PRINCIPLES OF COMPUTER SYSTEMS DESIGN

CSE130

Winter 2020

File Systems II - The Virtual File System

Baskin Engineering UC SANTA CRUZ

Notices

- **Lab 3** due **Sunday March 1**
- **Assignment 5** will be available **Monday March 2**
- **Week 10 Lectures** (Final Prep) will ***not*** be webcast

Today's Lecture

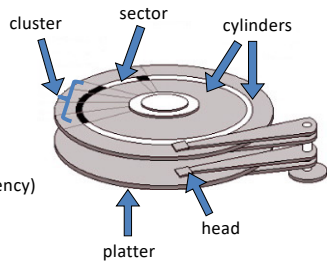
- File System Basics
- The Layered File System
- Logical to Physical Mapping
- The Virtual File System
- More Lab 3 Secret Sauce 

File Systems : The Basics

- Hard Disk Drives (HDD)
- Solid State Drives (SSD)
- Logical to Physical Mapping
- The Virtual File System
- Partitions
- Mount Points

How Does a Hard Drive Work?

- Key Attributes
  - Read / Write any sector
  - Direct access
  - Supports sequential, indexed etc.
  - **Seeks are slow** (mechanical)
- To read/write a disk sector:
  - Move head to cylinder (seek)
  - Wait for sector to rotate to head (latency)
  - Read or write (transfer)



How Does a Solid State Drive Work?

- **RAM is Volatile Memory** - Information is lost when power removed
- **Flash Memory is Non Volatile**
  - NOR: Transistors wired in parallel
  - NAND: Transistors wired in series
    - Less expensive than NOR
    - Two transistors to a "cell"
    - Single Level Cell (SLC) can store one bit per cell (0 or 1)
    - **Multi Level Cell** (MLC) store two bits per cell (00, 01, 10, or 11)
      - Inexpensive, better storage capacity, but "wears out" faster than SLC
- **MLC NAND Flash Memory**
  - Typically can deliver 50,000 write/erase operations per cell during it's life
  - Device drivers & disk controllers implement "wear leveling" to make sure individual cells don't "wear out" faster than any other

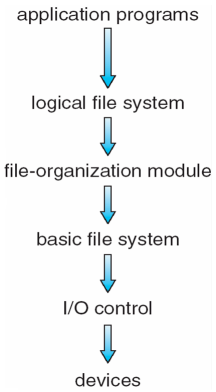
HDD vs. SSD (a non scientific evaluation)

- **Program Load**
  - SSD twice as fast as HDD
- **Data Read/Write**
  - SSD ten times as fast as HDD
- **Power Consumption**
  - SSD consume less power than HDD
  - SSD generates less heat
  - SSD produces less noise
- **Price**
  - SSD around five times more expensive than HDD for the same capacity in the 128GB to 500GB range
- **Lifespan**
  - HDDs are expected to last decades, SSDs just years

HDD & SSD Working Together (non scientific)

- **SSDs Store:**
  - Things that get *read* a lot, are small, and you don't have many of
    - The Operating System
    - User Programs
- **HDDs Store:**
  - Things that get *written* a lot
    - Transaction Databases
    - User Documents (Word, Excel, Source Code)
    - Memory Swap Space
  - Things that are large
    - Movies
    - Genome Sequencer Output
  - Things you have lots of
    - Music
    - Images

The Layered File System



Logical to Physical Mapping

- File is a sequence of **Logical Records**
- Basic file system is an `array[0..N]` of **disk blocks**
  - Multiple disks => multiple arrays
- Mapping function required to go from logical to basic file system

Logical File System

```
// direct
record = read(f)
// buffered
read(f, buf, record_size)
```

Map

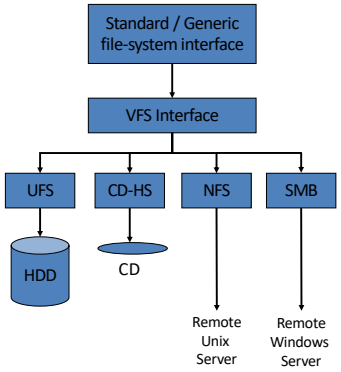
Basic File System

```
get(disk no. , block no.)
```

- We need to store which block on which disk each file resides in
- **Problems?**

The Virtual File System

- Goal: **Seamlessly integrate different file systems**
  - File IDs unique within partition
  - Hide different semantics for different physical file system types
  - Provide network transparency
    - User program does not know if the file system it is using is local or remote
- **Virtual File System**
  - Supports generic interface
  - Identifiers unique network wide



File System Structures

- **Boot control block**
  - Dedicated to OS for booting
  - The first block of a volume
- **Partition control block**
  - Partition - an instance of a file system
    - Multiple partitions per disk, or
    - One partition spanning multiple disks
  - PCB contains details of this instance
- **Directory structure**
- **File control blocks**

Disks & Partitions

```
$ diskutil list
/dev/disk0 (internal, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *1.0 TB     disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:                  Apple_HFS Macintosh HD            999.3 GB   disk0s2
   3:                 Apple_Boot Recovery HD             650.0 MB   disk0s3

/dev/disk1 (external, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *2.0 TB     disk1
   1:                        EFI EFI                     209.7 MB   disk1s1
   2:                  Apple_HFS Time Machine            2.0 TB     disk1s2
```


Partitions, Mounting and Mount Points

- **Partition** contains raw or cooked (formatted) file system
- Boot information has own partition (boot partition)
- **Root partition** contains the OS
- Other partitions can hold other OSes, other file systems, or be raw
- A file system must be **mounted** before it can be accessed
- A file system is mounted at a **mount point**
  - Windows automatically mounts all volumes
  - Unix mounts some but not others
- Mounting creates entry in a **mount table**
- At mount time, file system **consistency** checked
  - If inconsistent, (try to) fix it, attempt mount again
  - If yes, add to mount table, allow access

Mount Points : `df`

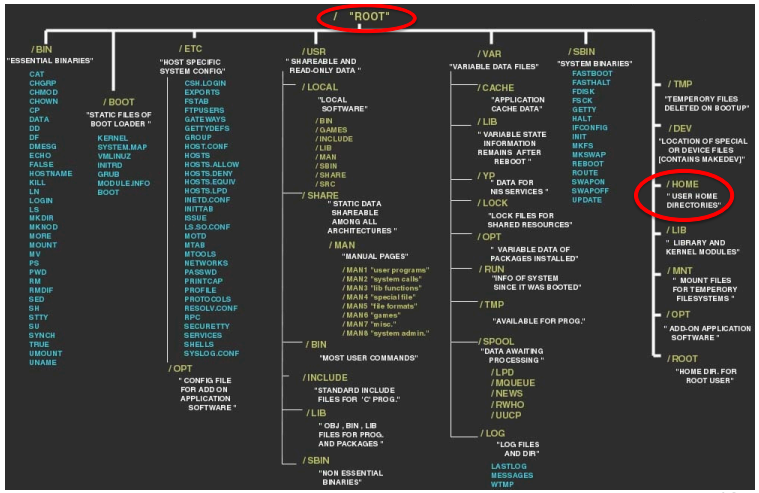
```
$ df -P -m
Filesystem    1M-blocks    Used Available Capacity  Mounted on
/dev/disk0s2     953049  294823    657976    31%    /
devfs                 0       0         0   100%    /dev
map -hosts            0       0         0   100%    /net
map auto_home         0       0         0   100%    /home
/dev/disk1s2    1907401 1240123    667278    66%    /Volumes/Time Machine
```



```
$ df -m
Filesystem                                    1M-blocks    Used Available Use% Mounted on
udev                                               1954       0      1954   0% /dev
tmpfs                                               395      41       354  11% /run
/dev/sda1                                         18015   16905       173  99% /
tmpfs                                              1974       0      1974   0% /dev/shm
tmpfs                                                 5       1         5   1% /run/lock
tmpfs                                              1974       0      1974   0% /sys/fs/cgroup
cgmfs                                                 1       0         1   0% /run/cgmanager/fs
bsd-05.soe.ucsc.edu:/export/home/sentinel       5242880 2276533   2966347  44% /soe/sentinel
bsd-05:/export/local/linux                        51200    3840     47361   8% /share/local
bsd-05.soe.ucsc.edu:/export/software            1048576  744812    303765  72% /bsoe/software
bsd-05:/export/local/share                        51200    3074     48127   7% /share/share
bsd-05.soe.ucsc.edu:/export/home/dcharris       5242880 2276533   2966347  44% /soe/dcharris
bsd-05.soe.ucsc.edu:/export/cse/classes          614400  443646    170755  73% /cse/classes
bsd-05.soe.ucsc.edu:/export/home/heidi2         5242880 2276533   2966347  44% /soe/heidi2
bsd-05.soe.ucsc.edu:/export/home/cgao15         5242880 2276533   2966347  44% /soe/cgao15
bsd-05.soe.ucsc.edu:/export/home/jmustain       5242880 2276533   2966347  44% /soe/jmustain
bsd-05.soe.ucsc.edu:/export/home/pohl           5242880 2276533   2966347  44% /soe/pohl
bsd-05.soe.ucsc.edu:/export/projects/public       10240    1821      8420  18% /projects/public
bsd-05.soe.ucsc.edu:/export/home/xinli          5242880 2276533   2966347  44% /soe/xinli
bsd-05.soe.ucsc.edu:/export/home/alw            5242880 2276533   2966347  44% /soe/alw
bsd-05.soe.ucsc.edu:/export/home/tjg            5242880 2276533   2966347  44% /soe/tjg
bsd-05.soe.ucsc.edu:/export/admin               5242880 2761557   2481324  53% /bsoe/admin
bsd-05.soe.ucsc.edu:/export/software/old/linuxSW 1048576  744812    303765  72% /projects/linuxSW
bsd-05.soe.ucsc.edu:/export/home/linisac        5242880 2276533   2966347  44% /soe/linisac
bsd-05.soe.ucsc.edu:/export/home/dalves         5242880 2276533   2966347  44% /soe/dalves
bsd-05.soe.ucsc.edu:/export/home/cgrim          5242880 2276533   2966347  44% /soe/cgrim
bsd-05.soe.ucsc.edu:/export/home/hzheng6        5242880 2276533   2966347  44% /soe/hzheng6
bsd-05.soe.ucsc.edu:/export/home/rcompton       5242880 2276533   2966347  44% /soe/rcompton
bsd-05.soe.ucsc.edu:/export/home/coco           5242880 2276533   2966347  44% /soe/coco
bsd-05.soe.ucsc.edu:/export/home/qchen63        5242880 2276533   2966347  44% /soe/qchen63
bsd-05.soe.ucsc.edu:/export/home/memory         5242880 2276533   2966347  44% /soe/memory
bsd-05.soe.ucsc.edu:/export/projects/linqs      6291456 4921128   1370329  79% /projects/linqs
bsd-05.soe.ucsc.edu:/export/home/musher         5242880 2276533   2966347  44% /soe/musher
```

Kernel File System Structures

- In-memory **mount table**
- **Directory cache**
- **Buffers** for reading and writing
  - Usually circular buffers
- **System wide open file table**:
  - File control blocks
  - Data buffers
- **Per-process open file tables**:
  - All files that are open as a result of an `open()` system call not matched with a corresponding `close()`
    - **You are doing this in Lab 3 system calls section!**
  - Position within the file as a result of the `seek()` or `read()` system calls



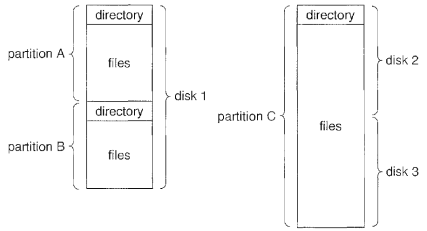
 http://leekgellin.co.uk/wp-content/uploads/2013/06/linux-file-system-hierarchy.jpg

Mounting a New File System

- Consider a Unix computer with one disk, formatted as a single partition mounted as the root directory "/"
- Users' home directories are beneath /home
  /home/fred, /home/barney, /home/wilma, /home/betty `etc.`
- Some time later, the single file system is getting full with user junk so the system administrator decides to add a new HDD just for the users, they:
  - Partition the disk into a single file system and format it
  - Mount the new file system on **/home**
  - **Problem?**
- Instead
  - Mount new file system on **/mnt**
  - Recursively copy all user directories from **/home** to **/mnt**
  - Recursively remove all directories below **/home**
  - Un-mount new file system from **/mnt**
  - Mount the new file system on **/home**

Device Directory

- For each file in a **partition**:
  - Name
  - Type (file, directory, link)
  - Address
  - Current length
  - Maximum length
  - Date last accessed
  - Date last updated
  - Owner ID
  - Protection information



Multiple partitions on one disk

One partition on multiple disks

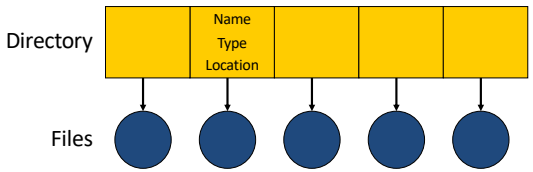
Device Directory Operations

- For any given file:
  - Search by name
  - **Create**
  - **Write**
  - **Read**
  - **Delete**
  - Rename
- List a directory
- Traverse the file system (enumerate its contents)

Device Directory Fundamentals

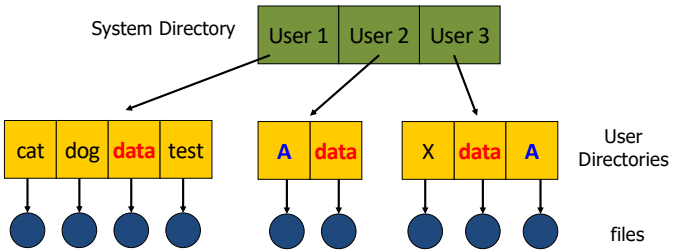
- Goals:
  - **Efficiency**
    - Locate files quickly
  - **Naming**
    - Multiple users can have same name for multiple files
    - One file can have multiple different names
  - **Grouping**
    - Logical grouping of files by properties
- Possible Directory Structures:
  - Single Level
  - Two Level
  - Acyclic Graph (Tree)

Single Level ("Flat") Directory Structure



Problems? Naming & Grouping ☹

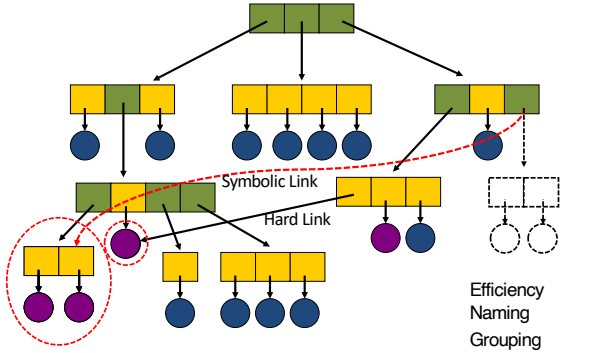
Two-Level Directory Structure



Problems? Grouping ☹

Benefits? Path + Name => Simple & Efficient ☺

Graph Directory Structure & Links



Efficiency
Naming
Grouping
Sharing is easy

Cycles can occur

Hard & Symbolic Links

- **Hard link**s place a single physical file (a file on disk) in multiple locations in the (logical) directory graph

```
$ ln <original-file-path> <new-link-path>
$ ln /home/fred/laundrylist.xls fredslaundrylist.xls
$ ls -l *.xls
lrwxr-xr-x 1 david staff  1007 22 Nov 10:11 fredslaundrylist.xls
```

- **Symbolic link**s make files **appear as if** they are in the directory graph in multiple locations

```
$ ln -s <real-file-path> <symbolic-link-path>
$ ln -s /home/fred/pintos mypintos
$ ls -l mypintos
lrwxr-xr-x 1 david staff     8 22 Nov 10:12 mypintos -> /home/fred/pintos
```

- Consider a graph structured directory with a file X and two symbolic links to X (sym1 and sym2)
  - What happens if we delete file X but not links sym1 and sym2?
  - **How you might avoid this problem?**

Directory Implementation

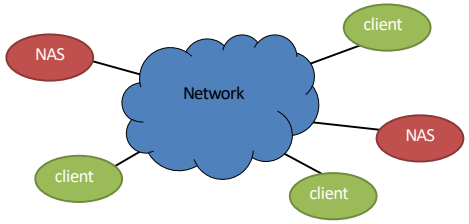
- **Linear List**
  - Simple to program but time-consuming to execute
    - Easy: Add
    - Hard: Delete, Find
- **Hash Table**
  - A **linear list** stores directory entries
  - A **hash table** takes a value computed from file name and returns a pointer to one location in the list
  - **Problems?**
    - Have to be careful with **name hash collisions**

File Consistency Semantics

- What is **file consistency**?
  - How multiple simultaneous users access a shared file
- **UNIX File System** (UFS)
  - Modifications immediately visible
  - Can share Location Pointer
    - All sharing processes progress through the file together
- **Andrew File System** (AFS)
  - A **Distributed File System**
  - Modifications not immediately visible
  - Once file is closed, changes become visible to future sessions, but existing sessions do not see the changes unless they close and reopen the file
  - When you login to `noggin`, `nogbad` or `unix` your files are in AFS

Network Attached Storage

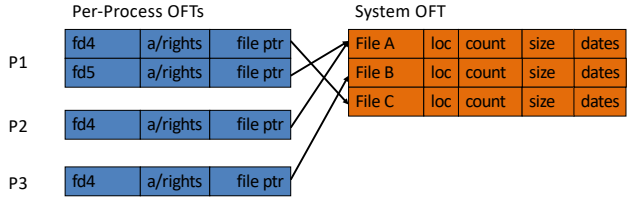
- Network-attached storage (NAS) is disk space made available over a network rather than over a local connection (such as a system bus)
- **NFS** (Unix) and **SMB** (Windows) are established protocols
- Amazon **S3** is a modern "cloud" NAS implementation
- Implemented via **remote procedure calls** (RPCs) between clients and storage hosts



30

Lab 3 Secret Sauce

The Per Process Open File Table



32

Pintos Per-Process Open File Table

- Pintos System Calls:

```
int open(const char *file); // returns a file descriptor
int read(int fd, void *buffer, unsigned length);
int write(int fd, const void *buffer, unsigned length);
```

- Pintos File System:

```
struct file *filesys_open(const char *name);
off_t file_read(struct file *, void *, off_t);
off_t file_write(struct file *, const void *, off_t);
```

- You need to:
  - Map numeric file descriptors to file pointers
  - Cannot use file descriptors 0, 1, or 2 (`stdin`, `stdout`, and `stderr`)
  - Where are you going to store the per process open file table?
  - No need to store access rights in your OFT

33

Next Lecture

- The UNIX File System
- The Network File System
- Mass Storage