실습

- 입력받은 파일의 접근 권한을 출력하고, 접근 권한을 변경하는 프로그램을 작성하라. 문자 모드 기능을 구현한다.

```
$./test g+x unix.txt
```

실습 문제들은 시험에 나올 수 있으므로 잘 이해하고 알아둬야 함.

```
switch(argv[1][0]) {
   case  'g' :
     switch (argv[1][2]) {
        case  'x' : x=S_IXGRP; break;
    ......
 if(argv[1][1]==  '+'  ) buf.st_mode |=x;
 chmod…
```

링크 파일 생성[1]

링크

링크 두 종류가 있음.
하드링크와 심볼릭링크(소프트링크)

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

하드링크 생성 : link(2)

하드링크는 원본파일을 삭제해도 링크가 남아있다면, inode 와 데이터는 살아있음.

파일시스템마다 이 링크를 지원하는 함수가 서로 다르게 구현이 될 수가 있음. 만약, 다른 파일시스템에 링크가 생성이 된다면, 또다른 의미로 지원이 될 수가 있음.

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

링크 카운터

- 두 경로는 같은 파일시스템에 존재해야 함

어떤 파일을 만들게 되면 반드시 이 파일에 대한 정보를 갖는 inode 가 붙게 됨.
하나의 파일을 여러 명이 공유한다면, 여러 명이 같은 카피 파일을 각각 갖는 것이 아닌 디스크에 있는 파일을 여러 명이 링크를 통해서 사용할 수 있음.
하드 링크의 경우에는 여러 학생이 하나의 파일에 대해 공유한다면 inode 를 공유하고, (사실 파일명을 공유한다기보다는 inode 를 공유) 링크 카운터가 공유하는 수만큼 증가하게 됨. (명령어 : ln 파일이름 사용자이름, 프로그램 내 함수 : link())
inode 넘버가 모두 같음. 하나의 공유자가 link 가 끊기더라도 파일의 데이터는 살아있음. link 카운터는 1 줄어듬.
두번째 방법은 심볼릭링크. (명령어 : ln -s 파일이름, 사용자이름, 함수 : symlink())
심볼릭링크로 공유하게 되면, 파일명을 통해 공유하게 됨. symlink() 는 파일을 삭제하게 되면 다른 공유자들도 삭제됨.

inode 넘버

```
[s16010980@sce 0927]$ cp unix.txt unix.cpy
[s16010980@sce 0927]$ ln unix.txt unix.ln1
[s16010980@sce 0927]$ ln unix.txt unix.ln2
[s16010980@sce 0927]$ ls -ali unix*
106301758 -rw-r--r--. 1 s16010980 class 238
2021-09-27 21:42 unix.cpy
106301754 -rw-r--r--. 3 s16010980 class 238
2021-09-27 21:24 unix.ln1
106301754 -rw-r--r--. 3 s16010980 class 238
2021-09-27 21:24 unix.ln2
106301754 -rw-r--r--. 3 s16010980 class 238
2021-09-27 21:24 unix.txt
```

uxin.txt 와 unix.cpy 는 다른 파일이기 때문에 링크가 다름.

[예제 3-8] link 함수 사용하기 (test1.c)

```
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)

원본 파일경로를 공유하는 링크

명령어 : ln -s 파일이름 링크파일(사용자이름)

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

inode 넘버도 파일도 다르지만, 원본 파일에 대한 경로를 가짐.

원본 파일의 경로만을 공유하는 것이기 때문에 원본 파일을 지우게 되면 링크파일이 깨지게 됨.

[예제 3-9] symlink 함수 사용하기 (test2.c)

```
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  }
```

```
106301759 lrwxrwxrwx. 1 s16010980 class   8 2021-09-27 21:54 unix.sym1
-> unix.txt
106301761 lrwxrwxrwx. 1 s16010980 class   8 2021-09-27 21:55 unix.sym2
-> unix.txt
106301758 -rw-r--r--. 1 s16010980 class 412 2021-09-27 21:54 unix.txt
```

데이터 크기가 다름.

```
# 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)

stat 를 사용하면 안됨.
stat 을 사용하면 심볼릭링크가 공유하고 있는 원본 파일의 정보를 보여줌.

```
#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 정보가 검색된다.**

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 경로에 대한 내용 상대경로가 나옴.

원본 파일의 경로 읽기 : realpath(3)

절대경로가 나옴.

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

상대 경로 : cd 2021/0927 에서의 2021/0927

- **심볼릭 링크가 가리키는 원본 파일의 실제 경로명 출력**

절대 경로 : /.../.../ 로 되어 있는 것.
루트로부터의 경로.

[예제 3-10] lstat 함수 사용하기 (test3.c)

```
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      printf("1. stat : unix.txt ---\n");
10      stat("unix.txt", &buf);
11      printf("unix.txt : Link Count = %d\n", (int)buf.st_nlink);
12      printf("unix.txt : Inode = %d\n", (int)buf.st_ino);
13
14      printf("2. stat : unix.sym ---\n");
15      stat("unix.sym", &buf);
16      printf("unix.sym : Link Count = %d\n", (int)buf.st_nlink);
17      printf("unix.sym : Inode = %d\n", (int)buf.st_ino);
18
19      printf("3. lstat : unix.sym ---\n");
20      lstat("unix.sym", &buf);
```

[예제 3-10] lstat 함수 사용하기

```
21      printf("unix.sym : Link Count = %d\n", (int)buf.st_nlink);
22      printf("unix.sym : Inode = %d\n", (int)buf.st_ino);
23
24      return 0;
25  }
```

```
# ls -li unix*
192 -rwxrwx---   2 root     other    24  1월   8일   15:47 unix.ln
202 lrwxrwxrwx   1 root     other     8  1월 11일   18:48 unix.sym->unix.txt
192 -rwxrwx---   2 root     other    24  1월   8일   15:47 unix.txt
# ex3_10.out
1. stat : unix.txt ---
unix.txt : Link Count = 2
unix.txt : Inode = 192
2. stat : unix.sym ---
unix.sym : Link Count = 2
unix.sym : Inode = 192
3. lstat : unix.sym ---
unix.sym : Link Count = 1
unix.sym : Inode = 202
```

[예제 3-11] readlink 함수 사용하기 (test4.c)

```
01  #include <sys/stat.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      char buf[BUFSIZ];
08      int n;
09
10      n = readlink("unix.sym", buf, BUFSIZ);
11      if (n == -1) {
12          perror("readlink");
13          exit(1);
14      }
15
16      buf[n] = '\0';
17      printf("unix.sym : READLINK = %s\n", buf);
18
19      return 0;
20  }
```

```
# ex3_11.out
unix.sym : READLINK = unix.txt
# ls -l unix.sym
lrwxrwxrwx   1 root other 8  1월  11일    18:48 unix.sym ->unix.txt
```

[예제 3-12] realpath 함수 사용하기

```
01  #include <sys/stat.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      char buf[BUFSIZ];
07
08      realpath("unix.sym", buf);
09      printf("unix.sym : REALPATH = %s\n", buf);
10
11      return 0;
12  }
```

```
# ex3_12.out
unix.sym : REALPATH = /export/home/jw/syspro/ch3/unix.txt
```

실습

unix.txt에 대한 두 개의 하드링크(unix1.ln, unix2.in)와 두개의 심볼릭 링크를 생성한다 (unix1.sym, unix2.sym)

입력 받은 파일의 확장자가 .ln 이면 inode와 링크 개수를 출력하고, .sym이면 심볼릭 링크 이름, inode, 링크개수, 원본파일경로를 출력하는 프로그램을 작성하라

```
int match(char *s1, char *s2) {

        int diff = strlen(s1) - strlen(s2);

        return(strcmp(&s1[diff], s2) == 0);
}
```

디렉토리 허가

- .과 ..은 모든 디렉토리에 항상 존재하는 파일 이름이며, 디렉토리가 생성될 때 자동적으로 포함됨
  - ls . 현재 디렉토리
  - cd .. 부모 디렉토리

[s16010980@sce test]$ ls -a
. ..

- 디렉토리 허가
  - 읽기 : 디렉토리 내의 파일이나 부디렉토리의 이름을 리스트
  - 쓰기 : 디렉토리 내의 파일을 제거하거나 새로운 파일을 생성
  - 실행 : cd혹은 chdir로 디렉토리 내부로 들어갈 수 있음

디렉토리 안에 있는 파일을 읽기 위해선 그 파일에 대한 읽기 권한이 있어야 함.

파일을 수정하기 위해서는 파일에 대한 쓰기 권한이 있어야 함. 디렉토리 쓰기 권한과는 다름

디렉토리 안에 있는 파일을 실행하기 위해서는 그 파일에 대한 실행권한이 있어야 함.
디렉토리 권한 아님!

디렉토리 관련 함수[1]

디렉토리 생성: mkdir(2)

함수임. 명령어 아님.

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

- path에 지정한 디렉토리를 mode 권한에 따라 생성한다.

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);
```

[예제 3-13] 디렉토리 생성/삭제/이름 변경하기 (test5.c)

```
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

[예제 3-13] 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);
```

- 현재 작업 디렉토리 위치를 알려주는 명령은 pwd, 함수는 getcwd

디렉토리 이동: chdir(2)

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

[예제 3-14] 작업 디렉토리 위치 검색/디렉토리 이동하기 (test6.c)

```
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
```