【OS】Day44(3)

Class	Operating System: Three Easy Pieces
Date	@February 28, 2022

【Ch39】Interlude: Files and Directories

39.9 Getting Information About Files

Beyond file access, we expect the file system to keep a fair amount of information about each file it is storing. We generally call such data about files metadata.

To see the metadata for a certain file, we can use the `stat()` or `fstat()` system calls. These calls take a pathname(or file descriptor) to a file and fill in a stat structres shown below.

```
struct stat {
  dev_t     st_dev;     // ID of device containing file
  ino_t     st_ino;     // inode number
  mode_t    st_mode;    // protection
  nlink_t   st_nlink;   // number of hard links
  uid_t     st_uid;     // user ID of owner
  gid_t     st_gid;     // group ID of owner
  dev_t     st_rdev;    // device ID (if special file)
  off_t     st_size;    // total size, in bytes
  blksize_t st_blksize; // blocksize for filesystem I/O
  blkcnt_t  st_blocks;  // number of blocks allocated
  time_t    st_atime;   // time of last access
  time_t    st_mtime;   // time of last modification
  time_t    st_ctime;   // time of last status change
};
```

Figure 39.5: **The `stat` structure.**

There is a lot of information kept about each file, including its size(in bytes), its low-level name(i.e. inode name), some ownership information, and some information about when the file was accessed or modified.

To see this information, we can use the command line tool `stat`.

```
prompt> echo hello > file
prompt> stat file
  File: 'file'
  Size: 6   Blocks: 8   IO Block: 4096   regular file
Device: 811h/2065d Inode: 67158084    Links: 1
Access: (0640/-rw-r-----) Uid: (30686/remzi)
  Gid: (30686/remzi)
Access: 2011-05-03 15:50:20.157594748 -0500
Modify: 2011-05-03 15:50:20.157594748 -0500
Change: 2011-05-03 15:50:20.157594748 -0500
```

Each file system usually keeps this information in a structure called an inode.

39.10 Removing Files

How do we delete files?

We can just run the program `rm`.

```
prompt> strace rm foo
...
unlink("foo");
...
```

`unlink()` takes the name of the file to be removed, and returns zero upon success.

39.11 Making Directories

We can never write to a directory directly. We can only update a directory indirectly by creating files, directories, or other object types within it. In this way, the file system makes sure that directory contents are as expected.

To create a directory, a single system call, `mkdir()`, is available. The mkdir program can be used to create such a directory.

```
prompt> strace mkdir foo
...
mkdir("foo", 0777)
```

When such a directory is created, it is considered “empty” although it does have a bare minimum of contents. Specifically, an empty directory has two entries: one entry that refers to itself, and one entry that refers to its parent.

The format is referred to as the “.”(dot) directory, and the latter as “..”(dot-dot) .

We can see these directories by passing a flag(`-a`) to the program `ls`:

```
prompt> ls -a
./ ../
```

39.12 Reading Directories

Below is an example program that prints the contents of a directory. The program uses three calls `opendir()`, `readdir()`, and `closedir()`.

```
int main(int argc, char *argv[]) {
  DIR *dp = opendir(".");
  assert(dp != NULL);
  struct dirent *d;
  while((d = readdir(dp)) != NULL) {
    printf("%lu %s\n", (unsigned long) d->d_ino, d->d_dname);
  }
  clsedir(dp);
  return 0;
}
```

The declaration below shows the information available within each directory entry in the struct dirent data structure:

```
struct dirent {
  char d_name[256]; //filename
  ino_t d_ino; //inode number
  off_t d_off; //offset to the next dirent
  unsigned short d_reclen; //length of this record
```

```
  unsigned char d_type; // type of file
};
```

39.13 Deleting Directories

Finally, we can delete a directory with a call to `rmdir()`(which is used by the program of the same name, `rmdir`). Unlike file deletion, removing directories is more dangerous, as we could potentially delete a large amount of data with a single command.

Thus, `rmdir()` has the requirement that the directory be empty(i.e. only has "." and ".." entires) before it is deleted. If we try to delete a non-empty directory, the call to `rmdir()` simply will fail.