Chapter 4 - Files and Directories

Information about files and directories
Management of files and directories

File Systems

- Unix File Systems
  - UFS - original FS
  - FFS - Berkeley
  - ext/ext2/ext3/ext4 - Linux
  - Many others similar to UFS ...
- Others -- possibly available on UNIX (Linux & *BSD)
  - FAT - DOS, smaller thumb drives
  - NTFS - Windows NT +
  - HFS - Apple (Very old)
  - NFS - Sun (now Oracle)
  - AFS/Coda - CMU
  - ZFS - Sun (now Oracle) (Zettabyte File System,1024^7 or 2^70)
  - torrent/dropbox/.... -- "cloud file systems"
  - many others (https://en.wikipedia.org/wiki/List_of_file_systems)

UNIX file system - design

File Types

- Regular File
- Directory File
- Character Special File
- Block Special File
- Symbolic Link
- FIFO
- Socket (Network)

UNIX file system - layout

Disk: | partition 1 | partition 2 | ..... |

Single partition (some machines)

| file system 1 | file system 2 | file system 3 | ... |

Single File System

| Boot | Super | inodes | data blocks |

- Boot - bootstrap program
- Super - contains information about partition
- inodes - One per real file
- data blocks - both files and directories ...
- File system blocks: usually power of 2, 1k to 8k
- Each section -- integral number of file system blocks

UNIX file system

Directory:

| Name, inode # | Name, inode # | ..... |

- Each entry in a directory is a "link"
- Inode contains number of links
- File (inode & data) is not deleted until link count is 0
- Original Unix FS, name limited to 14 characters
- Berkeley FFS, <sys/dirent.h> MAXNAMLEN -- 255

Inode

- owner, group, permissions
- file type (reg, dir, sym link, ...)
- number of links
- size, number of blocks (different!)
- times (accessed, modified, status changed)
- Access to data blocks
  - n data block pointers (disk address)
    - inode -> data block
  - 1 - indirect block
    - inode -> pointer block -> data block
  - 1 - 2 level indirect block
    - inode -> pointer block -> pointer block -> data block
  - 1 - 3 level indirect block
    - inode -> ptr block -> ptr block -> ptr block -> data block

NetBSD: 32 bit block address, 8K blocks, 2048 pointers/block, 12 direct

12 + 2048 + 2048 ^ 2 + 2048 ^ 3 = 8,594,130,956 blocks

8,594,130,956 * 8 * 1024 = 70,403,120,791,552 bytes per file

information about files/dirs

system calls - stat(2), fstat(2), lstat(2), (stat(1), stat.c)

- int stat(const char *path, struct stat *sb)
- int lstat(const char *path, struct stat *sb)
- int fstat(int fd, struct stat *sb)

```
struct stat {  /* NetBSD version */
    dev_t     st_dev;          /* device containing the file */
    ino_t     st_ino;          /* file's serial number */
    mode_t    st_mode;         /* file's mode (protection and type) */
    nlink_t   st_nlink;        /* number of hard links to the file */
    uid_t     st_uid;          /* user-id of owner */
    gid_t     st_gid;          /* group-id of owner */
    dev_t     st_rdev;         /* device type, for device special file */
    struct timespec st_atimespec;       /* time of last access */
    struct timespec st_mtimespec;       /* time of last data modification */
    struct timespec st_ctimespec;       /* time of last file status change */
    off_t     st_size;         /* file size, in bytes */
    int64_t   st_blocks;       /* # of 512 byte blocks allocated for file */
    u_int32_t st_blksize;      /* optimal file sys I/O ops blocksize */
    u_int32_t st_flags;        /* user defined flags for file */
    u_int32_t st_gen;          /* file generation number */
};
```

stat calls information

- Macros for file type
  - S_ISREG(st_mode)
  - S_ISDIR(st_mode)
  - S_ISCHR(st_mode)
  - S_ISBLK(st_mode)
  - S_ISFIFO(st_mode)
  - S_ISLNK(st_mode)
  - S_ISSOCK(st_mode)

- User, Group, Other protection bits in st_mode

- Extra special protection bits in st_mode
  - Set User ID
  - Set Group ID
  - Sticky bit

- stat.c program -- blocks vs file size

Other system calls

- int access(const char *path, int mode)
  - R_OK, W_OK, X_OK, F_OK

- change modes
  - int chmod(const char *path, mode_t mode)
  - int lchmod(const char *path, mode_t mode)
  - int fchmod(int fd, mode_t mode)
  - chmod(1) -- changes setuid/setgid/sticky bits

- Change owner
  - int chown(const char *path, uid_t owner, gid_t group);
  - int lchown(const char *path, uid_t owner, gid_t group);
  - int fchown(int fd, uid_t owner, gid_t group);

Other system calls (page 2)

- truncate a file
  - int truncate(const char *path, off_t length)
  - int ftruncate(int fd, off_t length)

- Add a link to a file
  - int link(const char *oldname, const char *newname)
  - Works only on files
  - On the same filesystem
  - need write permission to last directory in newname
  - Adds a directory entry
  - Does not double file storage needs
  - ln(1), link(1)

Other system calls (page 3)

- unlink a file name
  - int unlink(const char *path)
  - Must have write and "execute" access to directory
  - Sticky bit for directory:
    - Off -> do not have to own the file
    - On -> must own file
  - deletes entry in directory
  - file is deleted when:
    - link count is zero
    - file is not open
  - remove(3) -- alias for unlink
  - rm(1) -- command line access

Other system calls (page 4)

- Rename a file
  - int rename(const char *oldname, const char *newname)
  - oldname and newname must be on same filesystem
  - File
    - newname can not be an existing directory
    - if newname exists and is a file, it is unlinked
    - must have write permission to both dirs
  - Directory
    - if newname exists and is empty, it is unlinked
    - if newname exists and is not empty, error
    - newname can not be a subdirectory of oldname
  - If unlinking a directory or a file, must have permission
  - mv(1) -- command line access
    - will copy files from one file system to another

Other system calls (page 5)

- Symbolic links
  - int symlink(const char *name1, const char *name2)
  - not a "hard link"
  - name2 is new entry
  - name1 is stored for use later
  - name1 and name2 do not have to be on same file system
  - name1 does not have to exist!
  - unlink removes only stored name

  - int readlink(const char *path, char *buf, size_t bufsiz)
  - readlink(1)

Other system calls (page 6)

- Times ...
  - int utimes(const char *path, const struct timeval *times)
  - int lutimes(const char *path, const struct timeval *times)
  - int futimes(int fd, const struct timeval *times)
  - Sets access and modification times.
  - times:
    - NULL -> set to current time
    - Non NULL -> points to a 2 element array,
      - access time
      - modification time

- Use?
  - e-mail -- modified time after access time
  - tar(1) -- "tape archive"
  - Backup / restore

Other system calls (page 7)

- Making directories
  - int mkdir(const char *path, mode_t mode)
  - must have write access to create
  - mkdir(1)
    - mkdir -p /full/new/path/you/want

- Deleting directories
  - int rmdir(const char *path)
  - directory must be empty (only . and ..)
  - must have write access to parent directory
  - rmdir(1)
  - rm -r tree

Other system calls (page 8)

- working directories
  - int chdir(const char *path)
  - int fchdir(int fd)
  - char * getcwd(char *buf, size_t size) /* Library call */

Reading directories

Early UNIX -- open, read, close

- Had to know format of directory
- Different code for different file systems

Now -- library of routines, supplied by each OS

- DIR *opendir(const char *filename)
- struct dirent *readdir(DIR *dirp)
- int closedir(DIR *dirp)
- long telldir(const DIR *dirp)
- void seekdir(DIR *dirp, long loc)
- void rewinddir(DIR *dirp)
- int dirfd(DIR *dirp)

- Works for any file system
- Do not need to know directory format

struct dirent

```
struct dirent {  /* NetBSD version */
    u_long  d_fileno;   /* file number of entry aka d_ino*/
    u_short d_reclen;   /* length of this record */
    u_short d_namlen;   /* length of string in d_name */
    char    d_name[MAXNAMLEN + 1]; /* maximum name length */
};
```

POSIX specifies only d_ino and d_name.

Simple ls program (ls.c)

Other system calls (page 9)

- File system sync
  - void sync(void)

- Single file sync
  - int fsync(int fd)