Files and Directories

What is the filesystem API?

By Matthew Tancreti
For COM S 352
Iowa State University

Files and Directories

File systems provide **persistent storage**

Users have expectation that persistent data is keep intact despite potential system crashes or power outages

Users also want to be able to organize, search and secure data

What is the file system API?

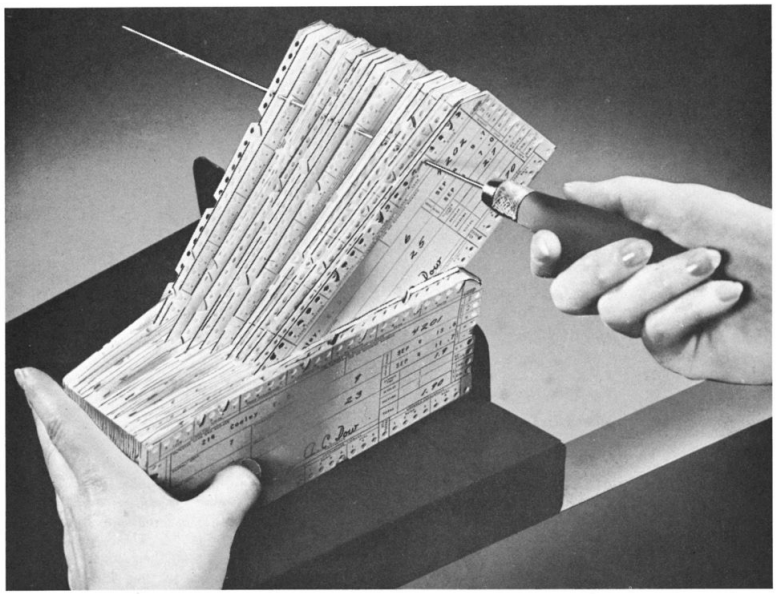


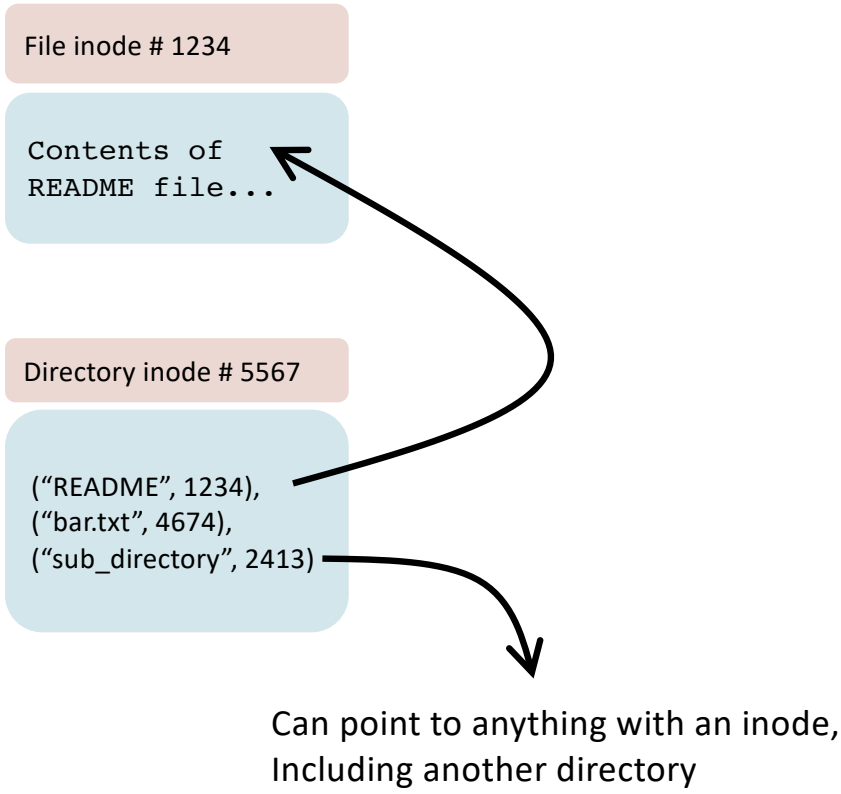
FIG. 5-1 Card selection with an edge-notched card system.

Edge-notched cards [source]

Files and Directories

File – an array of bytes given an identifier (**inode number** in Unix)

Directory – a list of (user-readable name, inode number) pairs that is also given and inode



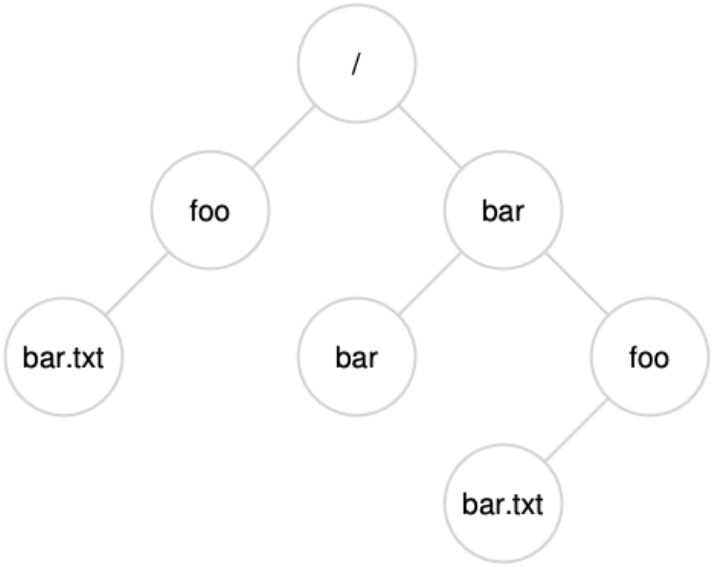
Directory Tree

Directory points to files or other directories resulting in a **directory tree**

The head of the tree is the **root directory**, often referred to as **/**

Same / separator is used to name subsequent directories

We can name a file by its **absolute path** starting at root, for example: **/bar/foo/bar.txt**



Creating File (POSIX API)

```
int fd = open("foo", O_CREAT|O_WRONLY|O_TRUNC,
                     S_IRUSR|S_IWUSR);
```

Open system call takes a file name and options. The file will be opened in the **current working directory**.

Options:

- O_CREAT – create the file if it does not exist
- O_WRONLY – only allow writing to the file, not reading
- O_TRUNC – zero out the contents of the file

Returns a **file descriptor** – an identifier the process uses to reference a resource when making system calls (read(), write(), etc.). Think of file descriptor as pointer to an object that stores information about an open file.

Open File Descriptors

Information about open files are stored in the Process Control Block (PCB)

In xv6, the file descriptor is used as an index to an array of open files, the array is stored in the process structure (i.e., PCB)

```
struct proc {
  ...
  struct file *ofile[NOFILE]; // Open files
  ...
};
```

Tracing System Calls of Example Program

```
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>
```

Sequential vs Random Read/Writing

By default, the first read/write begins at byte 0 of the file. Every read/write after that starts at the next byte following the previous read/write.

Change the location of the next read/write with lseek.

```
off_t lseek(int fd, off_t offset, int whence);
```

Whence describes how to apply the offset

- If whence is SEEK_SET, the offset is set to offset bytes.
- If whence is SEEK_CUR, the offset is set to its current location plus offset bytes.
- If whence is SEEK_END, the offset is set to the size of the file plus offset bytes.

Support for Radom Access

Process Control Block (PCB)

```
struct proc {
  ...
  struct file *ofile[NOFILE]; // Open files
  ...
};
```

Open file

```
struct file {
  int ref;
  char readable;
  char writable;
  struct inode *ip;
  uint off;
};
```

offset for next read/write

Support for Concurrent Access

What if multiple processes access file at same time? Race condition?

Each process has a list of pointers to open files in its PCB

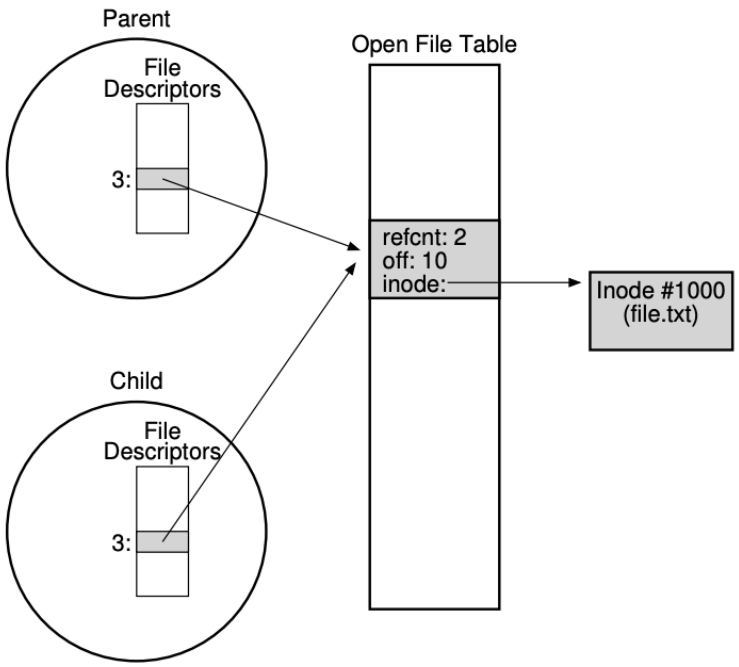
The kernel also keeps a list of all files open by processes

xv6 uses this list of open files to place a lock such that only one process can be accessing a file at one time

```
struct {
  struct spinlock lock;
  struct file file[NFILE];
} ftable;
```

Shared File Table Entries

A child inherits its parents open file descriptors from the fork

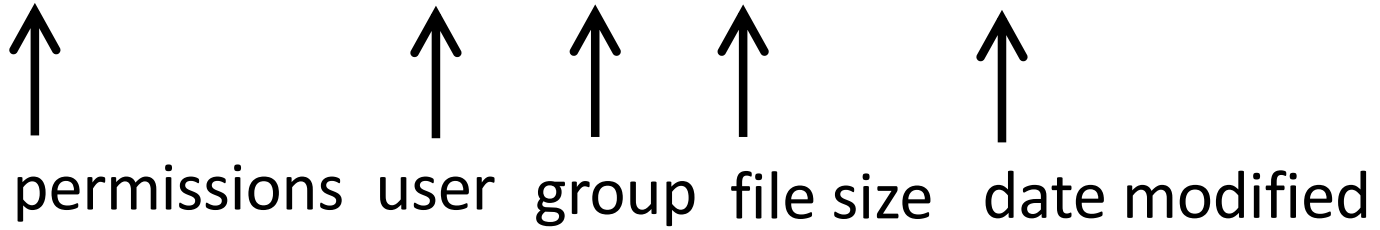


Permission Bits and Access Control

Files are owned by a user and a group

Permissions for read, write and execute are specified for the user, group and all other users

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



Command Line

Useful Linux commands for files and directories

mv – rename a file

rm – remove a file

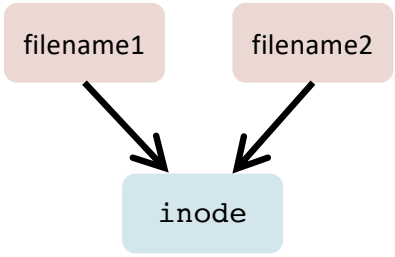
mkdir – make a directory

ls – list the contents of a directory

rmdir – remove a directory

Hard Links vs Soft Links

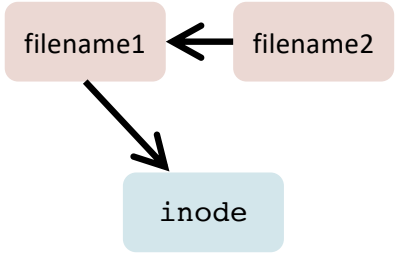
Hard link connects a filename to an inode



`ln filename1 filename2`

make filename2 will point to the same inode as filename1

Soft (symbolic) link gives an alias to a name



`ln –s filename1 filename2`

Make filename2 another name for filename1