CS 537: Intro to Operating Systems (Fall 2017)

Worksheet 11 - File System

Due: Nov 29nd 2017 (Wed) email Simmi before 11:59 pm

1. Files & Directories

Files and directories provide a basic abstraction of persistent data to users. Here we explore (abstractly) how basic file systems work, focusing on links. Sometimes links lead to pretty odd performance problems.

a. Assume we have a regular file referred by the path name `/a/b/file.txt`. Enumerate the directories sequentially accessed when opening this file.

b. Now we create a **soft link** to this file using `ln -s /a/b/file.txt /a/soft.txt`. Enumerate the directories sequentially accessed when opening `/a/soft.txt`.

c. What happens to `soft.txt` if we rename `file.txt`? (Explain)

d. Let's say we create a symbolic link to a parent directory using `ln -s /a/b /a/b/loop`. List the different pathnames we can use to refer to `file.txt` in the directory `/a/b`.

e. Now we create a **hard link** to this file using `ln /a/b/file.txt /a/hard.txt`. Enumerate the directories sequentially accessed when opening `/a/hard.txt`.

f. What happens to `hard.txt` if we rename `file.txt`? (Explain)

2. File System Implementation

- Assuming a very simple file system that supports 7 operations: **`mkdir()`** : creates a new directory; **`creat()`** : creates a new (empty) file; **`open()`** and **`close()`** : opens and closes a file, respectively; **`write()`** : appends a block to a file; **`link()`** : creates a **hard link** to a file; **`unlink()`** : unlinks a file (removing it if `linkcnt==0`).

- The state of the file system is indicated by the contents of two data structures, i.e., **inodes** and **data**.

- The inodes each have three fields: the first field indicates the type of file (`f` for a regular file, `d` for a directory); the second indicates which data block belongs to a file (here, files can only be empty, which have the address of the data block set to -1, or one block in size, which would have a non-negative address); the third shows the reference count for the file or directory.

  - For example, the following inode is a regular file, which is empty (address field set to -1), and has just one link in the file system: `[f a:-1 r:1]`. If the same file had a block allocated to it (say block 10), it would be shown as follows: `[f a:10 r:1]`. If someone then created a hard link to this inode, it would then become `[f a:10 r:2]`.
  - Note: the reference count of directory here is different from xv6. Here we need to account for the parent ".." as well, e.g. for an empty root directory, the reference count should be 2 because both "." and ".." refer to it.

- Data blocks can either retain user data or directory data.

  - An empty root directory looks like this, assuming the root inode is 0: `[(.,0) (..,0)]`. If we add a single file named `f` to the root directory, which has been allocated inode number 1, the root directory contents would then become: `[(.,0) (..,0) (f,1)]`.
  - If a data block contains user data, it is shown as just a single character within the block, e.g., `[D]`.
  - If it is empty and unallocated, just a pair of empty brackets (`[]`) are shown.

- Assume when allocating a new inode or data block, the first empty inode or block will be used.

Now the initial state of the file system is as follows:

```
inodes [d a:0 r:2] [] [] []; data [(.,0) (..,0)] [] [] []
```

Write down the file system states after the following operations are performed ***sequentially***:

a. `mkdir("/p");`

b. `creat("/p/q");`

c. `link("/p/q", "/r");`

d. `write(open("/r"), "M", BLOCKSIZE);`

e. `unlink("/p/q");`