15-410

"...RADIX-50??..."

File System (Interface)
Oct. 31, 2008

Dave Eckhardt

Roger Dannenberg

Contributions from

- **Rahul Iyer**

Synchronization

Checkpoint #3

- See Tuesday's posting
- Deadline: 23:59
- Opportunity to do a conscious planning cycle
  - Fill out STATUS file
  - Later see if you were right

Today

- Chapter 10, File system interface
  - Ok to skip: remote/distributed (10.5.2!!)

Also read Chapter 13

- Might help demystify readline() some

Synchronization

Two interesting papers about disks

- **http://www.seagate.com/content/docs/pdf/whitepaper/D2c_More_than_Interface_ATA_vs_SCSI_042003.pdf**
- **Google for “200 ways to revive a hard drive”**

What's a file?

Abstraction of *persistent storage*

- **Hide details of storage devices**
  - **sector addressing: CHS vs. LBA**
  - **SCSI vs. IDE**
- **Hide details of allocation/location on a storage device**

***Logical* grouping of data**

- **May be *physically* scattered**

Programs, data

Some internal structure

Typical file attributes

Name – 14? 8.3? 255?

- Unicode? ASCII? 6-bit? RADIX-50?

Identifier - “file number”

Type (or not)

Location – device, block list

Size – real or otherwise

Protection – Who can do what?

Time, date, last modifier – monitoring, curiousity

“Extended” file attributes

BSD Unix

- archived
- nodump
- append-only (by user/by operating system)
- immutable (by user/by operating system)

MacOS

- icon color

Plan 9

- Identity of most recent mutator

Operations on Files

Create –locate space, enter into directory

Write, Read –according to position pointer/cursor

Seek –adjust position pointer

Delete –remove from directory, release space

Truncate

- Trim data from end
- Often all of it

Append, Rename

I/O to a File –Take 1

Users will read/write files

- **Not being able to defies the point in having them**

So, how do you read from and write to one?

- **read("README.dox", input_buffer, num_bytes);**
  - **What's the problem with this?**

I/O to a File –Take 1

Users will read/write files

- Not being able to defies the point in having them

So, how do you read from and write to one?

- read(“README.dox”, input_buffer, num_bytes);
  - What's the problem with this?
- read(“README.dox”, input_buffer, num_bytes, start_loc);
  - What's the problem with this?

What's the solution?

I/O to a File –Take 2: Open-file State

Expensive to specify name for each read()/write()

- String-based operation
- Directory look-up

Add an open() operation

- Adds “state”

“Open-file” structure stores:

- File-system / partition
- File-system-relative file number
- Read vs. write
- Cursor position

Something still missing?

Open files (Unix Model)

"In-core" **file state**

- Mirror of on-disk structure
  - File number, size, permissions, modification time, ...
- Housekeeping info
  - Back pointer to enclosing file system
  - Pointer to disk device hosting the file
  - Who holds locks on ranges of file
- How to access file (vector of methods)
- Pointer to file's type-specific data

Shared **when file is opened multiple times**

Open files (Unix Model)

"Open file" **state (result of one open() call)**

- Access mode (read vs. write, auto-append, ...)
- Credentials of process (when it opened the file)
- Cursor position
- Pointer to underlying "open file"

Shared **by multiple processes**

- "copied" by fork()
- inherited across exec()

Example

```
int fd1, fd2, fd3;
off_t pos2, pos3;
char buf[10];

fd1 = open(“foo.c”, O_RDONLY, 0);
fd2 = dup(fd1);
fd3 = open(“foo.c”, O_RDONLY, 0);
read(fd1, &buf, sizeof (buf));

pos2 = lseek(fd2, 0L, SEEK_CUR); /* ⇒? */
pos3 = lseek(fd3, 0L, SEEK_CUR); /* ⇒? */
```

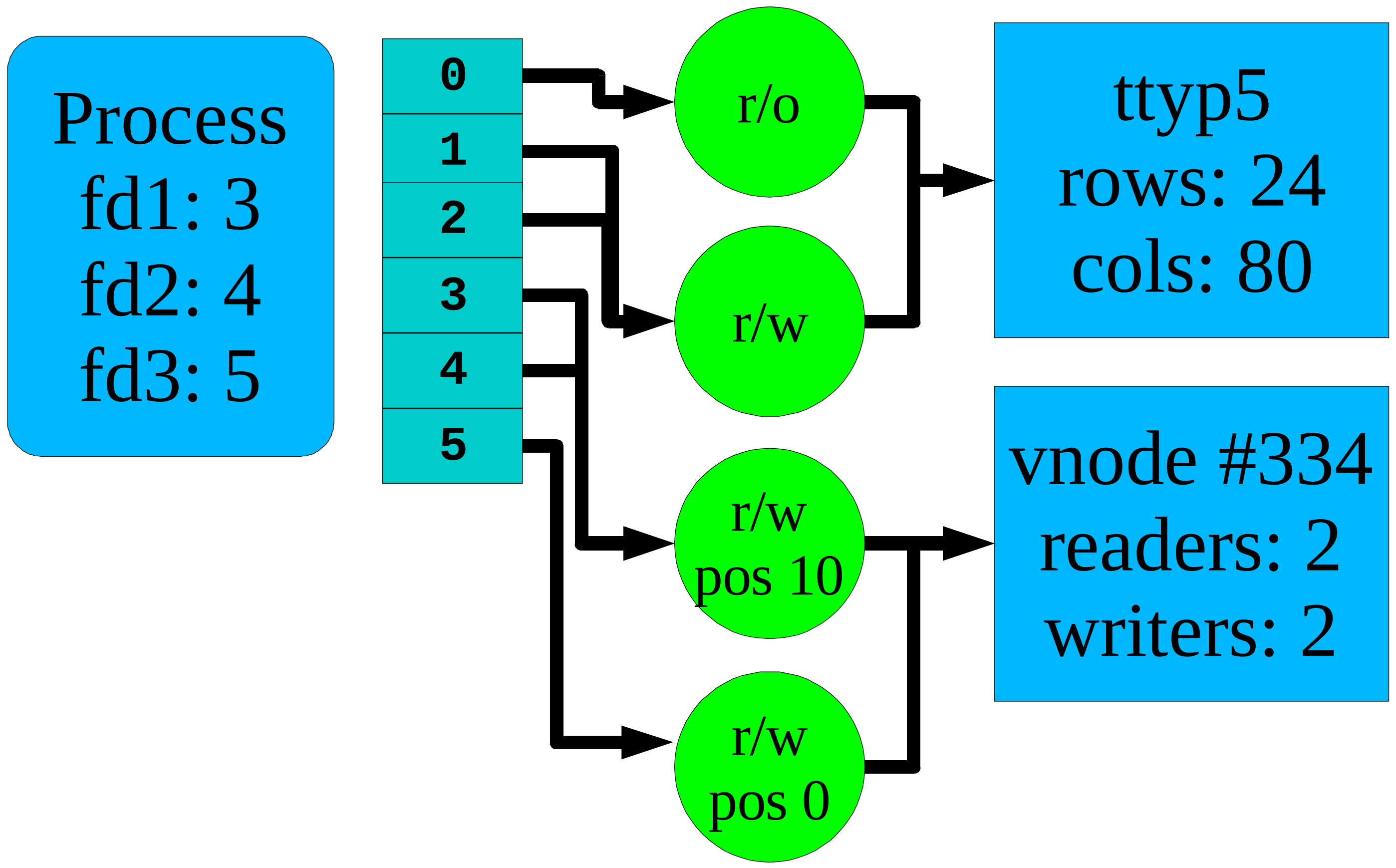
Example

```
int fd1, fd2, fd3;
off_t pos2, pos3;
char buf[10];

fd1 = open(“foo.c”, O_RDONLY, 0);
fd2 = dup(fd1);
fd3 = open(“foo.c”, O_RDONLY, 0);
read(fd1, &buf, sizeof (buf));

pos2 = lseek(fd2, 0L, SEEK_CUR); /* 10 */
pos3 = lseek(fd3, 0L, SEEK_CUR); /*  0 */
```

"Open File" vs. "In-Core File"



File types (or not)

Goal

- Avoid printing a binary executable file
- Find program which “understands” a file selected by user

Derive “type” from file names

- *.exe are executable, *.c are C

Tag file with type information

- MacOS: 4-byte *type*, 4-byte *creator*

Unix: Both/neither

- Leave it (mostly) up to users

File Structure

What's *in* a file?

- Stream of bytes?
  - What character set? US-ASCII? Latin-1? Unicode?
- Stream of records?
- *Array* of records? *Tree* of records?

Record structure?

- End of “line”
  - CR, LF, CR+LF
- Fixed-length? Varying? Bounded?

File Structure - Unix

Program loader *needs to know* about executables

- "Magic numbers" in first two bytes
  - obsolete A.OUT types - OMAGIC, NMAGIC, ZMAGIC
  - ELF
  - #! - script

Otherwise, *array of bytes*

- User/application remembers meaning (hopefully!)

For a good time...

- Try the "file" command
- Read /usr/share/misc/magic or /usr/share/file/magic
  - Marvel at the dedication of the masses

File Structure –MacOS

Data fork

- Array of bytes
- Application-dependent structure

Resource fork

- Table of resources
  - Icon, Menu, Window, Dialog box
- Many resources are widely used & understood
  - Desktop program displays icons from resource fork

Access Methods

Provided by OS or optional program library

Sequential

- Like a tape
- read() next, write() next, rewind()
- Sometimes: skip forward/backward

Direct/relative

- Array of fixed-size records
- Read/write any record, by #

Access Methods –Indexed

File contains *records*

Records contain *keys*

Index maps keys ⇒ records

- Sort data portion by key
- Binary search in multi-level list

Fancy extensions

- Multiple keys, multiple indices
- Are we having a database yet?
  - Missing: relations, triggers, consistency, transactions, ...
- Unix equivalent: dbm/ndbm/gdbm/bdb/...

Directory Operations

Lookup(“index.html”)

Create(“index.html”)

Delete(“index.html”)

Rename(“index.html”, “index.html~”);

Iterate over directory contents

Scan file system

- Unix “find” command
- Backup program

Directory Types

Single-level

- **Flat global namespace –only *one* test.c**
- **Ok for floppy disks (maybe)**

Two-level

- **Every user has a directory**
- **One test.c *per user***
  - **[1003,221]PROFILE.CMD vs. [1207,438]PROFILE.CMD**
- **Typical of early timesharing**

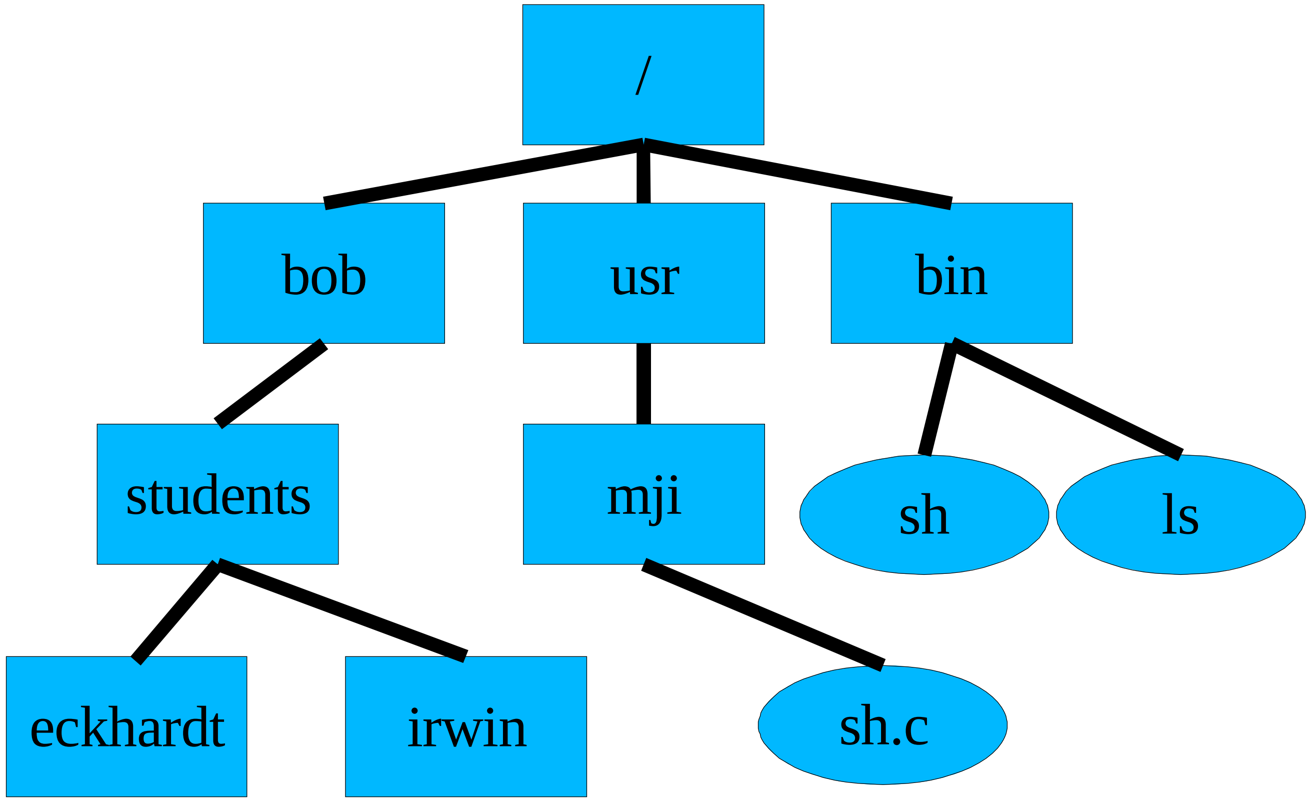
Are we having fun yet?

Tree Directories

Absolute **Pathname**

- **Sequence of directory names**
- **Starting from “root”**
- **Ending with a file name**

Tree Directories



Tree Directories

Directories are special files

- Created with special system calls – mkdir()
- Format understood & maintained by OS

Current directory (“.”)

- “Where I am now” (e.g., /usr/zzz)
- Start of *relative* pathname
  - ./stuff/foo.c or stuff/foo.c ⇒ /usr/zzz/stuff/foo.c
  - ../joe/foo.c ⇒ /usr/joe/foo.c
- Directory reference in, e.g., `p->p_fd->fd_cdir`

DAG Directories

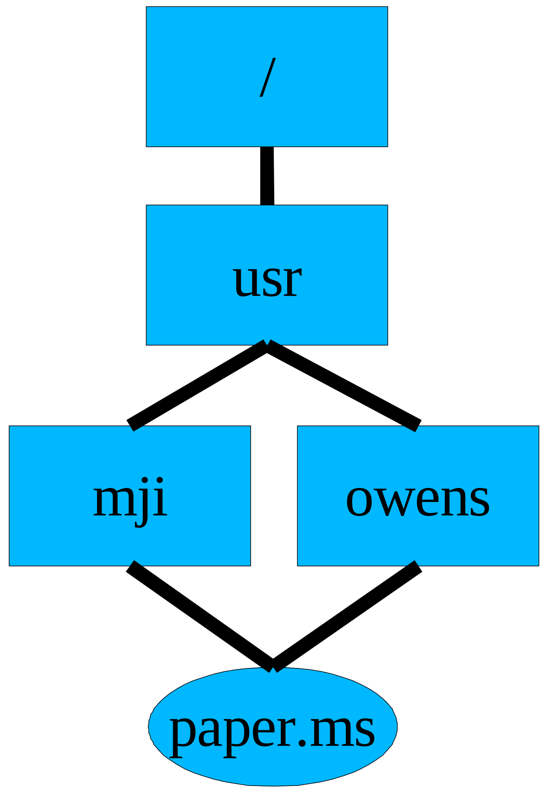
Share files and directories between users

Not mine, not yours: *ours*

Destroy when *everybody* deletes

Unix "hard link"

- Files, not directories
  - (".. problem")



Soft links

Hard links "too hard"?

- Need a level of indirection in file system?
- No "one true name" for a file
- NIH syndrome?

Alternative: soft link / symbolic link / "short cut"

- Tiny file, special type
- Contains *name* of another file
- OS dereferences link when you open() it

Hard vs. Soft Links

Hard links

- Enable reference-counted sharing
- No name is “better” than another

Soft links

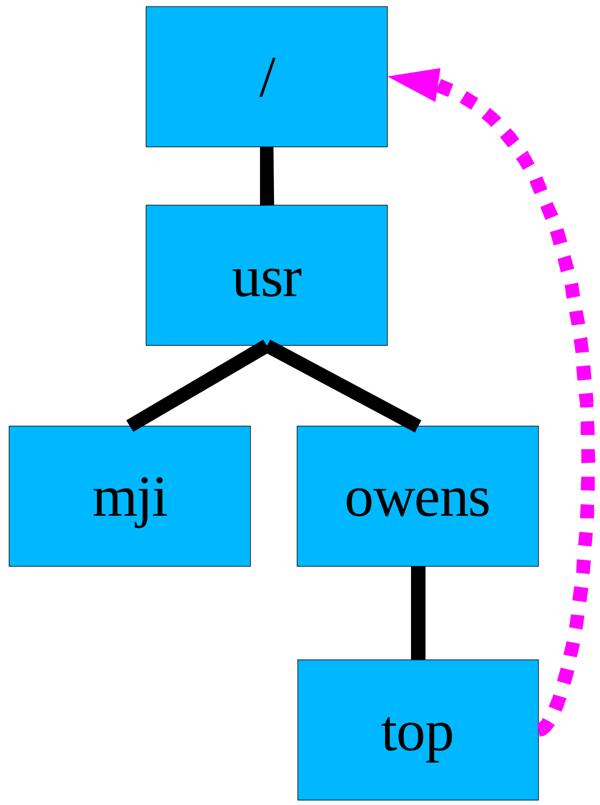
- Can soft-link a directory
    - one “true” parent, so no “.. problem”
- Work across file system & machine boundaries
- Easier to explain
- “Dangling link” problem
    - Owner of “one true file” can delete it
    - Soft links now point to nothing

Cyclic Graph Directories

Depth-first traversal can be slow!

May need *real* garbage collection

Do we really need this?



Mounting

Multiple disks on machine

Multiple partitions on disk

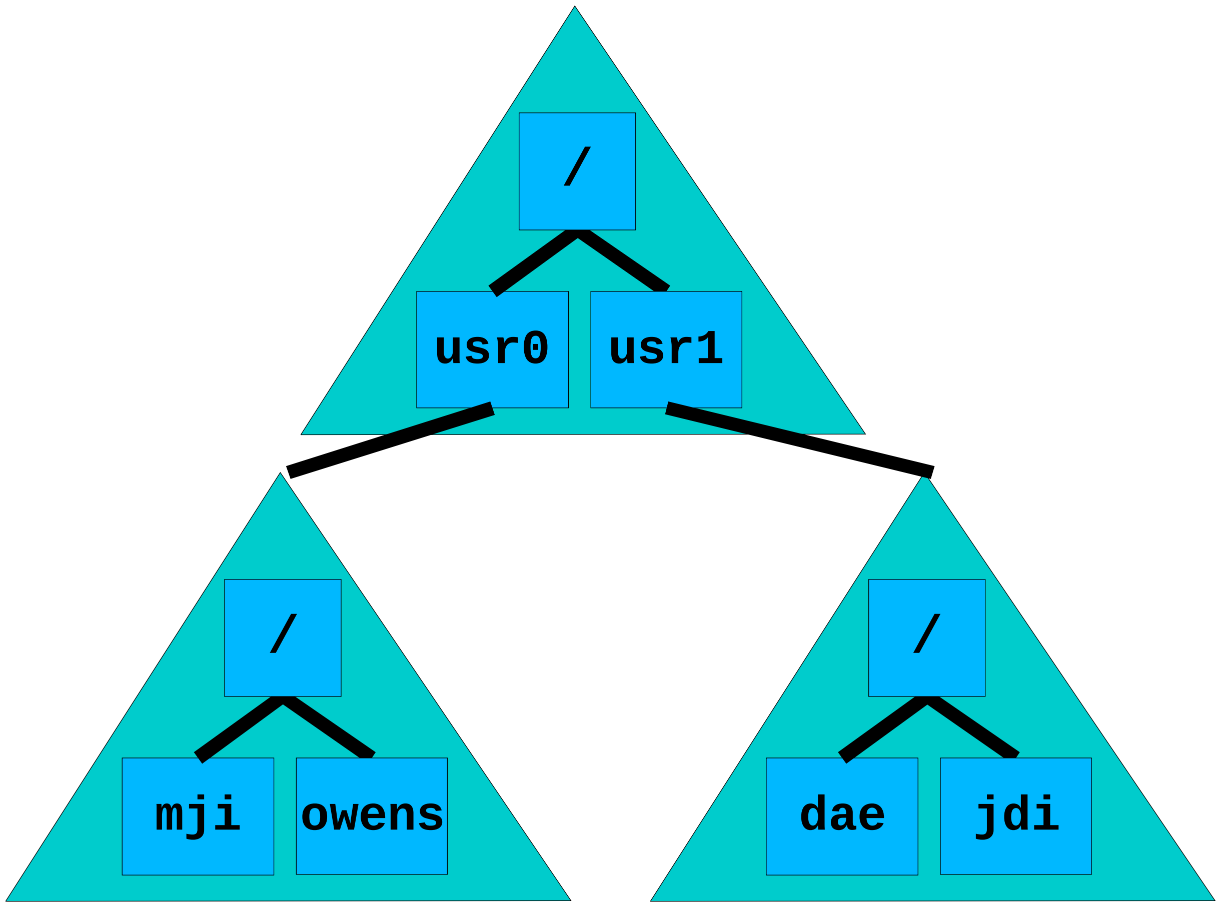
File system *within* a partition

- Or, within a volume / logical volume / ...

How to name files in “another” file system?

- Wrong way
  - C:\temp vs. D:\temp
  - [1003,221]PROFILE.CMD vs. [1207,438]PROFILE.CMD

Mounting



Multiple Users

Users want to share files

What's a user?

- Strings can be cumbersome
- Integers are nicer for OS to compare
- Unix: User ID / “uid”
- Windows: Security ID / “SID”

What's a group?

- A set of users
- Typically has its own gid / SID

Protection

Override "bit" (e.g., MS-DOG)

- Bit says "don't delete this file"
  - Unless I clear the bit

Per-file passwords

- Annoying in a hurry

Per-directory passwords

- Still annoying

Protection

Access modes

- Read, Write, Execute, Append, Delete, List, Lock, ...

Access Control List (ACL)

- File stores list of (user, modes) tuples
- Cumbersome to store, view, manage

Capability system

- User is given a list of (file, access keys) tuples
- Revocation problem

Protection –typical

File specifies *owner, group*

- Permissions for owner, permissions for group members
  - Read, write, ...
- Permissions for “other” / “world”
  - Read, write, ...

Unix

- r, w, x = 4, 2, 1
- rwxr-x--x = 0751 (octal)
- V7 Unix: 3 16-bit words specified all permission info
  - permission bits, user #, group #
    - » Andrew's /etc/passwd has 29.438 users...

Summary

File

- Abstraction of disk/tape storage
  - Records, not sectors
  - Type information
- Naming
  - Complexity due to linking
- Ownership, permissions
- Semantics of multiple open()s

Extra details in 20.7, 20.8