Blksize = %d\n", (int)statbuf.st_blksize);
18     printf("Blocks = %d\n", (int)statbuf.st_blocks);
19
20     printf("** timespec Style\n");
21     printf("Atime = %d\n", (int)statbuf.st_atim.tv_sec);
22     printf("Mtime = %d\n", (int)statbuf.st_mtim.tv_sec);
23     printf("Ctime = %d\n", (int)statbuf.st_ctim.tv_sec);
24     printf("** old Style\n");
25     printf("Atime = %d\n", (int)statbuf.st_atime);
26     printf("Mtime = %d\n", (int)statbuf.st_mtime);
27     printf("Ctime = %d\n", (int)statbuf.st_ctime);
28 }
```

실행

```
$ ch3_1.out
Inode = 1048600
Mode = 100644
Nlink = 1
UID = 1000
GID = 1000
SIZE = 219
Blksize = 4096
Blocks = 8
** timespec Style
Atime = 1614502459
Mtime = 1614502459
Ctime = 1614502459
** old Style
Atime = 1614502459
Mtime = 1614502459
Ctime = 1614502459
```

■ 파일명으로 inode 정보 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <unistd.h>
04 #include <stdio.h>
05
06 int main() {
07     struct stat statbuf;
08
09     stat("linux.txt", &statbuf);
10
11     printf("Inode = %d\n", (int)statbuf.st_ino);
12     printf("Mode = %o\n", (unsigned int)statbuf.st_mode);
13     printf("Nlink = %o\n", (unsigned int) statbuf.st_nlink);
14     printf("UID = %d\n", (int)statbuf.st_uid);
15     printf("GID = %d\n", (int)statbuf.st_gid);
16     printf("SIZE = %d\n", (int)statbuf.st_size);
17     printf("Blksize = %d\n", (int)statbuf.st_blksize);
18     printf("Blocks = %d\n", (int)statbuf.st_blocks);
19
20     printf("*** timespec Style\n");
21     printf("Atime = %d\n", (int)statbuf.st_atim.tv_sec);
22     printf("Mtime = %d\n", (int)statbuf.st_mtim.tv_sec);
23     printf("Ctime = %d\n", (int)statbuf.st_ctim.tv_sec);
24     printf("*** old Style\n");
25     printf("Atime = %d\n", (int)statbuf.st_atime);
26     printf("Mtime = %d\n", (int)statbuf.st_mtime);
27     printf("Ctime = %d\n", (int)statbuf.st_ctime);
28 }
```

- **09행** stat() 함수로 linux.txt 파일의 정보를 읽음
- **11~12행** inode 번호는 1048600, st_mode의 값은 100644로 출력
- **13행** 하드링크의 개수는 1이다.
- **14~15행** UID와 GID가 1000으로 출력
UID 1000이 어떤 사용자인지 알고 싶다면 /etc/passwd 파일을 참조
- **16~18행** 파일의 크기는 219바이트, 블록의 크기는 4096바이트
이를 512바이트 블록으로 계산해 출력한 블록 수는 8개
- **21~23행** 타임스탬프 값을 timespec 구조체의 tv_sec 값을 출력
- **25~27행** 타임스탬프 값을 이전 형식의 변수명을 사용해 출력

■ 파일 기술자로 파일 정보 검색 : fstat(2)

```
#include <sys/types.h>                                [함수 원형]
#include <sys/stat.h>
#include <unistd.h>

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

- fd : 열려 있는 파일의 파일 기술자
- statbuf : 검색한 파일 정보를 저장할 구조체 주소
- fstat() 함수의 특징
 - 파일 경로 대신 현재 열려 있는 파일의 파일 기술자를 인자로 받아 파일 정보를 검색한 후 statbuf로 지정한 구조체에 저장

■ 파일 기술자로 파일 정보 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <fcntl.h>
04 #include <unistd.h>
05 #include <stdlib.h>
06 #include <stdio.h>
07
08 int main() {
09     int fd;
10     struct stat statbuf;
11
12     fd = open("linux.txt", O_RDONLY);
13     if (fd == -1) {
14         perror("open: linux.txt");
15         exit(1);
16     }
17
18     fstat(fd, &statbuf);
19
20     printf("Inode = %d\n", (int)statbuf.st_ino);
21     printf("UID = %d\n", (int)statbuf.st_uid);
22     close(fd);
23 }
```

실행

\$ ch3_2.out

Inode = 1048600

UID = 1000

- **12행** fstat() 함수는 파일 기술자를 인자로 받으므로 파일을 열어 파일 기술자를 리턴받음
- **18행** fstat() 함수로 파일 정보를 읽어옴
- **실행 결과** [예제 3-1]과 동일한 파일의 정보를 검색한 것이므로 Inode와 UID의 출력 결과가 동일

파일 접근 권한 제어

■ st_mode의 구조

- st_mode 항목의 데이터형인 mode_t는 unsigned int로 정의되어 있음
- 실제로는 16비트를 사용
- sys/stat.h 파일에 정의된 상수와 매크로는 구조로 저장된 값과 상수를 AND 연산한 값을 추출하는 것

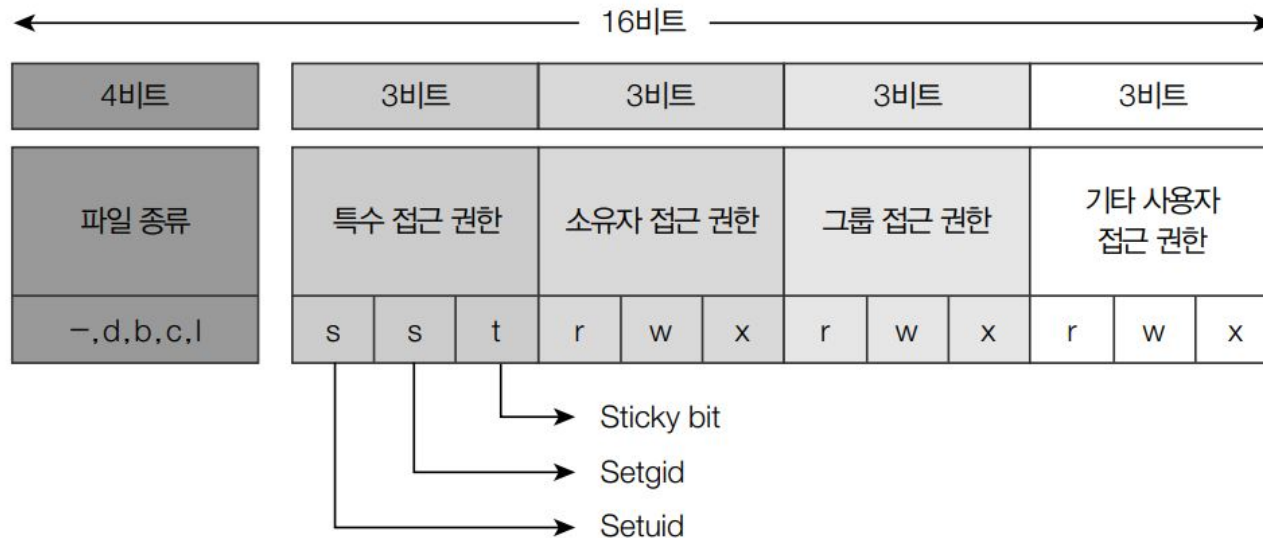


그림 3-1 st_mode의 비트 구조

파일 접근 권한 제어

■ 파일의 종류 검색

- 상수를 이용한 파일 종류 검색

표 3-4 파일의 종류 검색 상수

| 상수명 | 상숫값(8진수) | 기능 |
|----------|----------|---------------------------|
| S_IFMT | 0170000 | 파일의 종류 비트를 가져오기 위한 비트 마스크 |
| S_IFSOCK | 0140000 | 소켓 파일 |
| S_IFLNK | 0120000 | 심벌릭 링크 파일 |
| S_IFREG | 0100000 | 일반 파일 |
| S_IFBLK | 0060000 | 블록 장치 특수 파일 |
| S_IFDIR | 0040000 | 디렉터리 |
| S_IFCHR | 0020000 | 문자 장치 특수 파일 |
| S_FIFO | 0010000 | FIFO 파일 |

파일 접근 권한 제어

■ 상수를 이용해 파일 종류 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <stdio.h>
04
05 int main() {
06     struct stat statbuf;
07     int kind;
08
09     stat("linux.txt", &statbuf);
10
11     printf("Mode = %o\n", (unsigned int)statbuf.st_mode);
12
13     kind = statbuf.st_mode & S_IFMT;
14     printf("Kind = %o\n", kind);
15
16     switch (kind) {
17         case S_IFLNK:
18             printf("linux.txt: Symbolic Link\n");
19             break;
20         case S_IFDIR:
21             printf("linux.txt: Directory\n");
22             break;
23         case S_IFREG:
24             printf("linux.txt: Regular File\n");
25             break;
26     }
27 }
```

실행

```
$ ch3_3.out
Mode = 100644
Kind = 100000
linux.txt: Regular File
```

- **09행** stat() 함수로 파일의 정보를 읽음
- **11행** st_mode의 값을 8진수로 출력, 실행 결과를 보면 100644임을 알 수 있음
- **13~14행** st_mode와 S_IFMT를 AND(&) 연산하고 결과를 출력
실행 결과를 보면 100000임을 알 수 있음
- **16~26행** switch 문으로 AND 연산의 결과와 상숫값을 비교해 파일의 종류를 출력
- **실행 결과** AND 연산의 결과가 100000인데, [표 3-4]를 보면 이 값은 S_IFREG로 일반 파일임을 알 수 있음
실행 결과에서도 24행을 실행해 'Regular File'이라고 출력

■ 파일의 종류 검색

- 매크로를 이용한 파일 종류 검색

표 3-5 파일의 종류 검색과 관련된 매크로

| 매크로명 | 매크로 정의 | 기능 |
|-------------|---------------------------------|-----------------|
| S_ISLNK(m) | $((m) \& S_IFMT) = S_IFLNK)$ | 참이면 심벌릭 링크 파일 |
| S_ISREG(m) | $((m) \& S_IFMT) = S_IFREG)$ | 참이면 일반 파일 |
| S_ISDIR(m) | $((m) \& S_IFMT) = S_IFDIR)$ | 참이면 디렉터리 |
| S_ISCHR(m) | $((m) \& S_IFMT) = S_IFCHR)$ | 참이면 문자 장치 특수 파일 |
| S_ISBLK(m) | $((m) \& S_IFMT) = S_IFBLK)$ | 참이면 블록 장치 특수 파일 |
| S_ISFIFO(m) | $((m) \& S_IFMT) = S_IFIFO)$ | 참이면 FIFO 파일 |
| S_ISSOCK(m) | $((m) \& S_IFMT) = S_IFSOCK)$ | 참이면 소켓 파일 |

파일 접근 권한 제어

■ EX: 매크로를 이용한 파일 종류 검색

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <stdio.h>
04
05 int main() {
06     struct stat statbuf;
07     int kind;
08
09     stat("linux.txt", &statbuf);
10
11     printf("Mode = %o\n", (unsigned int)statbuf.st_mode);
12
13     if(S_ISLNK(statbuf.st_mode))
14         printf("linux.txt : Symbolic Link\n");
15     if(S_ISDIR(statbuf.st_mode))
16         printf("linux.txt : Directory\n");
17     if(S_ISREG(statbuf.st_mode))
18         printf("linux.txt : Regular File\n");
19 }
```

실행

```
$ ch3_4.out
```

```
Mode = 100644
```

```
linux.txt : Regular File
```

- **09행** stat() 함수로 파일의 정보를 읽음
- **13~18행** st_mode를 S_ISLNK, S_ISDIR, S_ISREG 매크로로 확인해 파일의 종류를 검색
- **실행 결과** st_mode의 값이 8진수 100644이므로 S_ISREG(statbuf.st_mode)의 처리 결과는 $100644 \& 0170000 = 100000$ 따라서 일반 파일임을 알 수 있음

파일 접근 권한 제어

■ 파일 접근 권한 검색

■ 상수를 이용한 파일 접근 권한 검색

- st_mode의 값을 왼쪽으로 3비트 이동시키거나 상숫값을 오른쪽으로 3비트 이동시킨 후 AND 연산을 수행하면
그룹의 접근 권한을 알 수 있음

```
st_mode & (S_IREAD >> 3)
```

- POSIX에서는 이와 같이 번거롭게 시프트 연산을 하는 대신 직접 AND 연산이 가능한 다른 상수를 정의

표 3-7 파일의 접근 권한 검색 상수(POSIX)

| 상수명 | 상숫값 | 기능 |
|---------|-------|----------------------|
| S_IRWXU | 00700 | 소유자에게 읽기/쓰기/실행 권한 |
| S_IRUSR | 00400 | 소유자에게 읽기 권한 |
| S_IWUSR | 00200 | 소유자에게 쓰기 권한 |
| S_IXUSR | 00100 | 소유자에게 실행 권한 |
| S_IRWXG | 00070 | 그룹에게 읽기/쓰기/실행 권한 |
| S_IRGRP | 00040 | 그룹에게 읽기 권한 |
| S_IWGRP | 00020 | 그룹에게 쓰기 권한 |
| S_IXGRP | 00010 | 그룹에게 실행 권한 |
| S_IRWXO | 00007 | 기타 사용자에게 읽기/쓰기/실행 권한 |
| S_IROTH | 00004 | 기타 사용자에게 읽기 권한 |
| S_IWOTH | 00002 | 기타 사용자에게 쓰기 권한 |
| S_IXOTH | 00001 | 기타 사용자에게 실행 권한 |

파일 접근 권한 제어

■ 상수를 이용해 파일의 접근 권한 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <stdio.h>
04
05 int main() {
06     struct stat statbuf;
07
08     stat("linux.txt", &statbuf);
09     printf("Mode = %o\n", (unsigned int)statbuf.st_mode);
10
11     if ((statbuf.st_mode & S_IREAD) != 0)
12         printf("linux.txt: User has a read permission\n");
13
14     if ((statbuf.st_mode & (S_IREAD >> 3)) != 0)
15         printf("linux.txt: Group has a read permission\n");
16
17     if ((statbuf.st_mode & S_IROTH) != 0)
18         printf("linux.txt: Other have a read permission\n");
19 }
```

실행

```
$ ch3_5.out
Mode = 100644
linux.txt: User has a read permission
linux.txt: Group has a read permission
linux.txt: Other have a read permission
```

- **08~09행** stat() 함수로 파일의 정보를 검색하고 st_mode 값을 출력
- **11~12행** st_mode의 값을 S_IREAD와 AND 연산해 소유자의 읽기 권한을 확인
- **14~15행** S_IREAD 상숫값을 오른쪽으로 3만큼 이동시켜 그룹의 읽기 권한을 확인
- **17~18행** POSIX가 정의한 상수 S_IROTH를 이용해 기타 사용자의 읽기 권한을 검색

파일 접근 권한 제어

■ 함수를 이용한 접근 권한 검색 : access(2)

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

[함수 원형]

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

- pathname : 접근 권한을 알고자 하는 파일의 경로
- mode : 접근 권한
- access() 함수의 특징
 - 파일의 접근 권한을 검색할 수 있는 시스템 호출
 - pathname에 지정된 파일이 mode로 지정한 권한을 지니고 있는지 확인하고 리턴
 - 유효 사용자 ID가 아닌 실제 사용자 ID에 대한 접근 권한만 확인할 수 있음
 - access() 함수는 접근 권한이 있으면 0을, 오류가 있으면 -1을 리턴
 - » ENOENT : 해당 파일이 존재하지 않거나 심벌릭 링크의 경우 원본 파일이 없음
 - » EACCES : 접근 권한이 없음
 - 두 번째 인자인 mode에 사용할 수 있는 상수
 - » **R_OK** : 읽기 권한 확인
 - » **W_OK** : 쓰기 권한 확인
 - » **X_OK** : 실행 권한 확인
 - » **F_OK** : 파일이 존재하는지 확인

파일 접근 권한 제어

■ 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() {
08     int perm;
09
10     if (access("linux.bak", F_OK) == -1 && errno == ENOENT)
11         printf("linux.bak: File not exist.\n");
12
13     perm = access("linux.txt", R_OK);
14
15     if (perm == 0)
16         printf("linux.txt: Read permission is permmitted.\n");
17     else if (perm == -1 && errno == EACCES)
18         printf("linux.txt: Read permission is not permmitted.\n");
19 }
```

- **10행** access() 함수의 mode에 F_OK를 지정해 linux.bak 파일이 존재하는지 확인
errno 변수에 저장된 메시지가 ENOENT면 파일이 없다는 의미
- **13행** R_OK로 읽기 권한 여부를 검색
- **15행** 읽기 권한 검색 결과가 0이면 읽기 권한이 있다는 의미
- **17행** 읽기 권한 검색 결과가 -1이고 errno에 저장된 오류 메시지가 EACCES면
접근 권한이 없다는 의미
- **실행 결과** linux.bak 파일은 없으므로 오류 메시지가 출력되고, linux.txt 파일에 대한
읽기 권한은 있으므로 권한이 있음을 알리는 메시지가 출력

실행

```
$ ls -l linux.*
-rw-r--r-- 1 jw jw 219  2월 28 17:54 linux.txt
$ ch3_6.out
linux.bak: File not exist.
linux.txt: Read permission is permmitted.
```

파일 접근 권한 제어

■ 파일명으로 접근 권한 변경 : chmod(2)

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

[함수 원형]

```
int chmod(const char *pathname, mode_t mode)
```

- pathname : 접근 권한을 변경하려는 파일의 경로
- mode : 접근 권한 파일의 접근 권한을 검색할 수 있는 시스템 호출
- chmod() 함수의 특징
 - 접근 권한을 변경할 파일의 경로를 받아 mode에 지정한 상숫값으로 권한을 변경
 - 특수 접근 권한을 변경할 때는 S_ISUID, S_ISGID, S_ISVTX를 이용
 - 소유자/그룹/기타 사용자의 접근 권한을 변경할 때는 상수를 이용

파일 접근 권한 제어

■ 파일명으로 접근 권한 변경 : chmod(2)

- 접근 권한 변경 예시

- 기존 접근 권한과 관계없이 접근 권한을 모두 새로 지정하려면 상수를 이용해 권한을 생성한 후 인자로 지정

```
chmod(pathname, S_IRWXU);  
chmod(pathname, S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH);
```

- 기존 접근 권한을 변경해 권한을 조정할 수도 있음

```
mode |= S_IWGRP;
```

- 접근 권한을 제거하려면 제거하려는 권한의 상숫값을 NOT 연산한 후 AND 연산을 실행

```
mode &= ~(S_IROTH);
```

- mode 값을 변경한 후 chmod() 함수를 호출해야 변경된 접근 권한이 적용

```
chmod(pathname, mode);
```

파일 접근 권한 제어

■ 파일명으로 접근 권한 변경하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <stdio.h>
04
05 int main() {
06     struct stat statbuf;
07
08     chmod("linux.txt", S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
09
10     stat("linux.txt", &statbuf);
11     printf("1.Mode = %o\n", (unsigned int)statbuf.st_mode);
12
13     statbuf.st_mode |= S_IWGRP;
14     statbuf.st_mode &= ~(S_IROTH);
15
16     chmod("linux.txt", statbuf.st_mode);
17
18     stat("linux.txt", &statbuf);
19     printf("2.Mode = %o\n", (unsigned int)statbuf.st_mode);
20 }
```

실행

```
$ ls -l linux.txt
-rw-r--r-- 1 jw jw 219  2월 28 17:54 linux.txt
$ ch3_7.out
1.Mode = 100754
2.Mode = 100770
$ ls -l linux.txt
-rwxrwx--- 1 jw jw 219  2월 28 17:54 linux.txt
```

- **08행** 기존 권한에 관계없이 linux.txt 파일의 권한을 변경
소유자는 읽기/쓰기/실행 권한, 그룹은 읽기/실행 권한, 기타 사용자는 읽기 권한(754)으로 변경
- **10~11행** stat() 함수로 파일의 정보를 읽고 st_mode 값을 출력
- **13~14행** 10행에서 읽은 st_mode의 현재 값에 그룹의 쓰기 권한을 추가하고 기타 사용자의 읽기 권한을 제거하도록 설정
- **16행 앞서** 설정한 값으로 chmod() 함수를 실행
- **실행 결과** linux.txt 파일의 접근 권한이 변경되었음을 알 수 있음

■ 파일 기술자로 접근 권한 변경 : fchmod(2)

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

[함수 원형]

```
int fchmod(int fd, mode_t mode);
```

- fd : 열려 있는 파일의 파일 기술자
- mode : 접근 권한
- fchmod() 함수의 특징
 - 접근 권한을 변경할 파일의 파일 기술자를 받아 mode에 미리 정의한 상숫값으로 변경할 권한을 지정
 - 이미 열려 있는 파일의 파일 기술자를 받아 접근 권한을 변경
 - 접근 권한 지정 방법은 chmod() 함수와 같음

■ 링크

- 링크의 정의
 - 이미 있는 파일이나 디렉터리에 접근할 수 있는 새로운 이름을 의미
 - 같은 파일이나 디렉터리지만 여러 이름으로 접근할 수 있게 하는 것
 - 심벌릭 링크와 하드 링크로 나뉨
 - 링크 기능을 이용하면 사용자 편의성을 높일 수 있음
 - » ex) 이전 시스템과의 호환성을 유지, 경로가 복잡한 파일에 간단한 경로를 제공
- 하드 링크
 - 파일에 접근할 수 있는 파일명을 새로 생성하는 것으로, 기존 파일과 동일한 inode를 사용
 - 하드 링크를 생성하면 inode에 저장된 링크 개수가 증가
- 심벌릭 링크
 - 기존 파일에 접근할 수 있는 다른 파일을 만듦
 - 기존 파일과 다른 inode를 사용하며, 기존 파일의 경로를 저장

■ 하드 링크 생성 : link(2)

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

[함수 원형]

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

- oldpath : 기존 파일의 경로
- newpath : 새로 생성할 링크의 경로
- link() 함수의 특징
 - 하드 링크를 생성할 때 사용
 - 기존 파일의 경로를 첫 번째 인자인 oldpath로 , 새로 생성할 하드 링크 파일의 경로를 두 번째 인자인 newpath로 받음
 - 하드 링크는 같은 파일 시스템에 있어야 하므로 두 경로를 반드시 같은 파일 시스템으로 지정해야 함
 - 수행에 성공하면 0을, 실패하면 -1을 리턴

■ link() 함수로 하드 링크 생성하기

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

실행

```
$ ls -l linux.*
-rwxrwx--- 1 jw jw 219  2월 28 17:54 linux.txt
$ ch3_8.out
Before Link Count = 1
After Link Count = 2
$ ls -l linux.*
-rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.ln
-rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.txt
$ ls -li linux.*
1048600 linux.ln 1048600 linux.txt
```

- **09~10행** 하드 링크를 생성하기 이전의 링크 개수를 확인
- **12행** linux.txt 파일의 하드 링크로 linux.ln을 생성
- **14~15행** 하드 링크를 생성한 이후의 링크 개수를 확인
- **실행 결과** linux.ln 파일이 생성되었고, 링크 개수가 변경
linux.txt와 linux.ln 파일의 inode가 1048600으로 동일하다는 것도 알 수 있음

■ 심벌릭 링크 생성 : symlink(2)

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

[함수 원형]

```
int symlink(const char *target, const char *linkpath);
```

- target : 기존 파일의 경로
- linkpath : 새로 생성할 심벌릭 링크의 경로
- symlink() 함수의 특징
 - 기존 파일의 경로를 첫 번째 인자인 target으로 받고 새로 생성할 심벌릭 링크의 경로를 두 번째 인자인 linkpath로 받음
 - 심벌릭 링크는 기존 파일과 다른 파일 시스템에도 생성할 수 있음
 - symlink() 함수는 수행에 성공하면 0을, 실패하면 -1을 리턴

링크 파일 생성

■ symlink() 함수로 심벌릭 링크 생성하기

```
01 #include <unistd.h>
02
03 int main() {
04     symlink("linux.txt", "linux.sym");
05 }
```

실행

```
$ ls -l linux,*
-rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.ln
-rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.txt
$ cd 3_9.out
$ ls -l linux,*
-rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.ln
lrwxrwxrwx 1 jw jw  9  3월  4 20:12 linux.sym -> linux.txt
-rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.txt
```

- **04행** linux.txt 파일의 심벌릭 링크인 linux.sym을 생성
- **실행 결과** linux.sym 파일이 생성

심벌릭 링크를 생성했지만 기존 파일에는 아무 변화가 없음을 알 수 있음

■ 심벌릭 링크의 정보 검색 : lstat(2)

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

[함수 원형]

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

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

```
int lstat(const char *pathname, struct stat *statbuf);
```

- pathname : 심벌릭 링크의 경로
- statbuf : 새로 생성할 링크의 경로
- lstat() 함수의 특징
 - lstat() 함수는 심벌릭 링크 자체의 파일 정보를 검색
 - 심벌릭 링크를 stat() 함수로 검색 하면 원본 파일에 대한 정보가 검색된다는 점에 주의
 - 첫 번째 인자로 심벌릭 링크가 온다는 것을 제외하면 stat() 함수와 동일한 방식으로 사용

■ 심벌릭 링크의 정보 검색하기

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <unistd.h>
04 #include <stdio.h>
05
06 int main() {
07     struct stat statbuf;
08
09     printf("1. stat : linux.txt ---\n");
10     stat("linux.txt", &statbuf);
11     printf("linux.txt : Link Count = %d\n", (int)statbuf.st_nlink);
12     printf("linux.txt : Inode = %d\n", (int)statbuf.st_ino);
13
14     printf("2. stat : linux.sym ---\n");
15     stat("linux.sym", &statbuf);
16     printf("linux.sym : Link Count = %d\n", (int)statbuf.st_nlink);
17     printf("linux.sym : Inode = %d\n", (int)statbuf.st_ino);
18
19     printf("3. lstat : linux.sym ---\n");
20     lstat("linux.sym", &statbuf);
21     printf("linux.sym : Link Count = %d\n", (int)statbuf.st_nlink);
22     printf("linux.sym : Inode = %d\n", (int)statbuf.st_ino);
23 }
```

실행

```
$ ls -li linux,*
1048600 -rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.ln
1048710 lrwxrwxrwx 1 jw jw  9  3월  4 20:12 linux.sym -> linux.txt
1048600 -rwxrwx--- 2 jw jw 219  2월 28 17:54 linux.txt

$ ch3_10.out
1. stat : linux.txt ---
linux.txt : Link Count = 2
linux.txt : Inode = 1048600

2. stat : linux.sym ---
linux.sym : Link Count = 2
linux.sym : Inode = 1048600

3. lstat : linux.sym ---
linux.sym : Link Count = 1
linux.sym : Inode = 1048710
```

- **10~12행** 기존 linux.txt 파일의 정보를 stat() 함수로 검색해 링크 개수와 inode 번호를 출력
- **15~17행** stat() 함수를 사용해 심벌릭 링크 파일인 linux.sym의 정보를 검색하고 링크 개수와 inode 번호를 출력
- **20~22행** lstat() 함수를 사용해 심벌릭 링크 파일인 linux.sym의 정보를 검색하고 링크 개수와 inode 번호를 출력
- **실행 결과** 10행과 15행에서 검색한 정보가 동일함 즉, 심벌릭 링크의 정보를 stat() 함수로 검색하면 원본 파일의 정보가 검색되는 것을 알 수 있음
20행에서 lstat() 함수로 검색한 심벌릭 링크의 정보는 링크 파일 자체에 대한 정보로, 15행과 다른 값이 출력

■ 심벌릭 링크의 내용 읽기 : readlink(2)

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

[함수 원형]

```
ssize_t readlink(const char *pathname, char *buf, size_t bufsiz);
```

- pathname : 심벌릭 링크의 경로
- buf : 읽어온 내용을 저장할 버퍼
- bufsiz : 버퍼의 크기
- readlink() 함수의 특징
 - 심벌릭 링크의 경로를 받아 해당 파일의 내용을 읽음
 - 호출시 읽은 내용을 저장할 버퍼와 해당 버퍼의 크기를 인자로 지정
 - 수행에 성공 하면 읽어온 데이터의 크기(바이트 수)를, 실패하면 -1을 리턴

링크 파일 생성

■ readlink() 함수로 심벌릭 링크의 내용 읽기

```
01 #include <sys/stat.h>
02 #include <unistd.h>
03 #include <stdlib.h>
04 #include <stdio.h>
05
06 int main() {
07     char buf[BUFSIZ];
08     int n;
09
10     n = readlink("linux.sym", buf, BUFSIZ);
11     if (n == -1) {
12         perror("readlink");
13         exit(1);
14     }
15
16     buf[n] = '\0';
17     printf("linux.sym : READLINK = %s\n", buf);
18 }
```

실행

```
$ ch3_11.out
linux.sym : READLINK = linux.txt
$ ls -l linux.sym
lrwxrwxrwx 1 jw jw 9 3월 4 20:12 linux.sym -> linux.txt
```

- **10행** linux.sym 파일의 내용을 readlink() 함수로 읽어 buf에 저장
- **16~17행** buf의 끝에 널을 추가하고 저장된 내용을 출력한
- **실행 결과** 심벌릭 링크를 ls -l 명령으로 확인했을 때 '->' 다음에 오는 원본 파일의 경로가 심벌릭 링크의 데이터 블록에 저장된 내용
심벌릭 링크의 크기는 'linux.txt'의 바이트 수인 9임을 알 수 있음

■ 심벌릭 링크 원본 파일의 경로 읽기 : realpath(3)

```
#include <limits.h>
```

[함수 원형]

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

```
char *realpath(const char *path, char *resolved_path);
```

- path : 심벌릭 링크의 경로명
- resolved_path : 경로명을 저장할 버퍼 주소
- realpath() 함수의 특징
 - 심벌릭 링크명을 받아 실제 경로명을 resolved_name에 저장
 - 성공하면 실제 경로명이 저장된 곳의 주소를, 실패하면 널 포인터를 리턴

■ realpath() 함수로 원본 파일의 경로 읽기

```
01 #include <sys/stat.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     char buf[BUFSIZ];
07
08     realpath("linux.sym", buf);
09     printf("linux.sym : REALPATH = %s\n", buf);
10 }
```

실행

```
$ ch3_12.out
linux.sym : REALPATH = /home/jw/src/ch3/linux.txt
```

- **08~09행** realpath() 함수를 사용해 심벌릭 링크인 linux.sym의 원본 파일 경로를 얻고 출력

■ 링크 끊기 : unlink(2)

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

[함수 원형]

```
int unlink(const char *pathname);
```

- pathname : 삭제할 링크의 경로
- unlink() 함수의 특징
 - unlink() 함수에서 연결을 끊은 경로명이 그 파일을 참조하는 마지막 링크라면 파일은 삭제
 - 만약 인자로 지정한 경로명이 심벌릭 링크이면 링크가 가리키는 원본 파일이 아니라 심벌릭 링크 파일이 삭제

■ realpath() 함수로 원본 파일의 경로 읽기

```
01 #include <sys/stat.h>
02 #include <unistd.h>
03 #include <stdio.h>
04
05 int main() {
06     struct stat statbuf;
07
08     stat("linux.ln", &statbuf);
09     printf("1.linux.ln : Link Count = %d\n", (int)statbuf.st_nlink);
10
11     unlink("linux.ln");
12
13     stat("linux.txt", &statbuf);
14     printf("2.linux.txt: Link Count = %d\n", (int)statbuf.st_nlink);
15
16     unlink("linux.sym");
17 }
```

실행

```
$ ls -l linux.*
-rwxrwx--- 2 jw jw 219 2월 28 17:54 linux.ln
lrwxrwxrwx 1 jw jw  9 3월  4 20:12 linux.sym -> linux.txt
-rwxrwx--- 2 jw jw 219 2월 28 17:54 linux.txt
$ ch3_13.out
1.linux.ln: Link Count = 2
2.linux.txt: Link Count = 1
$ ls -l linux.*
-rwxrwx--- 1 jw jw 219 2월 28 17:54 linux.txt
```

- **08~09행** stat() 함수로 linux.ln 파일의 정보를 검색해 링크 개수를 출력
- **11행** unlink() 함수를 사용해 하드 링크 파일인 linux.ln의 링크를 끊어 삭제
- **13~14행** stat() 함수로 원본 파일인 linux.txt 파일의 정보를 검색해 링크 개수를 출력
- **16행** unlink() 함수를 사용해 심벌릭 링크 파일인 linux.sym의 링크를 끊어 삭제
- **실행 결과** 09행에서 검색한 하드 링크 파일의 링크 개수는 2
11행에서 unlink() 함수로 하드 링크를 삭제한 뒤 원본 파일의 링크 개수를 확인한 결과 1로 변경됨
심벌릭 링크 파일이 삭제된 것을 알 수 있음

Question?