ITP 30002 Operating System

Files and Directories

Chapters 39

Shin Hong

Two Abstractions for Persistent Storage

- A **file** is a linear array of bytes, each of which can be read or write
  - each file has a low-level identifier often called **inode number**
  - the structure inside a file is managed by the application program, but the file's persistency is managed by the OS

- A **directory** is a list of names of the contained files and directories
  - a name is a pair of user-readable one and low-level one
  - a directory is named by an inode number

- The directory hierarchy is formed as a directory exists inside another one
  - the hierarchy starts at a root directory (i.e., /)
  - the **file location** is defined by the concatenation of all directory names from the root down to the immediate container and the file name

File System Interface – File Access (1)

- Open a File (i.e., `open()`)
  - create a new file, open an existing one, truncate a file to a size of zero
  - return a file descriptor which is a non-negative integer private per process, works as a pointer to a file object

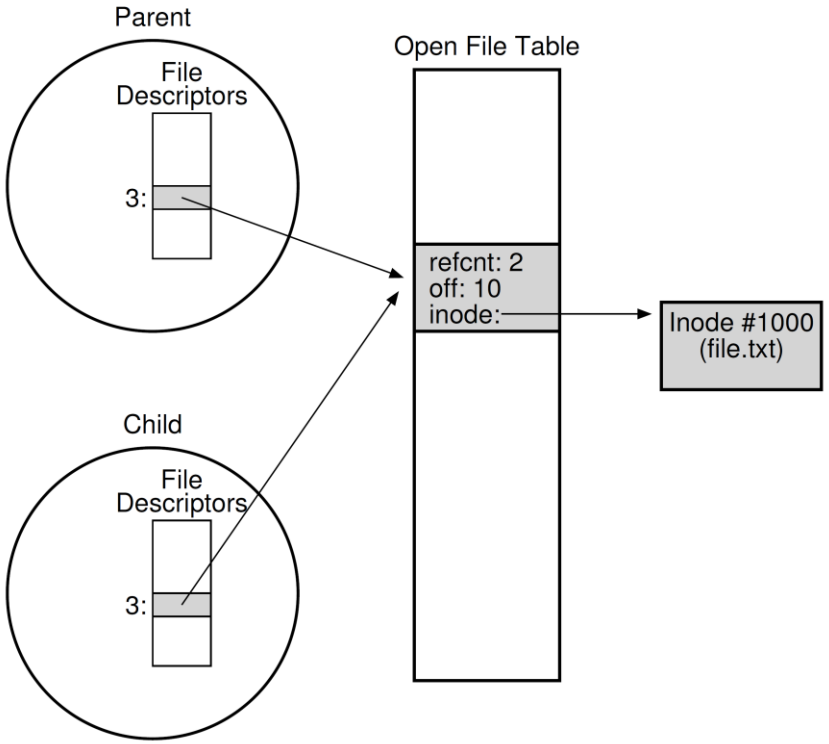
- Read or write a file sequentially
  - example:

```
prompt> strace cat foo
...
open("foo", O_RDONLY|O_LARGEFILE)          = 3
read(3, "hello\n", 4096)                   = 6
write(1, "hello\n", 6)                     = 6
hello
read(3, "", 4096)                          = 0
close(3)                                   = 0
...
prompt>
```

File Access (2)

- Change the current offset of a file (i.e., `lseek()`)
  - once a file is opened, the OS tracks the current offset to determine which offset will be read or written in the next
  - `lseek` is to explicitly move a file offset to a specific point within a file to read or write to random offsets

- Share file table entries
  - a process has a unique entry in the open file table as it opens and reads/writes a file
  - a child processes created by `fork()` shares the same open file entry with its parent process
  - `dup()` allows a process to create a new file descriptor that refers to the same underlying open file as an existing file descriptor



File Access (3)

- Flush (i.e., `fsync()`)
  - enforce the file system to write the buffered data of the given file descriptor to the persistent storage

- Get information about files (i.e., `stat()`, `fstat()`)
  - the metadata for a file contains the size of a file, the low-level name, ownership information, when the file was accessed or modified, etc.
  - the metadata of a file is stored in an inode structure.

```
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
};
```

Directory

- Create an empty directory (i.e., mkdir())
  - an empty directory contains two entries (i.e., . and ..) by default.
  - each directory entry, which works as a link to a file/directory, is structured as follows:

```
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
};
```

- Create a new directory entry from an existing one (i.e., ln())
  - create a new file name (i.e., hard link) to refer the inode of the same existing file
    - Example.

```
prompt> ls -i file file2
67158084 file
67158084 file2
```

  - a soft link is a special file that points to another file name
- Delete a file from a directory
  - unlink the corresponding link from a directory (i.e., the reference count is decreased)
  - the file system will eventually reclaim if an inode has zero count

Permission Bits and Access Control

- Unlike process and virtual memory, file systems are basically shared among multiple users, thus they require more ways to control permissions

- Permission bit
  - the owner of a file can update the permission
  - (owner, group, others) X (read, write, execute)
  - for a directory, an execute bit allows the user to change directory into the one
  - example:

```
prompt> ls -l foo.txt
-rw-r--r--  1 remzi wheel  0 Aug 24 16:29 foo.txt
```

- Access control list
  - specify what kinds of rights a specific user has on specific files

Making and Mounting A File System

- Make a file system
  - give a file system creating tool (e.g., mkfs), as input, a device and a file system type.

- Mount a file system
  - take an existing directory as a target **mount point** and paste a new file system onto the directory free at that point
  - example

```
prompt> mount -t ext3 /dev/sda1 /home/users
prompt> ls /home/users/
a b
```