UNIT

07

파일과 디렉토리

로봇SW 교육원

최상훈(shchoi82@gmail.com)

학습 목표

- 리눅스 파일의 속성 이해

stat, fstat, lstat


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

- **기능** : **파일에 대한 정보를 얻어옴**
- **리턴 값** : **성공하면** 0, **실패하면** -1
- *buf* : stat **구조체 포인터**
- lstat( )**는 심볼릭 링크 파일 자체에 대한 정보를 돌려줌**

stat 구조체

- <sys/stat.h> 에 정의

```
struct stat {
    mode_t st_mode;    /* type & mode (permissions) */
    ino_t st_ino;      /* i-node number (serial number) */
    dev_t st_dev;      /* device no (file system) */
    dev_t st_rdev;     /* device no for special file */
    nlink_t st_nlink;  /* # of links */
    uid_t st_uid;      /* user ID of owner */
    gid_t st_gid;      /* group ID of owner */
    off_t st_size;     /* sizes in byes, for regular files */
    struct timespec st_atim;  /* time of last access */
    struct timespec st_mtim;  /* time of last modification */
    struct timespec st_ctim;  /* time of last status change */
    long st_blk_size;  /* best I/O block size */
    long st_blocks;    /* number of disk blocks allocated */
};
```

```
struct timespec {
    time_t tv_sec;
    Long tv_spec;
}
```

파일의 종류(st_mode)

- **정규 파일**(regular)
  - **데이터를 포함하고 있는 텍스트 또는 이진 파일**
- **디렉토리 파일**(directory)
  - **다른 파일들의 이름과 그 파일에 대한 정보를 가리키는 포인터를 담은 파일**
- **블록 특수 파일** (block special)
  - **시스템에 장착된 장치를 가리키는 파일**
  - **고정 크기 단위의 버퍼링** I/O**접근을 제공**

    예) DISK
- **문자 특수 파일**(charcter special)
  - **시스템에 장착된 장치를 가리키는 파일**
  - **가변 크기단위 비버퍼링** I/O**접근을 제공**

    예) **키보드**
- FIFO(first in first out)
  - **프로세스 간 통신에 사용되는 파일** (**명명된** pipe **라고도 불림**)
- **소켓** (socket)
  - **네트워크를 통한 프로세스 간 통신에 사용되는 파일**
- **심볼릭 링크**(symbolic link)
  - **다른 파일을 가리키는 파일**

파일의 종류(st_mode)

- **파일 타입을 검사하는 매크로 함수** (<sys/stat.h> **에 정의**)
  - S_ISREG( ) : **정규 파일**
  - S_ISDIR( ) : **디렉토리 파일**
  - S_ISCHR( ) : **문자 특수 파일**
  - S_ISBLK( ) : **블록 특수 파일**
  - S_ISFIFO( ) : pipe **또는** FIFO
  - S_ISLNK( ) : **심볼릭 링크**
  - S_ISSOCK( ) : **소켓**

```
ex) if(S_ISREG(buf.st_mode))
        printf("this is a regular file.");
```

실습1-1:stat

- **파일의 종류 확인하기**
- **파일명** : stat.c

```
#include<stdio.h>
#include<sys/stat.h>
int main(int argc, char *argv[])
{
    int i;
    struct stat buf;
    char *ptr;
    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (stat(argv[i], &buf) < 0) {
            printf("lstat error\n");
            continue;
        }
        if (S_ISREG(buf.st_mode))           ptr = "regular";
        else if (S_ISDIR(buf.st_mode))      ptr = "directory";
        else if (S_ISCHR(buf.st_mode))      ptr = "character special";
        else if (S_ISBLK(buf.st_mode))      ptr = "block special";
        else if (S_ISFIFO(buf.st_mode))     ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))      ptr = "symbolic link";
        else if (S_ISSOCK(buf.st_mode))     ptr = "socket";
        else                                ptr = "** unknown mode **";
        printf("%s\n", ptr);
    }
    return 0;
}
```

실습1-2:stat


```
$ ./stat .
.: directory
$ ./stat ..
..: directory
$ ./stat stat.c
stat.c: regular
$ ./stat /dev/tty
/dev/tty: character special
$ ./stat /dev/mmcblk0
/dev/mmcblk0: block special
$
```

파일의 접근 모드(st_mode)


```
#define S_IRWXU 00700
    #define S_IRUSR 00400       /* read : user */
    #define S_IWUSR 00200       /* write : user */
    #define S_IXUSR 00100       /* execute : user */
#define S_IRWXG 00070
    #define S_IRGRP 00040       /* read : group */
    #define S_IWGRP 00020       /* write : group */
    #define S_IXGRP 00010       /* execute : group */
#define S_IRWXO 00007
    #define S_IROTH 00004       /* read : other */
    #define S_IWOTH 00002       /* write : other */
    #define S_IXOTH 00001       /* execute : other */
```

파일의 접근 권한

- **파일의 소유 정보**
  - **사용자** (st_uid) : **파일을 소유한 사용자** ID
  - **그룹**(st_gid) : **파일을 소유한 그룹** ID
- **파일에 대한 읽기, 쓰기, 실행 권한**
  - **사용자**(user), **그룹**(group), **기타**(other)**으로 구분하여 관리**
  - rwxrwxrwx, 0777
  - **"접근 허용 비트" 라고 보통 부름**
- **디렉토리에 대한 읽기, 쓰기, 실행 권한**
  - **읽기**(read)**권한**
    - **디렉토리 항목의 이름을 읽는 권한**
  - **쓰기**(write)**권한**
    - **디렉토리 항목을 생성,삭제,수정할수 있는 권한**
    - execute**권한 필요**
  - **실행**(execute)**권한**
    - **디렉토리안으로 이동/검색할수 있는 권한**
    - Search bit **라고도 함**

실습2:파일의 접근 권한 변경(chmod)


```
$ touch testfile
$ ls -l testfile
-rw-r--r-- 1 pi pi 0 Feb  9 12:53 testfile
$ chmod 0 testfile
$ ls -l testfile
---------- 1 pi pi 0 Feb  9 12:53 testfile
$ chmod 0777 testfile
$ ls -l testfile
-rwxrwxrwx 1 pi pi 0 Feb  9 12:53 testfile
$ chmod u-rw testfile
$ ls -l testfile
---xrwxrwx 1 pi pi 0 Feb  9 12:53 testfile
$ chmod o-r testfile
$ ls -l testfile
---xrwx-wx 1 pi pi 0 Feb  9 12:53 testfile
$ chmod 0 testfile
$ ls -l testfile
---------- 1 pi pi 0 Feb  9 12:53 testfile
$ chmod u+rw testfile
$ ls -l testfile
-rw------- 1 pi pi 0 Feb  9 12:53 testfile
$ chmod g+rw testfile
$ ls -l testfile
-rw-rw---- 1 pi pi 0 Feb  9 12:53 testfile
$
```

실습3:파일의 접근 권한 변경(chmod)


```
$ mkdir dir
$ touch dir/a
$ chmod a-w dir
$ ls -l
total 16
dr-xr-xr-x 2 pi pi 4096 Feb  9 12:58 dir
-rwxr-xr-x 1 pi pi 6220 Feb  9 12:46 stat
-rw-r--r-- 1 pi pi  708 Feb  9 12:46 stat.c
-rw-rw---- 1 pi pi    0 Feb  9 12:53 testfile
$ rm dir/a
rm: cannot remove `dir/a': Permission denied
$ chmod u+w dir
$ rm dir/a
$ ls -l
total 16
drwxr-xr-x 2 pi pi 4096 Feb  9 12:59 dir
-rwxr-xr-x 1 pi pi 6220 Feb  9 12:46 stat
-rw-r--r-- 1 pi pi  708 Feb  9 12:46 stat.c
-rw-rw---- 1 pi pi    0 Feb  9 12:53 testfile
$
```

실습4:파일을 소유한 사용자/그룹 ID

- **파일의 소유 사용자/그룹** ID **확인하기**
- **파일명** : owner.c

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

int main(int argc, char *argv[])
{
    struct stat buf;
    if(stat(argv[1], &buf) == -1){
        printf("stat error\n");
        return 1;
    }
    printf("file onwer user  : %d\n", buf.st_uid);
    printf("file owner group : %d\n", buf.st_gid);
    return 0;
}
```

```
$ ./owner /etc/passwd
file onwer user  : 0
file owner group : 0
$ ./owner ./owner
file onwer user  : 1000
file owner group : 1000
$ ./owner /usr/bin/passwd
file onwer user  : 0
file owner group : 0
```

프로세스와 관련된 사용자/그룹 ID

- **프로세스는 여러 가지** ID**와 관련되어 있음**
  - **실제**(real) **사용자/그룹** ID
    - **로그인 시 패스워드 파일에서 읽어 오는** ID
    - **로그인 세션 동안에는 바뀌지 않음**
  - **유효**(effective) **사용자/그룹** ID, **추가 그룹** ID
    - **파일에 대한 접근 권한을 결정함**
  - **저장된** set-user-ID(saved SUID), **저장된** group-user-ID(saved SGID)
    - exec( ) **함수 수행 시 저장된 유효 사용자/그룹** ID

- **일반적인 프로그램을 실행**
  - **유효 사용자/그룹** ID = **실제 사용자/그룹** ID

- set-user-ID(SUID), set-group-ID(SGID)**비트가 설정된 파일을 실행**
  - **유효 사용자/그룹** ID = **실행 파일의 소유 사용자/그룹** ID

실습: 프로세스와 관련된 ID


```
uid_t getuid(void);
                                        Returns : real user ID of calling process
uid_t geteuid(void);
                                    Returns : effective user ID of calling process
gid_t getgid(void);
                                        Returns : real group ID of calling process
gid_t getegid(void);
                                    Returns : effective group ID of calling process
```

파일명 : ids.c

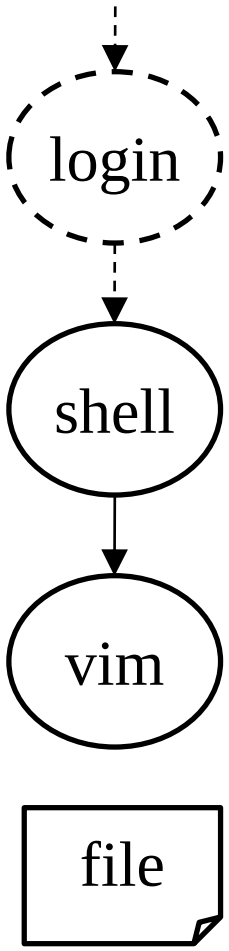
```
#include<stdio.h>
#include<unistd.h>

int main(void)
{
    printf("getuid:%d\n", getuid());
    printf("geteuid:%d\n", geteuid());
    printf("getgid:%d\n", getgid());
    printf("getegid:%d\n", getegid());
    return 0;
}
```

접근 권한 검사

- **프로세스가 파일에 접근할 때 접근하기 위해 쓰이는** ID
  - **유효 사용자/그룹** ID, **추가 그룹** ID**를 사용**

- **접근 권한 점검 순서**
  1. **프로세스의 유효 사용자** ID**가** 0(root)**이면 접근 허용**
  2. **프로세스의 유효 사용자** ID**와 파일을 소유한 사용자** ID**가 같으면 사용자**(user) **접근 권한 비트 확인**
  3. **프로세스의 유효 그룹** ID**와 파일을 소유한 그룹** ID**가 같으면 그룹**(group) **접근 권한 비트 확인**
  4. **기타**(other) **접근 권한 비트 확인**

파일의 접근 권한 검사 과정1



<shell 에서 pi 사용자가 vim file 명령어 실행시 접근검사>

1. shell 프로세스는 vim를 기타 권한으로 실행
   lrwxrwxrwx 1 root root 21 Jan 29 03:40 /usr/bin/vim
2. vim 프로세스의 RUID, EUID는 shell 프로세스로부터 상속받음
3. vim 프로세스가 file을 열때 vim 프로세스의 EUID가 file의 소유한 사용자 ID와 같고 접근허용비트의 사용자(user) 읽기(r) 권한이 있기 때문에 읽기 접근허용

유효 사용자 ID

- **보통의 경우 유효 사용자ID는 실제 사용자 ID와 동일함**
- **보통의 경우 유효 그룹ID는 실제 그룹 ID와 동일함**

- Set User-ID (SUID)
  - "**이 파일이 실행될 때 프로세스의 유효 사용자 ID를 파일을 소유한 사용자 ID로 설정해라!**"
- Set Group-ID (SGID)
  - "**이 파일이 실행될때 프로세스의 유효 그룹 ID를 파일을 소유한 그룹 ID로 설정해라!**"

- IS_SUID, IS_SGID **상수를 사용하여 알 수 있음**

- SUID **프로그램 예**
  - /usr/bin/passwd

실습5:SUID, SGID

- **파일의** SUID**와** SGID **비트 확인하기**
- **파일명** : suidsgid.c

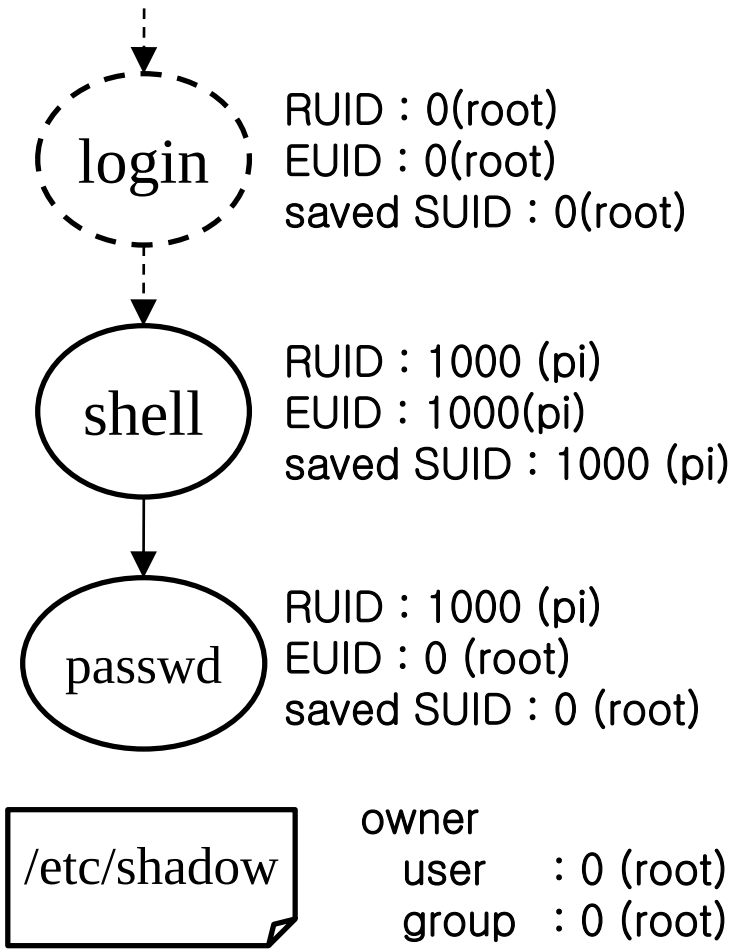
```
#include<stdio.h>
#include<sys/stat.h>

int main(void)
{
    struct stat buf;
    if(stat("/usr/bin/passwd", &buf) == -1){
        printf("stat error\n");
        return 1;
    }
    printf("st_uid : %d\n", buf.st_uid);
    printf("st_gid : %d\n", buf.st_gid);
    if(S_ISUID & buf.st_mode)
        printf("SUID\n");
    if(S_ISGID & buf.st_mode)
        printf("SGID\n");
    return 0;
}
```

```
$ ./suidsgid
st_uid : 0
st_gid : 0
SUID
$
```

실습6:SUID/SGID비트 설정(chmod)


```
$ sudo passwd
sudo: unable to resolve host raspberrypi-robotcode77
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
$ su
Password:
# ls -l stat
-rwxr-xr-x 1 pi pi 6220 Feb  9 12:46 stat
# chown root:root stat
# ls -l stat
-rwxr-xr-x 1 root root 6220 Feb  9 12:46 stat
# chmod u+s stat
# ls -l stat
-rwsr-xr-x 1 root root 6220 Feb  9 12:46 stat
# chmod g+s stat
# ls -l stat
-rwsr-sr-x 1 root root 6220 Feb  9 12:46 stat
# exit
$
```

파일의 접근 권한 검사 과정2



<shell 에서 pi 사용자가 passwd 명령어 실행시 접근검사>

1. shell 프로세스는 passwd를 기타 권한으로 실행 가능
   -rwsr-xr-x 1 root root 41280 Jun 5 2012 /usr/bin/passwd
2. passwd 프로세스 실행시 RUID는 shell로부터 상속받음
3. SUID 비트가 설정 되어있으므로 passwd프로세스의 EUID를 passwd 파일의 소유자 0(root)ID로 변경함
4. 사용자로부터 패스워드가 입력받음
5. passwd 프로세스 EUID와 /etc/shadow 파일을 소유한 사용자 ID가 같고 사용자(user)의 쓰기(w) 권한이 있기 때문에 쓰기 접근 허용

sticky bit

- ***기능***
  - ***처음 실행시 파일의 이미지를 프로그램의 종료 후에도 swap 영역에 저장되어 프로그램의 빠른 재실행***
  - ***한 시스템에서 동시 사용 가능한 sticky bit 프로그램의 수는 제한적***
  - ***초기의 느린 디스크 입출력 속도 때문에 사용되었으나, 가상 메모리 기법과 빠른 파일 시스템으로 인하여 현재는 이러한 용도로는 사용 안함***

- **새로운 기능**
  - **디렉토리에 적용**
    - **디렉토리 항목의 권한(생성, 수정, 삭제)에 추가 조건을 설정함**
      - sticky bit **적용된 디렉토리 대한 권한**
        - » **슈퍼 사용자**
        - » **디렉토리 소유자**
        - » **파일의 소유자**
  - Sticky bit**가 적용된 디렉토리 예**: /tmp

- S_ISVTX
  - st_mode & S_ISVTX

실습7:sticky bit(1/3)


```
$ sudo adduser student1
sudo: unable to resolve host raspberrypi-robotcode77
Adding user `student1' ...
Adding new group `student1' (1004) ...
Adding new user `student1' (1001) with group `student1' ...
Enter new UNIX password:
Retype new UNIX password:
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
$ sudo adduser student2
sudo: unable to resolve host raspberrypi-robotcode77
Adding user `student2' ...
Adding new group `student2' (1005) ...
Adding new user `student2' (1002) with group `student2' ...
Enter new UNIX password:
Retype new UNIX password:
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
$
```

실습7:sticky bit(2/3)


```
pi@raspberrypi ~/08 $ mkdir stickydir
pi@raspberrypi ~/08 $ ls -l
total 4
drwxr-xr-x 2 pi pi 4096 Feb  9 13:51 stickydir
pi@raspberrypi ~/08 $ chmod o+w stickydir/
pi@raspberrypi ~/08 $ ls -l
total 4
drwxr-xrwx 2 pi pi 4096 Feb  9 13:51 stickydir
pi@raspberrypi ~/08 $ su student1
Password:
student1@raspberrypi /home/pi/08 $ cd stickydir/
student1@raspberrypi /home/pi/08/stickydir $ cat > student1file
this is student1's file.
[ctrl+d]student1@raspberrypi /home/pi/08/stickydir $
student1@raspberrypi /home/pi/08/stickydir $ exit
pi@raspberrypi ~/08 $ su student2
Password:
student2@raspberrypi /home/pi/08 $ cd stickydir/
student2@raspberrypi /home/pi/08/stickydir $ cat > student2file
this is student2's file.
[ctrl+d]student2@raspberrypi /home/pi/08/stickydir $
student2@raspberrypi /home/pi/08/stickydir $ exit
pi@raspberrypi ~/08 $ ls -l
total 4
drwxr-xrwx 2 pi pi 4096 Feb  9 13:54 stickydir
pi@raspberrypi ~/08 $ ls -l stickydir/
total 8
-rw-r--r-- 1 student1 student1 24 Feb  9 13:53 student1file
-rw-r--r-- 1 student2 student2 24 Feb  9 13:54 student2file
pi@raspberrypi ~/08 $
```

실습7:sticky bit(3/3)

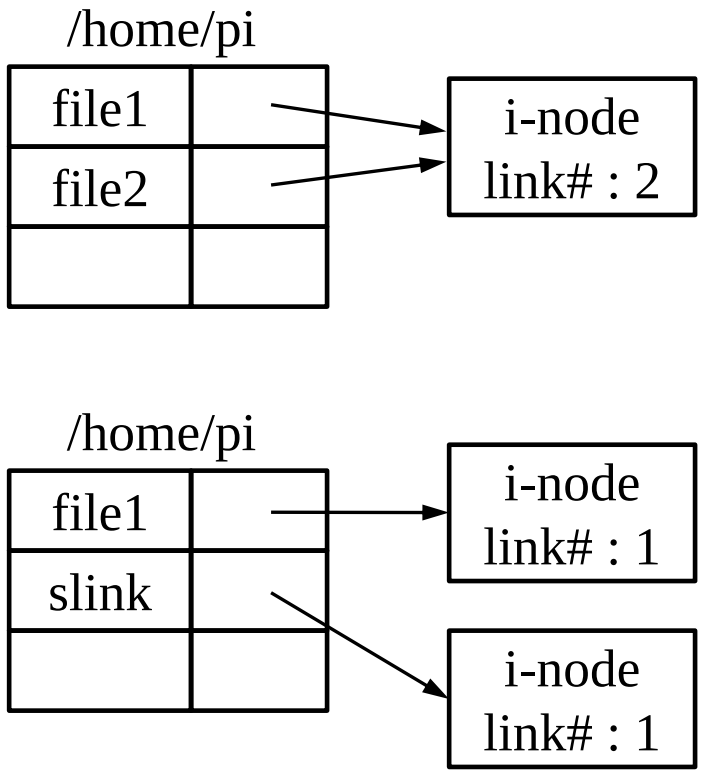

```
pi@raspberrypi ~/08 $ su student1
Password:
student1@raspberrypi /home/pi/08 $ rm stickydir/student2file
rm: remove write-protected regular file `stickydir/student2file'? y
student1@raspberrypi /home/pi/08 $ ls -l stickydir/
total 4
-rw-r--r-- 1 student1 student1 24 Feb  9 13:53 student1file
student1@raspberrypi /home/pi/08 $ exit
pi@raspberrypi ~/08 $ chmod +t stickydir/
pi@raspberrypi ~/08 $ ls -l
total 4
drwxr-xrwt 2 pi pi 4096 Feb  9 14:01 stickydir
pi@raspberrypi ~/08 $ su student2
Password:
student2@raspberrypi /home/pi/08 $ rm stickydir/student1file
rm: remove write-protected regular file `stickydir/student1file'? y
rm: cannot remove `stickydir/student1file': Operation not permitted
student2@raspberrypi /home/pi/08 $ exit
pi@raspberrypi ~/08 $ su student1
Password:
student1@raspberrypi /home/pi/08 $ rm stickydir/student1file
student1@raspberrypi /home/pi/08 $
```

Hard link 와 Symbolic link

- Hard Link
  - **한 파일의** i-node**를 여러 개의 디렉토리 항목들이 가리킬수 있음**
  - **같은** i-node **번호를 갖음**

- Symbolic Link (Soft Link)
  - **파일에 대한 간접 포인터**
  - **파일의 내용은 파일의 경로**(link)
  - **파일의 크기는 파일 경로의 길이**
  - MS Windows**의 바로가기와 유사함**

- Hard vs Symbolic



slink:
/home/advc/file1

실습8:Hard link 와 Symbolic link


```
$ ls -al
total 8
drwxr-xr-x  2 pi pi 4096 Feb  9 14:06 .
drwxr-xr-x 15 pi pi 4096 Feb  9 13:14 ..
$ touch file1
$ ln file1 file2
$ ls -li
total 0
15430 -rw-r--r-- 2 pi pi 0 Feb  9 14:06 file1
15430 -rw-r--r-- 2 pi pi 0 Feb  9 14:06 file2
$ ln -s /home/pi/08/file1 slink
$ ls -li
total 0
15430 -rw-r--r-- 2 pi pi  0 Feb  9 14:06 file1
15430 -rw-r--r-- 2 pi pi  0 Feb  9 14:06 file2
51015 lrwxrwxrwx 1 pi pi 17 Feb  9 14:08 slink -> /home/pi/08/file1
$
```

link


```
#include <unistd.h>

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

- **기능**: **이미 존재하는 파일에 대해서 새로운 디렉토리 항목을 만듬** (hard link **생성**)
- **리턴 값**: **성공하면** 0, **실패하면** -1
- **성공하면**, **해당** i-node**에 대한 연결 개수** (link count) 1 **증가**

- Hard link : **한 파일의** i-node**를 여러 개의 디렉토리 항목들이 가리킬수 있음**

unlink


```
#include <unistd.h>

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

- **기능**: **해당 디렉토리 항목을 지우고**, **연결 개수를 감소시킴**
- **리턴 값**: **성공하면** 0, **실패하면** -1
  - **연결 개수가** 0**이 되면**, **파일을 디스크에서 제거함**

- **파일의 내용은 링크 횟수가** 0**에 도달했을 때에만 삭제됨**
- **프로세스에의해 열려있다면 삭제 되지 않고**
  파일이 닫히면 커널이 파일의 개수를 점검하고 0**이면 삭제함**
- Symbolic link**를 따라가지 않음**

실습9 : unlink(1/2)

- **임시파일 사용하기**
- **파일명** : tempfile.c

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

int main(void)
{
    int fd, len;
    char buf[20];

    if ((fd = open("tempfile", O_RDWR | O_CREAT | O_TRUNC, 0666)) == -1) {
        printf("open error\n");
        return 1;
    }
    unlink("tempfile");
    if(write(fd, "How are you?", 12) != 12) {
        printf("write error\n");
        return 1;
    }
    lseek(fd, 0L, SEEK_SET);
```

실습9 : unlink(2/2)


```
    if((len = read(fd, buf, sizeof(buf))) <=0){
        printf("read error\n");
        return 1;
    }
    else
        buf[len] = '\0';

    printf("%s\n", buf);
    close(fd);
    if ((fd = open("tempfile", O_RDWR)) < 0) {
        printf("Second open error\n");
        return 1;
    }
    close(fd);
    return 0;
}
```

```
$ ./tempfile
How are you?
Second open error
$
```

remove


```
#include <unistd.h>

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

- **기능**: **파일의 연결 개수를 감소 시킴**
- **리턴 값**: **성공하면** 0, **실패하면** -1
- pathname**이 파일이면** unlink( )**와 같은 동작을 하고**, **디렉토리이면** rmdir( )**과 같은 역할**

symlink, readlink


```
#include <unistd.h>
int  symlink (const char *actualpath, const char *sympath );
int  readlink (const char *pathname, char *buf, int bufsize );
```

- symlink( )
  - **기능**: **심볼릭 링크를 만듬**
  - **리턴 값**: **성공하면** 0, **실패하면** -1
  - actualpath : **실제 파일**
  - sympath : **심볼릭 링크의 이름**

- readlink( )
  - **기능**: **심볼릭 링크 파일을 읽는 함수**
  - **리턴 값**: **성공하면 읽은 바이트 수**, **실패하면** -1

실습10 : symlink, readlink 예제

- **심볼릭 링크 생성**
- **파일명** : symlinkEx.c

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

int main(void)
{
    printf("%d\n", symlink("./file", "slink"));
    return 0;
}
```

실습11 : symlink, readlink 예제

- **심볼릭 링크 읽기**
- **파일명** : readlinkEx.c

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

int main(void)
{
    char buf[1024] = {0};
    int ret;
    ret = readlink("./slink", buf, sizeof(buf));

    printf("ret:%d\n", ret);
    printf("buf:%s\n", buf);

    return 1;
}
```

파일의 시간 정보

- **마지막 접근 시간**
  - stat**구조체의** st_atime
  - read **연산**
  - ls -lu
- **마지막 수정 시간**
  - stat**구조체의** st_mtime
  - write **연산**
  - ls -l
- i-node **상태의 마지막 수정 시간**
  - stat **구조체의** st_ctime
  - chmod, chown, link **등등**
  - ls -lc

utime


```
#include <sys/types.h>
#include <utime.h>
int  utime (const char *pathname, const struct utimbuf *times );
```

- **기능**: **파일의 시간 설정**
- **리턴 값**: **성공하면** 0, **실패하면** -1
- utimbuf **구조체**

```
struct utimbuf {
    time_t  actime;          /* 마지막 접근 시간*/
    time_t  modtime;         /* 마지막 수정 시간 */
}
```

  - **각 필드는** 1970. 1.1. 00:00 **부터 현재까지의 시간을 초로 환산한 값**
  - times**가** NULL **이면**, **현재시간으로 설정됨**

utime 예제


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

int main(int argc, char *argv[])
{
    int i, fd;
    struct stat statbuf;
    struct utimbuf timebuf;

    for (i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) {  /* fetch current times */
            printf("%s: stat error", argv[i]);
            continue;
        }

        if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate */
            printf("%s: open error", argv[i]);
            continue;
        }
        close(fd);

        timebuf.actime  = statbuf.st_atime;
        timebuf.modtime = statbuf.st_mtime;
```

utime 예제


```
        if (utime(argv[i], &timebuf) < 0) {     /* reset times */
            printf("%s: utime error", argv[i]);
            continue;
        }
    }
    return 0;
}
```

```
advc@shchoi82:~/source/ch4$ ls -l test
-rw-r--r-- 1 advc advc 0 2011-07-30 21:21 test
advc@shchoi82:~/source/ch4$ ls -lu test
-rw-r--r-- 1 advc advc 0 2011-07-30 21:21 test
advc@shchoi82:~/source/ch4$ ls -lc test
-rw-r--r-- 1 advc advc 0 2011-07-30 21:21 test
advc@shchoi82:~/source/ch4$ ./a.out test
advc@shchoi82:~/source/ch4$ ls -l test
-rw-r--r-- 1 advc advc 0 2011-07-30 21:21 test
advc@shchoi82:~/source/ch4$ ls -lu test
-rw-r--r-- 1 advc advc 0 2011-07-30 21:21 test
advc@shchoi82:~/source/ch4$ ls -lc test
-rw-r--r-- 1 advc advc 0 2011-07-31 13:38 test
advc@shchoi82:~/source/ch4$ date
2011. 07. 31. (일) 13:39:04 KST
advc@shchoi82:~/source/ch4$
```

mkdir, rmdir


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

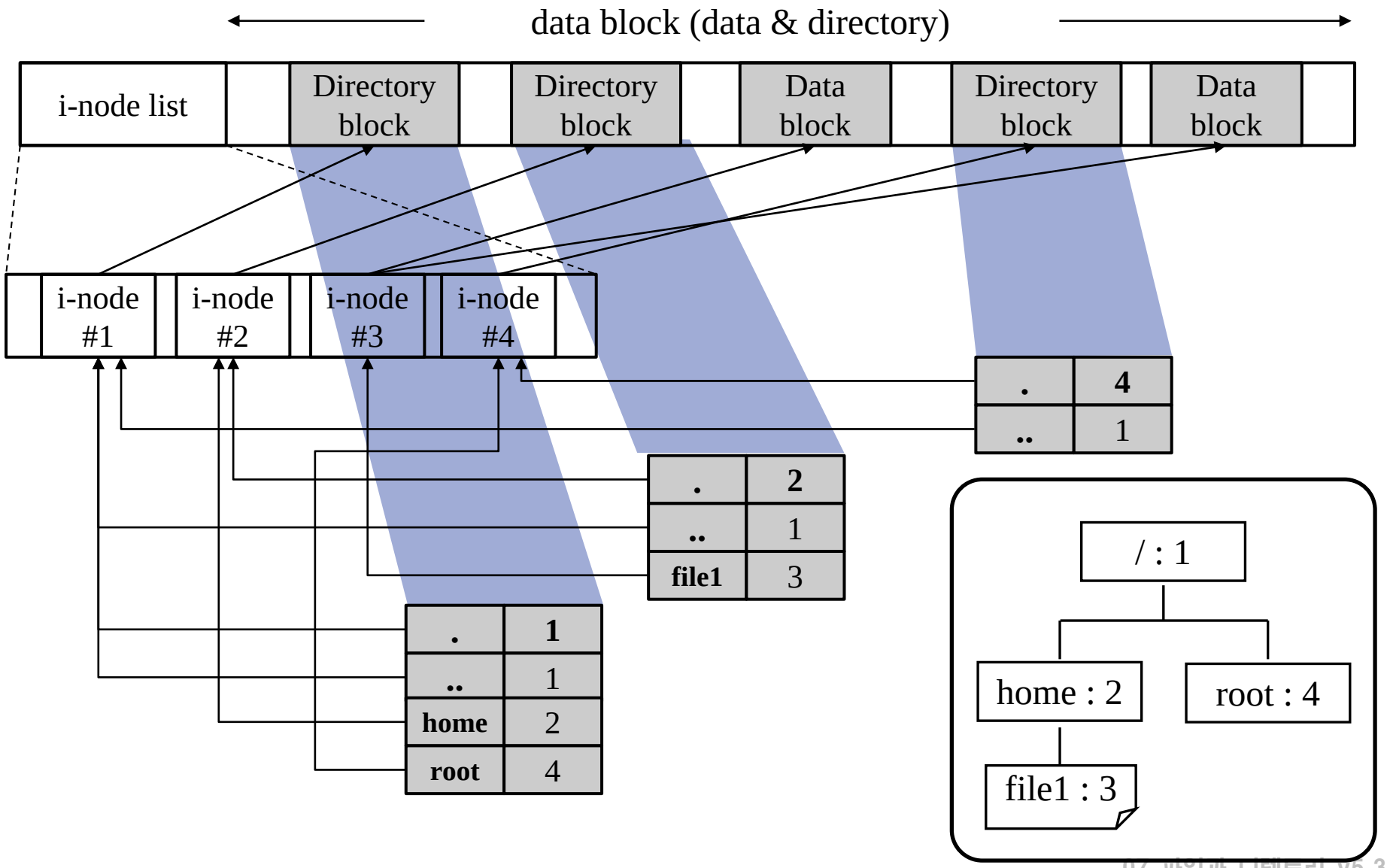
- mkdir( )
  - **기능** : **새로운 디렉토리를 만듬**
  - **리턴 값**: **성공하면** 0, **실패하면** -1
  - **성공하면** . **와** . . **파일은 자동적으로 만들어짐**
- rmdir( )
  - **기능** : **비어 있는 디렉토리를 삭제함** (**연결 개수** 1 **감소**)
  - **리턴 값**: **성공하면** 0, **실패하면** -1
  - **디렉토리에 대한 영역이 해제 되는 것은 연결 개수가** 0**이 될 때임**

디렉토리의 구조

- 디렉토리
  - '파일' 및 '디렉토리'의 이름 저장하는 테이블
  - 테이블의 각 항목의 구성
    - dirent **구조체** dirent.h

```
struct dirent {
      ino_t  d_ino;                    /* i-node 번호 */
      char   d_name[NAME_MAX + 1];     /* 파일명 */
}
```

디렉토리의 구조


data block (data & directory)
i-node list
Directory block
Directory block
Data block
Directory block
Data block
i-node #1
i-node #2
i-node #3
i-node #4
. 4
.. 1
. 2
.. 1
file1 3
. 1
.. 1
home 2
root 4
/ : 1
home : 2
root : 4
file1 : 3

opendir, readdir


```
#include <dirent.h>
DIR *opendir (const char *pathname);
struct dirent *readdir(DIR *dp);
```

- opendir
  - **기능** : DIR**구조체의 포인터를 리턴**
  - **리턴 값**: **성공하면** DIR**구조체 포인터**, **실패하면** NULL
  - **해당 디렉토리에 대한 읽기 권한이 있어야 함**
  - DIR **구조체** : **디렉토리에 대한 정보**

- readdir
  - **기능** : DIR **구조체의 다음 디렉토리항목을 가져옴**
  - **리턴 값** : **성공시** dirent **구조체 포인터**, **디렉토리의 끝이거나 실패시** NULL

rewinddir, closedir


```
#include <dirent.h>
void rewinddir (DIR *dp);
int closedir (DIR *dp);
```

- rewinddir( )
  - **기능**: DIR**구조체를 시작 디렉토리항목으로 초기화**
- closedir( )
  - **기능**: DIR **구조체를 닫음**
  - **리턴 값**: **성공하면** 0, **실패하면** -1

opendir 예제

- **디렉토리의 항목 읽기**
- **파일명** : readDirent.c

```
#include<stdio.h>
#include<dirent.h>
int main(int argc, char *argv[])
{
    DIR *dp;
    struct dirent *dirp;

    if (argc != 2){
        printf("usage: ./a.out <directory_name>\n");
        return 1;
    }
    if ((dp = opendir(argv[1])) == NULL){
        printf("can't open %s", argv[1]);
        return 1;
    }
    while ((dirp = readdir(dp)) != NULL){
        printf("%20s ", dirp->d_name);
        printf("%20ld\n", dirp->d_ino);
    }

    closedir(dp);
    return 0;
}
```

chdir, fchdir


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

- **기능: 현재 작업 디렉토리를 변경함**
- **반환 값: 성공하면 0, 실패하면 –1**

- **파일에 대한 작업을 할 때, 절대 경로를 쓰는 것 보다 상대 경로를 쓰는 것이 더 효율적임**
  - **현재 작업 디렉토리 내에서만 찾을 경우 루트에서 찾는 보다 더 효율적임**

    - fd1 = open(“/home/pi/data/file1”, O_RDONLY);
    - fd2 = open(“/home/pi/data/file2”, O_RDONLY);

    - chdir (“/home/pi/data/”);
    - fd1 = open(“file1”, O_RDONLY);
    - fd2 = open(“file2”, O_RDONLY);

getcwd


```
#include <unistd.h>

char *getcwd (char *buf, size_t size );
```

- **기능**: **현재 작업 디렉토리를 얻음**
- **리턴 값**: **성공하면** buf, **실패하면** NULL
- **프로세스가 변경한 현재 작업 디렉토리의 값은**
  부모 프로세서의 현재 작업 디렉토리에는 영향을 주지 않음
  (각 프로세스마다 자신의 현재 작업 디렉토리 정보를 갖고 있음)

chdir, getcwd 예제

- **프로세스의 현재 작업 디렉토리 변경**
- **파일명** : chdir.c

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

#define PATH_MAX 1024

int main(void)
{
    char path[PATH_MAX+1];
    if(chdir("/tmp")<0){
        printf("error chdir");
        return 1;
    }else{
        if(getcwd(path, PATH_MAX) == NULL){
            perror("error getcwd");
            return 1;
        }else
            printf("Current working directory changed to %s \n", path);
    }

    return 0;
}
```

```
$ pwd
/home/pi
$ ./chdir
Current working directory changed to
/tmp
$ pwd
/home/pi
```

fchdir 예제

- **프로세스의 현재 작업 디렉토리 변경**
- **파일명** : fchdir.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
int main(void)
{
    int fd;
    char buf[4096];
    if((fd = open("/etc", O_RDONLY)) == -1){
        printf("open error\n");
        return 1;
    }
    if(fchdir(fd) == -1){
        printf("fchdir error\n");
        return 1;
    }
    if(getcwd(buf, sizeof(buf)) == NULL){
        printf("getcwd error\n");
        return 1;
    }
    printf("%s\n", buf);
    return 0;
}
```