CS360 Lecture notes -- Links and inodes

- Jian Huang, referencing Dr. Plank's notes
- Directory: **~huangj/cs360/notes/Links**
- Lecture notes: **http://www.cs.utk.edu/~huangj/cs360/360/notes/Links/lecture.html**

Links

A *file* in Unix is a named collection of bytes on disk. We can think of this definition as having two parts:

- A collection of bytes on disk
- With a name

Each file is stored on disk in a certain disk location. For example, my file **.cshrc** is stored on the partition "**sd0h**" on the disk inside my sun workstation. Each physical file on disk has an associated data structure in the operating system, called an "*inode*". This inode contains information such as where the file is located on disk, its size, a special "inode number," which is unique, the protection mode, who owns the file, etc. If you are interested, here is a typical inode data structure. However, details of this data structure are beyond our scope.

```
struct inode {
        LIST_ENTRY(inode) i_hash;     /* Hash chain. */
        struct  vnode  *i_vnode;      /* Vnode associated with this inode. */
        struct  vnode  *i_devvp;      /* Vnode for block I/O. */
        u_int32_t i_flag;             /* flags, see below */
        dev_t     i_dev;              /* Device associated with the inode. */
        ino_t     i_number;           /* The identity of the inode. */

        union {                       /* Associated filesystem. */
                struct  fs *fs;       /* FFS */
        } inode_u;

        struct   klist i_knotes;      /* knotes attached to this vnode */
        struct   dquot *i_dquot[MAXQUOTAS]; /* Dquot structures. */
        u_quad_t i_modrev;            /* Revision level for NFS lease. */
        struct   lockf *i_lockf;      /* Head of byte-level lock list. */
        struct   lock__bsd__ i_lock; /* Inode lock. */
        /*
         * Side effects; used during directory lookup.
         */
        int32_t   i_count;            /* Size of free slot in directory. */
        doff_t    i_endoff;           /* End of useful stuff in directory. */
        doff_t    i_diroff;           /* Offset in dir, where we found last entry. */
        doff_t    i_offset;           /* Offset of free space in directory. */
        ino_t     i_ino;              /* Inode number of found directory. */
        u_int32_t i_reclen;           /* Size of found directory entry. */
        /*
         * The on-disk dinode itself.
         */
        struct  dinode i_din;         /* 128 bytes of the on-disk dinode. */
};

struct dinode {
```

```
        u_int16_t       di_mode;        /*   0: IFMT, permissions; see below. */
        int16_t         di_nlink;       /*   2: File link count. */
        union {
                u_int16_t oldids[2];    /*   4: Ffs: old user and group ids. */
                int32_t   inumber;      /*   4: Lfs: inode number. */
        } di_u;
        u_int64_t       di_size;        /*   8: File byte count. */
        int32_t         di_atime;       /*  16: Last access time. */
        int32_t         di_atimensec;   /*  20: Last access time. */
        int32_t         di_mtime;       /*  24: Last modified time. */
        int32_t         di_mtimensec;   /*  28: Last modified time. */
        int32_t         di_ctime;       /*  32: Last inode change time. */
        int32_t         di_ctimensec;   /*  36: Last inode change time. */
        ufs_daddr_t     di_db[NDADDR];  /*  40: Direct disk blocks. */
        ufs_daddr_t     di_ib[NIADDR];  /*  88: Indirect disk blocks. */
        u_int32_t       di_flags;       /* 100: Status flags (chflags). */
        u_int32_t       di_blocks;      /* 104: Blocks actually held. */
        int32_t         di_gen;         /* 108: Generation number. */
        u_int32_t       di_uid;         /* 112: File owner. */
        u_int32_t       di_gid;         /* 116: File group. */
        int32_t         di_spare[2];    /* 120: Reserved; currently unused */
};
```

The way we name a file is by attaching a "*link*" to the inode. Links are stored in "*directories*" -- each entry in a directory maps the name of the link to the inode number of the inode that points to the file.

For example, when we say

```
UNIX> cat > f1
This is f1
^D
UNIX>
```

This creates a file on disk whose contents are the bytes:

```
"This is f1\n"
```

An inode is created for that file which points to that file's location on disk. Moreover, a link is created in the current directory. This link maps the name **f1** to the inode just created.

You can use the "**-i**" flag of **ls** to see the inode number of a file:

```
UNIX> ls -i f1
34778 f1
UNIX>
```

We can have more than one link point to a file. Suppose we've made file **f1** above, and now we do the following:

```
UNIX> ln f1 f2
```

This says to create another link to the file **f1**, and call it "**f2**". Now we have two pointers to the same file. When we do a listing:

```
UNIX> ls -li f1 f2
34778  -rw-r--r--  2 plank          11 Sep 16 10:12 f1
34778  -rw-r--r--  2 plank          11 Sep 16 10:12 f2
UNIX> cat f1
This is f1
```

```
UNIX> cat f2
This is f1
UNIX>
```

We see that the files are exactly the same, except that the links have different names. If we change either of these files -- for example, let's edit **f2** using **vi**, and change the word "This" to "That", then the change is seen in both **f1** and **f2**:

```
UNIX> vi f2
...
UNIX> cat f2
That is f1
UNIX> cat f1
That is f1
UNIX> ls -li f1 f2
34778  -rw-r--r--  2 plank          11 Sep 16 10:14 f1
34778  -rw-r--r--  2 plank          11 Sep 16 10:14 f2
UNIX>
```

Note that even though we only modified **f2**, the file modification time for **f1** has changed as well. That is because file modification time is stored as part of the inode -- thus, when **f2** changes it, the change is seen in **f1** as well. Same with file protection modes. If we change the protection for **f1**, then we will see the changes in **f2**:

```
UNIX> chmod 0400 f1
UNIX> ls -li f1 f2
34778  -r--------  2 plank          11 Sep 16 10:14 f1
34778  -r--------  2 plank          11 Sep 16 10:14 f2
UNIX>
```

Note the third column of the **ls** command. It is the number of links to the file. If we make another link to **f1**, then this column will be updated:

```
UNIX> ln f1 f3
UNIX> ls -li f1 f2 f3
34778  -r--------  3 plank          11 Sep 16 10:14 f1
34778  -r--------  3 plank          11 Sep 16 10:14 f2
34778  -r--------  3 plank          11 Sep 16 10:14 f3
```

When we use the "**rm**" command, we are actually removing links. E.g.

```
UNIX> chmod 0644 f1
UNIX> rm f1
UNIX> ls -li f*
34778  -rw-r--r--  2 plank          11 Sep 16 10:14 f2
34778  -rw-r--r--  2 plank          11 Sep 16 10:14 f3
UNIX>
```

When the last link to a file is removed, then the file itself, inode and all, is deleted. As long as there is a link pointing to a file, however, the file remains. It is interesting to see what happens when files with links are overwritten. For example, suppose I do the following:

```
UNIX> cat > f2
This is now file f2
^D
UNIX> cat f2
This is now file f2
UNIX> cat f3
This is now file f2
```

By saying you want to redirect output to the file **f2**, you end up changing **f3**. This means that when the shell performs output redirection, it opens the file and truncates it, instead of removing the file and creating it anew.

Instead, suppose you do:

```
UNIX> gcc -o f2 ls1.c
UNIX> ls -li f*
34779  -rwxr-xr-x  1 plank       24576 Sep 16 10:16 f2
34778  -rw-r--r--  1 plank          20 Sep 16 10:16 f3
UNIX>
```

You'll note that the c compiler **gcc** did a "**rm f2**" before creating **f2** as an executable.

Note that all directories have at least 2 links:

```
UNIX> mkdir test
UNIX> ls -li | grep test
34800  drwxr-xr-x  2 plank         512 Sep 16 10:17 test
UNIX>
```

This is because every directory contains two subdirectories "**.**" and "**..**" The first is a link to itself, and the second is a link to the parent directory. Thus, there are two links to the directory file "**test**": "**test**" and "**test/.**" Similarly, suppose we make a subdirectory of **test**:

```
UNIX> mkdir test/sub
UNIX> ls -li | grep test
34800  drwxr-xr-x  3 plank         512 Sep 16 10:17 test
UNIX>
```

Now there are three links to "**test**": "**test**", "**test/.**", and "**test/sub/..**"

Besides these links which are automatically created for you, you cannot manually create links to directories. Instead, there is a special kind of a link called a "*soft link*", which you make using the command "**ln -s**". For example, we can create a soft link to the test directory as follows:

```
UNIX> ln -s test test-soft
UNIX> ls -li | grep test
34800  drwxr-xr-x  3 plank         512 Sep 16 10:17 test
34801  lrwxrwxrwx  1 plank           4 Sep 16 10:18 test-soft -> test
```

Note that soft links have a different kind of directory listing. Moreover, note that the creation of a soft link to "**test**" doesn't update the link field of test's inode. That only records regular, or "hard" links.

A soft link is a way of pointing to a file without changing the file's inode. However, soft links can do pretty much everything that hard links can do:

```
UNIX> cat > f1
This is f1
UNIX> ln -s f1 f2
UNIX> cat f2
This is f1
UNIX> cat > f2
This is f2
UNIX> cat f1
This is f2
UNIX> ls -l f*
```

```
-rw-r--r--  1 plank          11 Sep 16 10:19 f1
lrwxrwxrwx  1 plank           2 Sep 16 10:18 f2 -> f1
UNIX> chmod 0600 f2
UNIX> ls -l f*
-rw-------  1 plank          11 Sep 16 10:19 f1
lrwxrwxrwx  1 plank           2 Sep 16 10:18 f2 -> f1
UNIX>
```

What is the main difference between hard and soft links then? Well, for one, if you delete all the hard links to a file, but not all the soft links, then the file still gets deleted.

```
UNIX> rm f1
UNIX> ls -l f*
lrwxrwxrwx  1 plank           2 Sep 16 10:18 f2 -> f1
UNIX> cat f2
cat: f2: No such file or directory
UNIX>
```

The link is called "unresolved"

In unix, you cannot make hard links from a file in one filesystem to a directory in another filesystem. I.e., from your student accounts, you cannot do a command such as:

```
UNIX> ln /blugreen/homes/plank/cs360/notes/Links/lecture.html ~/lecture.html
```

because your home directory is not on the same filesystem as mine. However, you can make a soft link:

```
UNIX> ln -s /blugreen/homes/plank/cs360/notes/Links/lecture.html ~/lecture.html
```