File Systems

The File System Abstraction

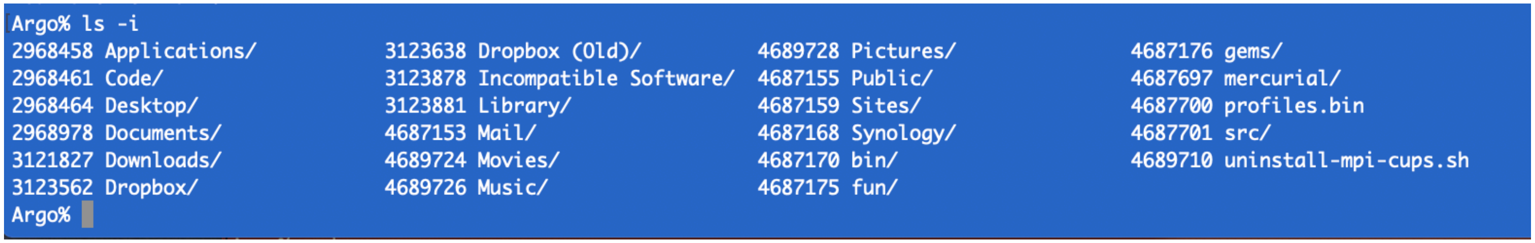
- Addresses need for long-term information storage:
  - store large amounts of information
  - do it in a way that outlives processes (RAM will not do)
  - can support concurrent access from multiple processes
- Presents applications with persistent, named data
- Two main components:
  - files
  - directories

The File

- A file is a named collection of data. In fact, it has many names, depending on context:
    - i-node number: low-level name assigned to the file by the file system
    - path: human friendly string
        - must be mapped to inode number, somehow
    - file descriptor
        - dynamically assigned handle process a uses to refer to i-node
- A file has two parts
    - data – what a user or application puts in it
        - array of untyped bytes
    - metadata – information added and managed by the OS
        - size, owner, security info, modification time, etc.

The Directory

A special file that stores mappings between human-friendly names of files and their inode numbers



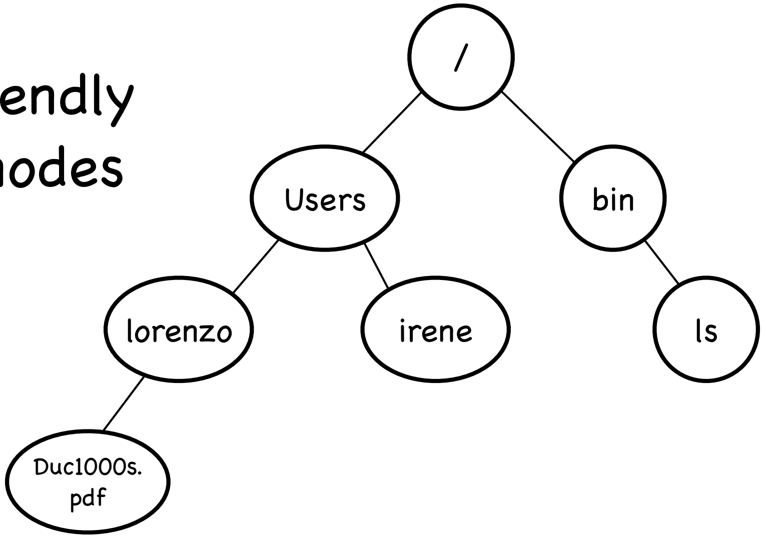
```
Argo% ls -i
2968458 Applications/       3123638 Dropbox (Old)/          4689728 Pictures/       4687176 gems/
2968461 Code/               3123878 Incompatible Software/  4687155 Public/         4687697 mercurial/
2968464 Desktop/            3123881 Library/                4687159 Sites/          4687700 profiles.bin
2968978 Documents/          4687153 Mail/                   4687168 Synology/       4687701 src/
3121827 Downloads/          4689724 Movies/                 4687170 bin/            4689710 uninstall-mpi-cups.sh
3123562 Dropbox/            4689726 Music/                  4687175 fun/
Argo% 
```

Has its own inode, of course

Mapping may apply to human-friendly names of directories and their inodes

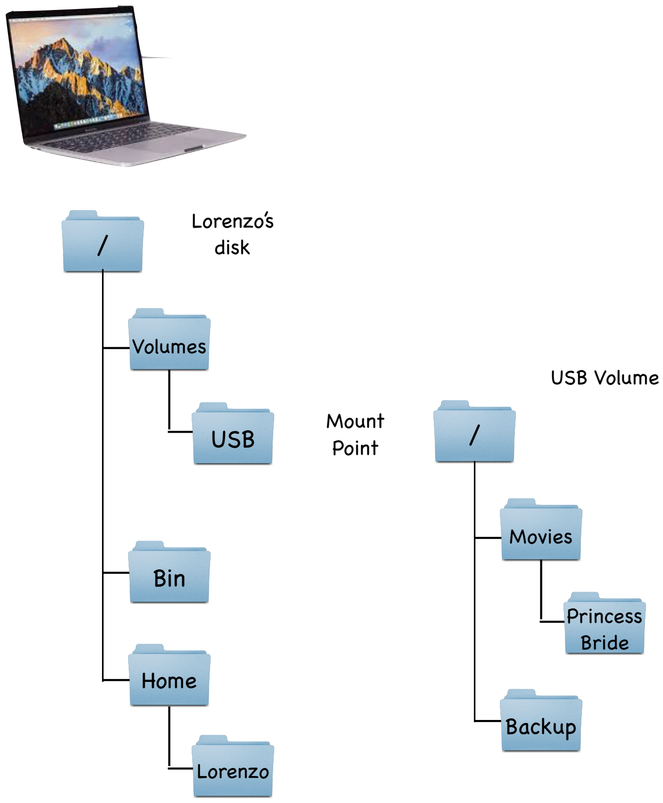
directory tree

/ indicates the root



Mount

- Mount: allows multiple file systems on multiple volumes to form a single logical hierarchy
  - a mapping from some path in existing file system to the root directory of the mounted file system



The Abstraction Stack

- I/O systems are accessed through a series of layered abstractions

Application

Library

File System

Physical Device

The Abstraction Stack

- I/O systems are accessed through a series of layered abstractions

File System API and Performance

Application

Library

File System

Block Cache

Block Device Interface

Device Access

Device Driver

MM I/O, DMA,Interrupts

Physical Device

The Abstraction Stack

- I/O systems are accessed through a series of layered abstractions
  - Caches blocks recently read from disk
  - Buffers recently written blocks

Application

Library

File System

Block Cache

Block Device Interface

Device Driver

MM I/O, DMA,Interrupts

Physical Device

The Abstraction Stack

- I/O systems are accessed through a series of layered abstractions
  - Caches blocks recently read from disk
  - Buffers recently written blocks
  - Single interface to many devices, allows data to be read/written in fixed sized blocks

Application

Library

File System

Block Cache

Block Device Interface

Device Driver

MM I/O, DMA,Interrupts

Physical Device

The Abstraction Stack

- I/O systems are accessed through a series of layered abstractions
  - Caches blocks recently read from disk
  - Buffers recently written blocks
  - Single interface to many devices, allows data to be read/written in fixed sized blocks
  - Translates OS abstractions and hw specific details of I/O devices

Application

Library

File System

Block Cache

Block Device Interface

Device Driver

MM I/O, DMA,Interrupts

Physical Device

The Abstraction Stack

- I/O systems are accessed through a series of layered abstractions
    - Caches blocks recently read from disk
    - Buffers recently written blocks
    - Single interface to many devices, allows data to be read/written in fixed sized blocks
    - Translates OS abstractions and hw specific details of I/O devices
    - Control registers, bulk data transfer, OS notifications

Application

Library

File System

Block Cache

Block Device Interface

Device Driver

MM I/O, DMA,Interrupts

Physical Device

File System API

- Creating a file

path flags permissions

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

returns a file descriptor, a per-process integer that grants process a capability to perform certain operations on the file

`int close(int fd);` closes the file

- Reading/Writing

```
ssize_t  read (int fd, void *buf, size_t count);

ssize_t  write (int fd, void *buf, size_t count);
```

return number of bytes read/written

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

repositions file's offset (initially 0, updates on reads and writes)

to offset bytes from beginning of file (SEEK_SET)

to offset bytes from current location (SEEK_CUR)

to offset bytes after the end of the file (SEEK_END)

File System API

- Writing synchronously

  int fsynch (int fd);

  flushes to disk all dirty data for file referred to by fd

  if file is newly created, must fsynch also its directory!

- Getting file's metadata

  stat() , fstat() — return a stat 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 */
};
```

retrieved from file's inode

on disk, per-file data structure may be cached in memory