CSci 4061
Introduction to Operating Systems

File Systems: Directories
Chapter 5

Directory

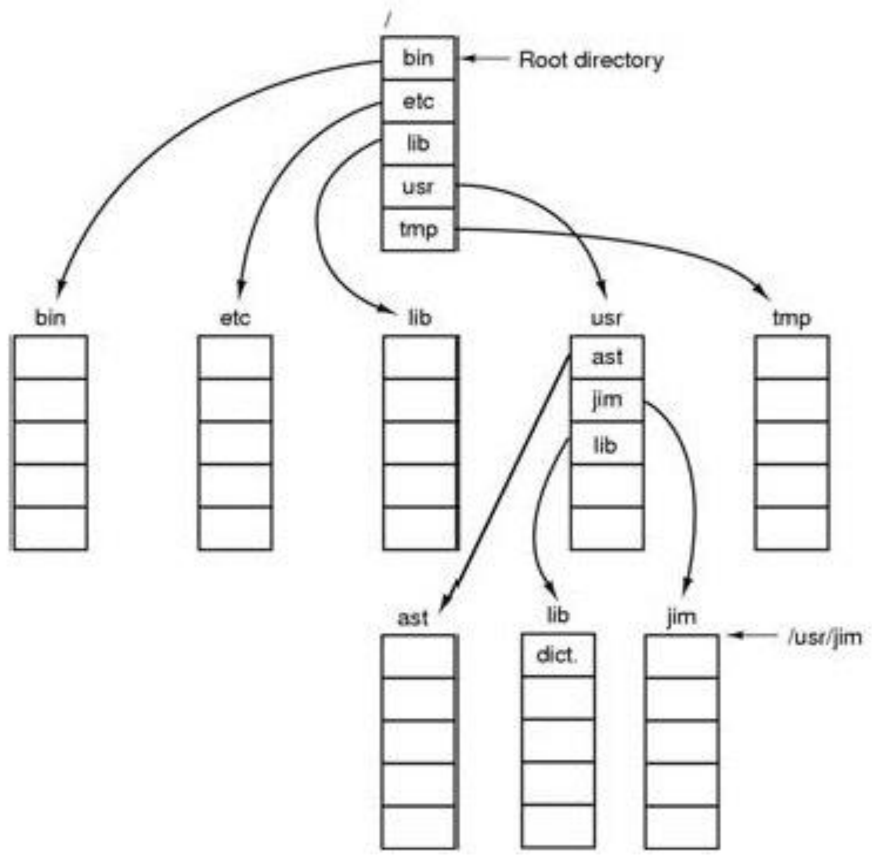
- What is it?

Directory

- Abstraction
  - Container for related files (and other directories)

  - name
  - location
  - contents
  - attributes
  - persistent

Unix Path Names



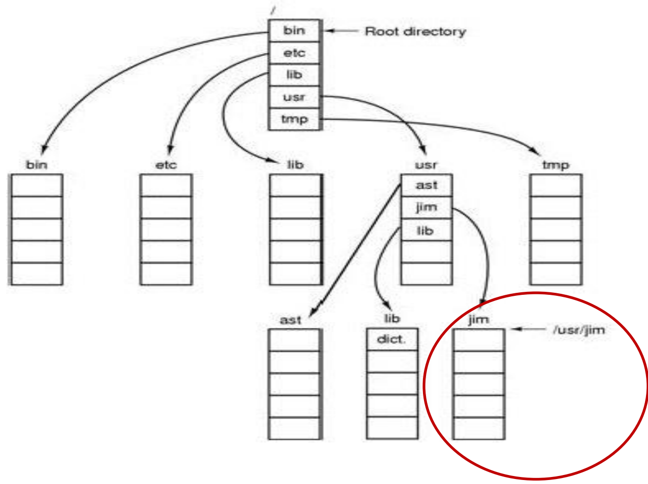
A Unix directory tree
MAX_PATH: 1024 chars

Path Names (cont'd)

- Home directory: dir you are logged into (~)
- Current working directory (`cwd`): `cd /usr/jim`
  - shell> `pwd`
  - shell> `/usr/jim`
- Relative file names(w/r to `cwd`)
  - shell> `ls foo`

  … advantage?
- Absolute file names (rooted from /)
  - Shell> `ls /usr/jim/foo`

  … advantage?



Path Names (cont’d)

- Value of absolute path names:

```
foo.c/foo

    …
    f = fopen (“bar”, “r”);
    …
```

bill> `/usr/jon/foo` will fail unless “bar” is in `cwd`

“bar” must be in the `cwd` of whomever runs it, instead:

```
f = fopen (“/user/jon/bar”, “r”);
```

bill> `/usr/jon/foo` works now

On the other hand, if we were distributing `foo` …

Path Names (cont'd)

```
int chdir (const char *path);
      ↕
shell> cd foo
```

Ex:

```
    fd1 = open (“/usr/ben/abs”, O_RDONLY);
    chdir (“/usr/ben”);
    fd1 = open (“abs”, O_RDONLY);
```

```
char *getcwd (char *name,
      ↕          size_t size);
shell> pwd
```

Links: files with multiple names

Each name is an alias or a "hard link"

```
#include <unistd.h>
int link (const char *original_path,
           const char *new_path)
```

cannot exist as a file already

```
link ("foo", "bar");  // "bar" refers to file "foo"
unlink ("bar");
```

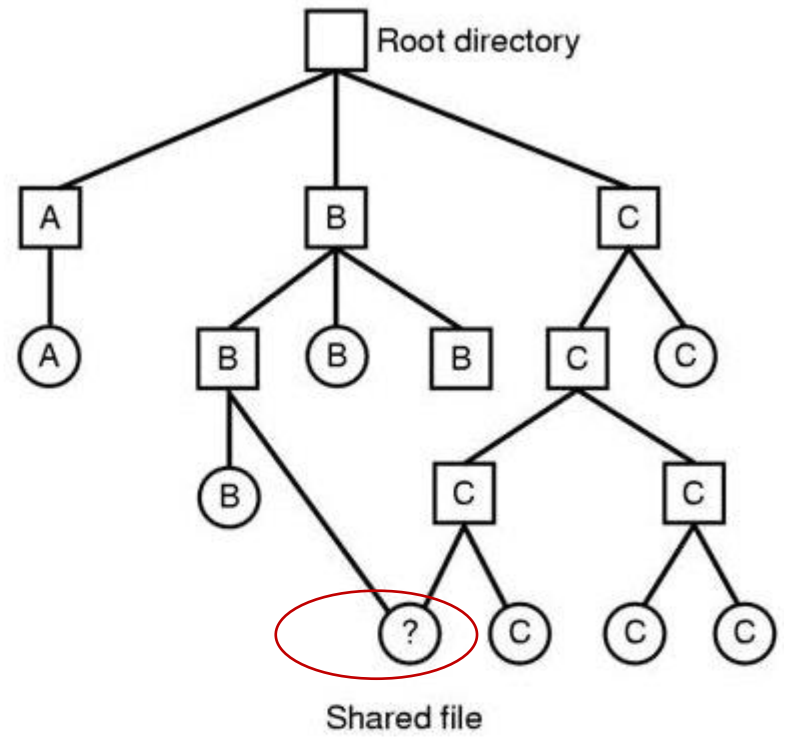
Number of links is the link count

Last `unlink` will delete the file (when no fd's to it, i.e. open)

Cannot `unlink` a directory

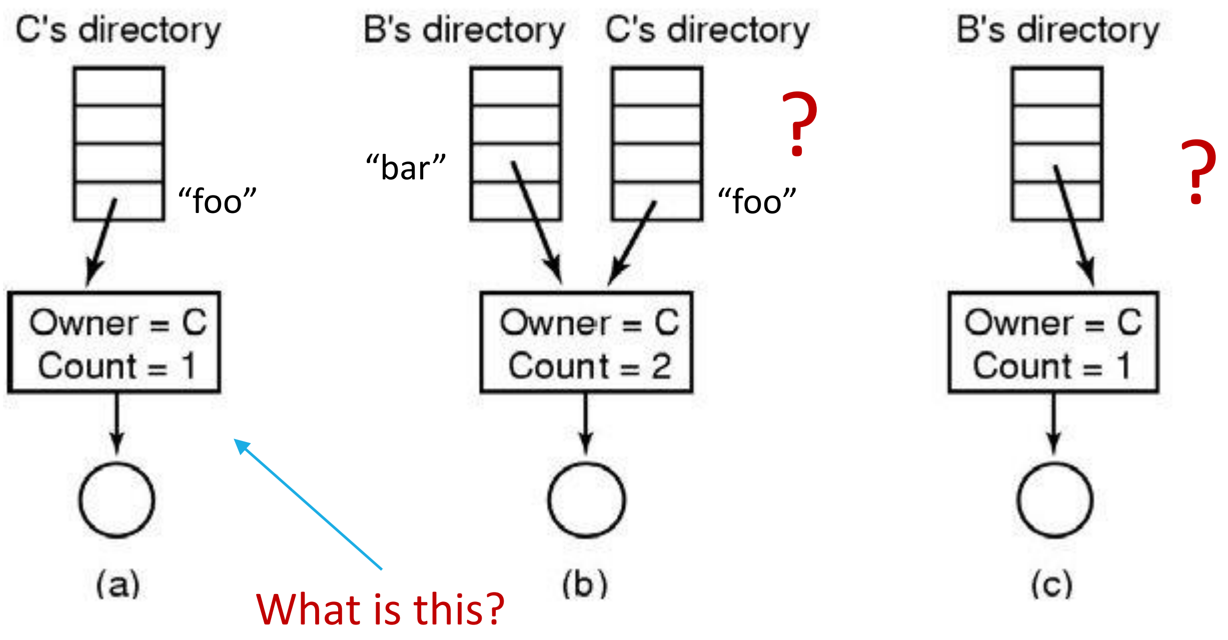
Does link affect the fd table?

Directories and Hard Links



File system containing a shared file

Directories and Hard Links (cont'd)



a) Situation prior is linking
b) After the link is created
c) After the original owner unlinks/"removes" the file

Symbolic Links

- Hard links cannot be made to directories or to files in other file systems
- Symbolic link: allows a file/dir name to “point to” another file/dir name

```
int symlink (const char *realname,
                 const char *symname);
symlink (“/usr/jon/tmp1”,
           “/usr/bill/tmp2”);
```

New inode created for symname `/usr/bill/tmp2`

Symbolic Links (cont’d)

```
symlink (“/user/jon/tmp1/f1”, “f2”);
```

```
        lrwxrwxrwx f2 -> /usr/jon/tmp1/f1
```

Remove `f2`, symbolic link goes away, file does not

Remove `/user/jon/tmp1/f1`, symbolic link remains!

Default Hard Links

- Two links: . and .. (ls -id <dir>)
  - `.` Refers to `cwd`
  - `..` Refers to one level up from `cwd`
  - shell> `./cat`

  - shell> `cd /usr/jim/tmp`
  - shell> `ls ./foo`
    - same as `foo` or `/usr/jim/tmp/foo`
  - shell> `ls ../bar`
    - same as `/usr/jim/bar`

Link to directories!
Special case

Directory Permissions

- Directories are themselves represented by files
    - Have a name
    - Contents are file names
    - Same protection bits are used for directories (rwx)

Directory Permissions (cont'd)

- Read means class of users can list '`ls`' contents of directory
- Write means class of users can create or remove files in the directory
- Execute means class of users can '`cd`' into directory also allows open and execute for files in the directory

Directory Operations

- create/remove
- opendir/closedir
- readdir

Create

```
#include <sys/stat.h>


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


mkdir (“tmp/dir1”, 0777);
```

Also places two links (. and .. in directory)

Remove

```
int rmdir (const char *pathname);
```

Removes the directory: directory must be empty!

Can be executed in the shell as well

```
shell> rmdir foo
```

Open/Close Directory

- Open a directory to look at its contents

```
#include <dirent.h>

DIR *opendir (const char *dirname);
struct dirent *readdir (DIR *dirptr);
int closedir (DIR *dirptr);

DIR *dp;
dp = opendir ("/tmp/dir1");

struct dirent {
      ino_t d_ino;
      char d_name[NAMESIZE];
}
```

readdir

```
#include <dirent.h>
struct dirent *readdir
               (DIR *dirptr);
```

Returns each directory entry, NULL at the end

readdir: example

Example: (very simple) `my_ls`

```
int my_ls (const char *name) {
    struct dirtent *d;
    DIR *dp;



    closedir (dp);
    return 1;
}
```