

# CS169 Assignment 2: Virtual File System

"There's no possible way stepping into a function could trash a variable's value unless I returned the address of a stack-allocated variable, and I *know* I didn't do that, it's too stupid." - colin  
<10 minutes later>

"You idiot, you returned the address of a variable on the stack." - atm

The CS169TAs  
Fall, 2008

# Virtual File System

- Interface between kernel and various filesystems (e.g., testfs, s5fs, devfs\*, procfs\*, zfs\*)
- Defines abstract operations like open, close
- Partially defines operations like read, write, mkdir, mknod, etc. which call into underlying filesystem to do some of the work.

\* not actually part of Weenix (yet)

# VFS



User Land!



VM

System Calls

open, close, read, write, lseek, getdents

fork, exec

yield

mmap

VFS

TTY driver

Line discipline

Virtual device drivers

Process Management

Scheduler

Dispatcher

TLB mgmt

Paging

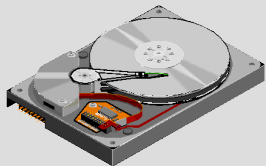
Core map

File system (S5FS)

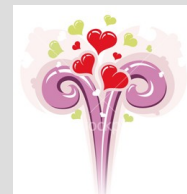
File system (testfs)

Disk driver

Terminal driver



/dev/null



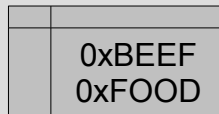
/dev/zero



CPU



Memory



TLB

# Important data structures

- filesystem – represents the FS itself
  - includes information like root vnode, FS ops (like reading vnodes)
  - only one mounted filesystem is required – and the mounting is hardcoded (you need not worry about it)
- vnode – represents a node in the FS tree
  - each one is initialized by the file system
  - one for each directory and file (and link, etc.)
  - contains operations for this object
  - contains attributes like mutex, type of node, pointer to inode (fs's counterpart to vnode), and **refcount**

# Vnode Memory Management

- Can't store vnodes for whole filesystem all day.
- Only need to be around when file is open/in-use
- Vnodes can be (and are) (re)constructed by the filesystem by reading the corresponding data off the disk
- So we can keep a reference count and free the vnode when it drops to 0
- `vget(vnum)`, `vput(vnode)`, `vref(vnode)`
- if refcount goes to 0 prematurely, data is garbage
- if left with extra vnodes around, you waste memory (and weenix panics on shutdown)

# What you (don't) have to do

- Filesystem setup (mounting) – done for you
- *testfs* – mostly done for you, but you must write *readdir* (see comments)
- Name operations (vnode lookup – difficult)
  - lookup, dir\_namev, open\_namev
- System calls (lots to do, but simpler and similar to each other)
  - open, close, read, write, dup[2], mk[nod,dir], rmdir, unlink, link, rename, chdir, getdent, lseek, stat, special\_file\_[read,write,mmap]

# Name: lookup

- lookup: given a vnode for a directory and a path for a file in that directory, return the vnode for that particular file
- Deals with special cases ('.', '..', etc.)
- Mostly delegates to filesystem-specific directory lookup routine

# Name: dir\_namev

- Paths have a directory-part and a base-part
- /usr/bin/ls: /usr/bin = dir, ls = base
- dir\_namev: given a full path, return the path for the base part, and the vnode for the directory part
- First, parse the path into parts
- Continue doing lookups down the tree until you get to the end

# dir\_namev example

- Path = “/usr/bin/l<sub>s</sub>”
- First Lookup “usr” in the root directory...
- ...then lookup “bin” in the resulting vnode...
- ...then return that vnode, and “l<sub>s</sub>” for the name...
- ...and make sure you didn't mess up the vnode ref counts in the process.

# Name: open\_namev

- open\_namev: given a pathname and the flags from open(2), return the vnode for the full path (creating it, if O\_CREAT is specified)
- Uses dir\_namev, lookup, and possibly the *create* vnode operation

# Testing

- Finish testing Kern.
- No, really. Finish testing Kern.
- Now, test vfs using the (hopefully working) testfs
  - Write lots of your own test code
  - Use our vfs\_privtest (but it is **not** exhaustive!)

# Getting started

- Uncomment “VFS = true” in Makefile.defines
- Go.