Linux System Programming #3-2

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

chmod(), fchmod()

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

int chmod(const char *pathname, mode_t mode);
int fchmod(int filedes, mode_t mode);
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- **기존 파일의 접근 권한을 변경하기 위해 사용되는 시스템호출**
- **chmod()로 파일의 접근 권한을 변경하려면**
  - 프로세스의 유효 사용자 ID = 파일의 사용자 ID
  - 루트 권한으로 실행
- **파일 접근 권한뿐만 아니라 set-user-ID 비트와 스티키 비트 변경 가능**
- **chmod()의 첫번째 인자 파일의 경로이름**
- **fchmod()의 첫번째 이미 오픈되어 있는 파일 디스크립터를**
- **mode 인자로 사용되는 상수들은 umask()에서 사용하는 cmask 인자와 동일**

파일 소유자 ID , 프로세스 사용자 ID

- **실제 사용자 ID (Real User ID, RUID)**
- **유효 사용자 ID (Effective User ID, EUID)**
- **저장된 사용자 ID (Saved User iD, SUID)**
- **파일 소유권 관련 ID = 파일소유자 ID = owner**

(참고) setuid()

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

int setuid (uid_t uid); //현재 프로세스의 EUID 설정
int setgid (gid_t gid); //현재 프로세스 그룹의 EUID 설정
리턴값 : 성공 시 – 0, 실패 시 –1 : errono에 다음 값 중 하나 설정
 (1) EAGAIN- UID가 실제 사용자 ID와 다르며, 실제 사용자 ID를 UID로 설정하는 것이 소유 가능한 프로세스의
개수(NPROC rlimit)가 넘은 경우
 (2)EPERM – 사용자가 root가 아니며, UID 가 유효사용자 ID 가 아닌 경우
```

- **프로세스의 현재 EUID가 root(0) 이면, RUID와 SUID가 함께 설정**
- **특권 사용자는 UID를 어떤 값으로도 설정 가능**
- **비특권 사용자는 UID로 실제 사용자 ID와 저장된 사용자 ID만 사용할 수 있음**
  - root가 아닌 사용자는 EUID를 RUID 또는 SUID로 설정 가능

chmod() 예제 1

<ssu_chmod_1.c>

```
#include <stdio.h>
#include <stdlib.h>#include <unistd.h>
#include <sys/stat.h>

int main(void)
{
   struct stat statbuf;
   char *fname1 = "ssu_file1";
   char *fname2 = "ssu_file2";

   if (stat(fname1, &statbuf) < 0)
      fprintf(stderr, "stat error %s\n", fname1);

   if (chmod(fname1, (statbuf.st_mode & ~S_IXGRP) | S_ISUID) <
0)
      fprintf(stderr, "chmod error %s\n", fname1);

   if (chmod(fname2, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH |
S_IXOTH) < 0)
      fprintf(stderr, "chmod error %s\n", fname2);

   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ls -l ssu_file1 ssu_file2
-rw-rw-rw- 1 root root 0 Jan  8 22:09 ssu_file1
-rw------- 1 root root 0 Jan  8 22:09 ssu_file2
root@localhost:/home/oslab# ./ssu_chmod_1
root@localhost:/home/oslab# ls -l ssu_file1 ssu_file2
-rwSrw-rw- 1 root root 0 Jan  8 22:09 ssu_file1
-rw-r--r-x 1 root root 0 Jan  8 22:09 ssu_file2
```

chmod() 예제 2

<ssu_chmod_2.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#define MODE_EXEC (S_IXUSR|S_IXGRP|S_IXOTH)

int main(int argc, char *argv[])
{
   struct stat statbuf;
   int i;
   if (argc < 2) {
      fprintf(stderr, "usage: %s <file1> <file2> ... <fileN>\n",
argv[0]);
      exit(1);
   }
   for (i = 1; i < argc; i++) {
      if (stat(argv[i], &statbuf) < 0) {
         fprintf(stderr, "%s : stat error\n", argv[i]);
         continue;
      }
      statbuf.st_mode |= MODE_EXEC;
      statbuf.st_mode ^= (S_IXGRP|S_IXOTH);

      if (chmod(argv[i], statbuf.st_mode) < 0)
         fprintf(stderr, "%s : chmod error\n", argv[i]);
      else
         printf("%s : file permission was changed.\n", argv[i]);
   }
   exit(0);
}
```

실행 결과 //일반사용자권한으로 실행해야 함(루트 권한 X)]

```
oslab@localhost:~$ mkdir ssu_test_dir
oslab@localhost:~$ ./ssu_chmod_2 /bin/su /home/oslab/ssu_test_dir
/bin/su : chmod error
/home/oslab/ssu_test_dir : file permission was changed.
```

비트연산자를 활용한 플래그 비트 조작 및 검사

- flag |= mask // 비트 or 연산 후 할당 => 플래그 특정 비트 켬
- flag &= mask // 비트 and 연산 후 할당 => 플래그 특정 비트 끔
- flag ^= mask // 비트 xor 연산 후 할당 => 플래그 특정 비트 토글
- flag & mask // 비트 and 연산 => 플래그 특정 비트 켜 있는지 검사

chown(), fchown(), lchown()

```
#include <unistd.h>

int chown(const char *pathname, uid_t owner, gid_t group);
int fchown(int filedes, uid_t owner, gid_t group);
int lchown(const char *pathname, uid_t owner, gid_t group);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno 설정

- **파일의 소유자 ID와 그룹 사용자 ID를 변경하기 위한 시스템호출 함수**
- **적절한 권한이 있는 사용자만 호출 가능**
- **chown(), fchown() : 첫 번째 인자로 지정한 파일이 심볼릭 링크이면 심볼릭 링크가 가리키는 파일의 소유자를 변경**
- **lchown() : 심볼릭 링크 파일 자체의 소유자 변경**
- **owner 인자나 group 인자가 –1이면 해당 ID는 변경되지 않음**

chown() 예제

<ssu_chown.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
int main(void)
{
   struct stat statbuf;
   char *fname = "ssu_myfile";
   int fd;

   if ((fd = open(fname, O_RDWR | O_CREAT, 0600)) < 0) {
      fprintf(stderr, "open error for %s\n", fname);
      exit(1);
   }

   close(fd);
   stat(fname, &statbuf);
   printf("# 1st stat call # UID:%d    GID:%d\n", statbuf.st_uid,
statbuf.st_gid);
    if (chown(fname, 501, 300) < 0) {
       fprintf(stderr, "chown error for %s\n", fname);
       exit(1);
    }

   stat(fname, &statbuf);
   printf("# 2nd stat call # UID:%d    GID:%d\n", statbuf.st_uid, statbuf.st_gid);

   if (unlink(fname) < 0) {
      fprintf(stderr, "unlink error for %s\n", fname);
      exit(1);
   }
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_chown
# 1st stat call # UID:0    GID:0
# 2nd stat call # UID:501    GID:300
```

truncate(), ftruncate()

```
#include <unistd.h>
#include <sys/types.h>
int truncate(const char *pathname, off_t length);
int ftruncate(int filedes, off_t length);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **파일을 지정된 길이(length)로 자르는 시스템호출**
- **파일의 원래 크기가 length보다**
  - 크다면 length 이후의 데이터는 잃어버림
  - 작다면 시스템마다 다를 수 있으나, 대부분의 시스템은 파일의 크기가 늘어남 => 원래의 EOF에서 늘어난 크기까지 0으로 채워짐

link(), unlink()

```
#include <unistd.h>
int link(const char *existingpath, const char *newpath);
int unlink(const char *pathname);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **link() : 기존 파일에 대한 링크를 생성하는 시스템호출**
  - 이미 존재하는 파일의 새로운 링크 생성 시
  - inode 내 link count(연결계수). 동일한 파일에 대한 다른 이름으로 접근 => 동일한 inode => existingpath와 newpath 가 동일한 inode =>하나의 파일의 내용을 변경/수정 하면 다른 파일은?
  - 주로 일반 파일에만 사용. 디렉토리에 대한 link() 호출은 루트 권한 필요 => 잘못된 연결에 의해 무한 루프 방지
  - 이종(다른) 파일 시스템의 파일 링크는 불가능 => existingpath와 newpath 모두 동일 파일 시스템 내
  - newpath가 존재하면 에러 리턴
- **unlink() : 기존 디렉토리 항목(파일)의 링크를 제거하는 시스템호출**
  - 파일 삭제 ?-> 해당 inode free -> 디스크 블럭 free
  - root 권한 필요 -> link counter 감소

(참고) 하드링크 vs 심볼릭링크

- **하드 링크**
  - %ln ./a.c ./b.c
- **심볼릭 링크**
  - %ln –s ./aa.c ./bb.c

link() 예제

<ssu_oslab.c>

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

int main(void)
{
   printf("This is oslab file\n");
   exit(0);
}
```

<ssu_link.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
   if (argc < 3) {
      fprintf(stderr, "usage: %s <file1> <file2>\n", argv[0]);       exit(1);
   }
   if (link(argv[1], argv[2]) == -1) {
      fprintf(stderr, "link error for %s\n", argv[1]);
      exit(1);
   }
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# gcc -o oslab ssu_oslab.c
root@localhost:/home/oslab# ./ssu_link oslab eoslab
root@localhost:/home/oslab# ls -l oslab eoslab
-rwxr-xr-x 2 root root 8656 Jan  8 23:16 eoslab
-rwxr-xr-x 2 root root 8656 Jan  8 23:16 oslab
root@localhost:/home/oslab# ./oslab
This is oslab file
root@localhost:/home/oslab# ./eoslab
This is oslab file
```

unlink() 예제 1

<ssu_dump.txt>
Linux System Programming!

<ssu_unlink_1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(void)
{
   char *fname = "ssu_dump.txt";
   if (open(fname, O_RDWR) < 0) {
      fprintf(stderr, "open error for %s\n", fname);
      exit(1);
   }
   if (unlink(fname) < 0) {
      fprintf(stderr, "unlink error for %s\n", fname);
      exit(1);
   }
   printf("File unlinked\n");
   sleep(20);
   printf("Done\n");
   exit(0);
}
```

실행 결과 [Ubuntu, Fedora] // [일반 사용자 권한으로 실행해야 함]

```
oslab@localhost:~$ ls -l ssu_dump.txt
-rw-rw-r-- 1 oslab oslab 26 Jan  9 23:06 ssu_dump.txt
oslab@localhost:~$ df /home/
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1      16381864 4253648  11273024  28% /
oslab@localhost:~$ ./ssu_unlink_1 &
[1] 3970
oslab@localhost:~$ File unlinked
oslab@localhost:~$ ls -l ssu_dump.txt
ls: cannot access 'ssu_dump.txt': No such file or directory
oslab@localhost:~$ df /home/
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1      16381864 4253648  11273024  28% /
oslab@localhost:~$ Done
[1]+  Done                    ./ssu_unlink
oslab@localhost:~$ df /home/
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1      16381864 4253644  11273028  28% /
```

unlink() 예제 2

<ssu_unlink_2.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(void)
{
   char buf[64];
   char *fname = "ssu_tempfile";
   int fd;
   int length;
   if ((fd = open(fname, O_RDWR | O_CREAT | O_TRUNC, 0600))
< 0) {
      fprintf(stderr, "first open error for %s\n", fname);
      exit(1);
   }
   if (unlink(fname) < 0) {
      fprintf(stderr, "unlink error for %s\n", fname);
      exit(1);
   }
   if (write(fd, "How are you?", 12) != 12) {
      fprintf(stderr, "write error\n");
      exit(1);
   }
   lseek(fd, 0, 0);
   if ((length = read(fd, buf, sizeof(buf))) < 0) {
      fprintf(stderr, "buf read error\n");
      exit(1);
   }
   else
      buf[length] = 0;

   printf("%s\n", buf);

   close(fd);

   if ((fd = open(fname, O_RDWR)) < 0) {
      fprintf(stderr, "second open error for %s\n", fname);
      exit(1);
   }

   else
      close(fd);

   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_unlink_2
How are you?
second open error for ssu_tempfile
```

remove(3), rename(2)

```
#include <stdio.h>
int remove(const char *pathname);
int rename(const char *oldname, const char *newname);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **remove() : 파일이나 디렉토리를 unlink하는 시스템호출**
  - unlink()/rmdir()과 동일한 기능
  - 차이점, unlink()는 root 권한이 있어야 디렉토리를 인자로 받을 수 있음. remove() 는 파일/디렉토리 모두 가능
- **rename() : 파일 이름을 변경할 때 호출하는 시스템호출**
  - 변경 대상이 디렉토리가 아니고 파일인 경우 두 번째 인자로는 이미 존재하는 파일 이름은 불가 => 이미 존재하는 파일을 인자로 받으려면 기존 파일을 먼저 삭제하고 이름 변경
  - 변경 대상이 디렉토리이고 두 번째 인자로 지정한 이름이 이미 존재하는 디렉토리의 이름인 경우, 이미 존재하는 디렉토리가 빈 디렉토리라면 이름이 변경되고 그렇지 않은 경우에는 호출 실패
  - 변경 대상 파일이 심볼릭 링크인 경우 심볼릭 링크 파일 자체의 이름이 변경되고 두 인자가 같을 경우 아무것도 변경하지 않고 성공적으로 리턴

remove() 예제

<ssu_remove.c>

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

int main(int argc, char *argv[]) {
   if (argc != 3) {
      fprintf(stderr, "usage: %s <oldname> <newname>\n",
argv[0]);
      exit(1);
   }

   if (link(argv[1], argv[2]) < 0) {
      fprintf(stderr, "link error\n");
      exit(1);
   }
   else
      printf("step1 passed.\n");

   if (remove(argv[1]) < 0) {
      fprintf(stderr, "remove error\n");
      remove(argv[2]);
      exit(1);
   }
   else
      printf("step2 passed.\n");

   printf("Success!\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# vi ssu_abcd.txt
root@localhost:/home/oslab# ls -l ssu_abcd.txt
-rw-r--r-- 1 root root 0 Jan  8 23:34 ssu_abcd.txt
root@localhost:/home/oslab# ./ssu_remove ssu_abcd.txt ssu_efgh.txt
step1 passed.
step2 passed.
Success!
root@localhost:/home/oslab# ls ssu_abcd.txt
ls: cannot access 'ssu_abcd.txt': No such file or directory
root@localhost:/home/oslab# ls -l ssu_efgh.txt
-rw-r--r-- 1 root root 0 Jan  8 23:34 ssu_efgh.txt
```

rename() 예제

<ssu_rename.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
   int fd;
   if (argc != 3) {
      fprintf(stderr, "usage: %s <oldname> <newname>\n",
argv[0]);
      exit(1);
   }
   if ((fd = open(argv[1], O_RDONLY)) < 0) {
      fprintf(stderr, "first open error for %s\n", argv[1]);
      exit(1);
   }
   else
      close(fd);

   if (rename(argv[1], argv[2]) < 0) {
      fprintf(stderr, "rename error\n");
      exit(1);
   }
    if ((fd = open(argv[1], O_RDONLY)) < 0)
      printf("second open error for %s\n", argv[1]);
    else {
      fprintf(stderr, "it's very strange!\n");
      exit(1);
   }

   if ((fd = open(argv[2], O_RDONLY)) < 0) {
      fprintf(stderr, "third open error for %s\n", argv[2]);
      exit(1);
   }

   printf("Everything is good!\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# vi ssu_test1.txt
root@localhost:/home/oslab# ./ssu_rename ssu_test1.txt ssu_test2.txt
second open error for ssu_test1.txt
Everything is good!
root@localhost:/home/oslab# ls ssu_test2.txt
ssu_test2.txt
```

symlink(), readlink()

```
#include <unistd.h>
int symlink(const char *actualpath, const char *sympath);
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
ssize_t readlink(const char *pathname, char *buf, size_t bufsize);
리턴 값: 성공 시 읽은 바이트 수, 에러 시 –1을 리턴하고 errno가 설정됨
```

- **symlink() : 심볼릭 링크를 생성하는 시스템호출**
  - link()를 통해 만들어진 하드 링크의 한계들을 해결하기 위해 만들어진 것
  - 디스크상의 inode를 가리키는 파일이 아닌 파일 자체를 가리키는 파일
  - Windows 운영체제의 "바로가기" 파일과 동일한
  - 심볼릭 링크는 서로 다른 파일 시스템의 파일과 링크 가능
  - actualpath : 실제 파일의 경로 이름, 실제 존재하지 않아도 됨. -> 대상 파일의 내용은 무관
  - Sympath : 심볼릭 링크의 이름

- **readlink() : 심볼릭 링크를 읽는 시스템호출**
  - 링크로 연결된 원래 파일 이름을 읽음
  - buf 인자에 저장된 문자열의 끝이 NULL로 끝나지 않음

symlink() 예제

< ssu_symlink.c>

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

int main(int argc, char *argv[]) {
   if (argc != 3) {
      fprintf(stderr, "usage: %s <actualname> <symname>\n",
argv[0]);
      exit(1);
   }

   if (symlink(argv[1], argv[2]) < 0) {
      fprintf(stderr, "symlink error\n");
      exit(1);
   }
   else
      printf("symlink: %s -> %s\n", argv[2], argv[1]);
   exit(0);

}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_symlink /home/oslab/oslab /bin/oslab
symlink: /bin/oslab -> /home/oslab/oslab
root@localhost:/home/oslab# oslab
This is oslab file
```