File and Directories

Table of Contents

- File and Directories
  - Table of Contents
  - Introduction
  - File Information
  - File Permissions
  - File Systems
  - Directory Operations
  - Device Special Files

Introduction

- Filename
- Pathname
- Directory
  - working directory: where process works
  - home directory: obtained from `/etc/passwd`

File Information

- `stat(2)`: `int stat(const char *path, struct stat *buf);`
- `fstat(2)`: `int fstat(int fd, struct stat *buf);`
- `lstat(2)`: `int lstat(const char *path, struct stat *buf);`
  - return: 0 OK, -1 error
  - `lstat` doesn't dereference a symbolic link
    - it returns the symbolic link itself
  - `fstat` used for an opened file
- Common File Information
  - file types
    - regular, directory
    - block/character special
    - FIFO
    - socket
    - symbolic link
  - permission
  - number of hard links
  - user ID, group ID
  - device number
  - file size
  - block size and number of used blocks
  - timestamps: access, modification, and change

```
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 file system I/O */
  blkcnt_t  st_blocks;  /* number of 512B 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 */
};
```

File Permissions

- UIDs and GIDs
  - real: UID and GID (now)
  - effective: EUID and EGID (permission check)
  - saved set: SUID and SGID
    - saved by `exec` function
- Relationships between UIDs/GIDs
  - normally: EUID = UID, EGID = GID
  - `setuid(2)`: `int setuid(uid_t uid);`
  - `setgid(2)`: `int setgid(gid_t gid);`
    - return: 0 OK, -1 error
    - only root can use them
  - example
    - user *A* owns *P*, and *P* has SUID permission
    - when other executes *P*, *P* runs `setuid` to user `A`
  - passwd permission: `-rwsr-xr-x`
    - lower case `s`: owner can execute
    - upper case `S`: owner cannot execute
- File Access Permissions
  - delete file: need valid write and execute permissions
  - EUID = 0 -> access is allowed
  - EUID = x -> access is allowed by UID x
  - EGID or supplementary GIDs equals group ID -> access is allowed
- Ownership of New Files and Directories
  - UID of new file = EUID of the creating process
  - GID of new file
    - EGID of the creating process or
    - GID of the parent directory
    - depends on OS
      - FreeBSD 5.2.1/Mac OS X 10.3 -> by parent directory

            - Linux: depend on `grpid`
                - `grpid` set or directory has SGID -> by parent directory
                - otherwise -> by the EGID of the creating process
- `access(2)`: `int access(const char *pathname, int mode);`
    - return: 0 OK, -1 error
    - `mode` can be bitwise OR of `R_OK`, `W_OK`, `X_OK`, and `F_OK` (test for existence)
    - **from a real user's perspective**
    - cf. `open` uses EUID, EGID to open file
- `umask(2)`: `mode_t umask(mode_t mask);`
    - return: previous file mode creation mask
    - set file mode creation mask for the process
    - default umask `022`, i.e.
        - `S_IWGRP | S_IWOTH`
        - `000010010`
        - `~022` = `111101101`
    - if `creat` uses `666`, then `666` & `~022` = `644`
- File Modes
    - `int chmod(const char *path, mode_t mode);`
    - `int fchmod(int fd, mode_t mode);`
    - mode
        - `S_ISUID` (4000): set-user-ID
        - `S_ISGID` (2000): set-group-ID
        - `S_ISVTX` (1000): sticky bit
        - `S_IRUSR` (0400): read by owner
        - `S_IWUSR` (0200): write by owner
        - `S_IXUSR` (0100): execute/search by owner
        - `S_IRGRP` (0040): read by group
        - `S_IWGRP` (0020): write by group
        - `S_IXGRP` (0010): execute/search by group
        - `S_IROTH` (0004): read by others
        - `S_IWOTH` (0002): write by others
        - `S_IXOTH` (0001): execute/search by others
- File Ownerships
    - `int chown(const char *path, uid_t owner, gid_t group);`
    - `int fchown(int fd, uid_t owner, gid_t group);`
    - `int lchown(const char *path, uid_t owner, gid_t group)`
        - does not dereference symbolic links
- The Sticky Bit
    - can be used on a executable or a directory
    - executable
        - cache in swap after execution
        - increase performance
    - directory
        - file in this directory can be only deleted or renamed by
            - the user owns the file
            - the user owns the directory

            - the superuser
        - usually set for global accessible directories, such as `/tmp`: `drwxrwxrwt`

File Systems

- Disk Drives, Partitions, and File System
    - disk
- inode
    - describe mata information about a file
        - type
        - permission
        - data blocks
        - timestamps
        - reference counts
        - ...
    - usually positive integer
    - some special number
        - 0: reserved, or does not exist
        - 1: list of bad/deffective bblocks
        - 2: root directory of a partition
    - same inode: hard link
        - e.g. `/home/a`, `/home/a/.` and `/home/a/b/..`
        - # of all hard links of a directory = 2 + # of its subdirectories
            - all of its subdirectories contain `..`
            - the directory itself contains `.`
            - the parent directory of it contains its name
        - parent of root directory is itself
- `int link(const char *existingpath, const char *newpath);`
- `int unlink(const char *pathname);`
- `int remove(const char *pathname);`
- `int rename(const char *oldname, const char *newname);`
    - return: 0 OK, -1 error
- `ls -ls`: show file sizes in 1kB blocks
- `ls -li`: show inode numbers
- Symbolic Links
    - also called soft-links
    - `ln(1)`
    - size of the link is its target's name
    - able to point to an nonexistence file
    - various functions: follow means it will dereference the lnk

Function	Not Follow Link	Follow Link
access		•
chdir		•
chmod		•

Function	Not Follow Link	Follow Link
chown		•
creat		•
exec		•
lchown	•	
link		•
lstat	•	
open		•
opendir		•
pathconf		•
readlink	•	
remove	•	
rename	•	
stat		•
truncate		•
unlnink	•	

- `int symlink(const char *actualpath, const char *sympath);`
  - return: 0 OK, -1 error
  - `sympath` and `actualpath` need not reside in the same file system
- `ssize_t readlink(const char *path, char *buf, size_t bufsize);`
  - return: number of bytes placed in the buffer, -1 error
  - no null character at the end of the buffer

- File Times
  - `st_atime`: last access time, `ls -lu`
  - `st_mtime`: last modification time, `ls -l` default
  - `st_ctime`: last change times, `ls -lc`
  - effect of various functions on times
    - r: referenced file or dir
    - p: parent directory
    - a: access
    - m: modify
    - c: change
    -

Function	ra	rm	rc	pa	rm	pc
[f]chmod			•			
[f]chown			•			
creat(new)	•	•	•		•	•

Function	ra	rm	rc	pa	rm	pc
creat(trunc)		•	•			
exec	•					
lchown			•			
link			•		•	•
mkdir	•	•	•		•	•
mkfifo	•	•	•		•	•
open(new)	•	•	•		•	•
open(trunc)		•	•			
pipe	•	•	•			
read	•					
remove(file)			•		•	•
remove(dir)					•	•
rename			•		•	•
rmdir					•	•
[f]truncate		•	•			
unlink			•		•	•
utime	•	•	•			
write		•	•			

- `int utime(const char *filename, const struct utimbuf *times);`
  - return: 0 OK, -1 error
  - `utimbuf` data structure

    ```
    struct utimbuf{
      time_t actime;
      time_t modtime;
    };
    ```

  - chagne access time and modification time
  - if `times` is `NULL`, the access time and modification time is set to the current time

Directory Operations

- `int mkdir(const char *pathname, mode_t mode);`

- `int rmdir(const char *pathname);`
  - return: 0 OK, -1 error
  - directory is empty + a process has opend the directory
    - after `rmdir`, the directory is removed but not freed
    - no new file can be created in the to-be-removed directory
    - it is freed when the process close the directory
- `DIR *opendir(const char *name);`
  - return: pointer to the directory if OK, `NULL` error
- `struct dirent *readdir(DIR *dir);`
  - return: pointer to a dirent structure if OK, `NULL` reaching EOF or error
  - ```
    struct dirent {
      ino_t d_ino;  /* inode */
      off_t d_off;  /* offset to the next dirent */
      unsigned short d_reclen;  /* length of this record */
      unsigned char d_type;  /* type of file */
      char d_name[256];  /* filename */
    };
    ```
- `int closedir(DIR *dir);`
  - return: 0 OK, -1 error
- `void rewinddir(DIR *dir);`
  - resets the position of the directory stream
- `off_t telldir(DIR *dir);`
  - return: current position of the opened dir
- `void seekdir(DIR *dir, off_t offset);`
  - sets the position of the directory stream
- `int chdir(const char *path);`
- `int fchdir(int fd);`
  - return: 0 OK, -1 error
- `char *getcwd(char *buf, size_t size);`
  - return: `buf` if OK, `NULL` error
- default working directory is configured in `/etc/passwd`

Device Special Files
    ```

- major number: the device driver
  - extract by macro `major(dev_t)`
- minor number: the specific sub device
  - extract by macro `minor(dev_t)`
- every file has a `st_dev` number, i.e. container file system
- device files have `st_rdev` numbers, i.e. device/sub-device
- `/dev` stores device special files
  - real file system: created with `mknod(1)`
  - pseudo file system: automatically generated when a device driver is registered
- code example

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>
int main(int argc, char *argv[]) {
  int i;
  struct stat buf;
  for (i = 1; i < argc; i++) {
    printf("%s: ", argv[i]);
    if (stat(argv[i], &buf) < 0) {
      printf("stat error");
      continue;
    }
    printf("dev=%d/%d", major(buf.st_dev), minor(buf.st_dev));
    if (S_ISCHR(buf.st_mode) || S_ISBLK(buf.st_mode)) {
      printf(" (%s) rdev = %d/%d",
             (S_ISCHR(buf.st_mode)) ? "character" : "block",
major(buf.st_rdev),
             minor(buf.st_rdev));
    }
    printf("\n");
  }
  return 0;
}
```

- output example
  - `./a.out / /dev /home/ee904 /dev/tty0 /dev/sda2`

```
/: dev=8/19
/dev: dev=0/6
/home/ee904: dev=8/20
```

```
/dev/tty0: dev=0/6 (character) rdev = 4/0
/dev/sda2: dev=0/6 (block) rdev = 8/2
```

- common devices

```
/dev/hdaN? - IDE disks and partitions
/dev/sdaN? - SCSI or SATA disks and partitions
/dev/scdN - CD/DVD-ROMs
/dev/ttyN - terminals
/dev/ttySN - COM ports
/dev/pts/N - pseudo terminals
/dev/null
/dev/zero
/dev/random
```