File Systems: Naming

- Learning Objective
  - Explain how to implement a hierarchical name space.
  - Identify the key SFS data structures.
  - Map system call level operations to manipulations of SFS data structures.
- Topics:
  - Naming exercise
  - In-depth study of directory implementation
  - SFS data structures
  - SFS operations

Exercise 3: Naming

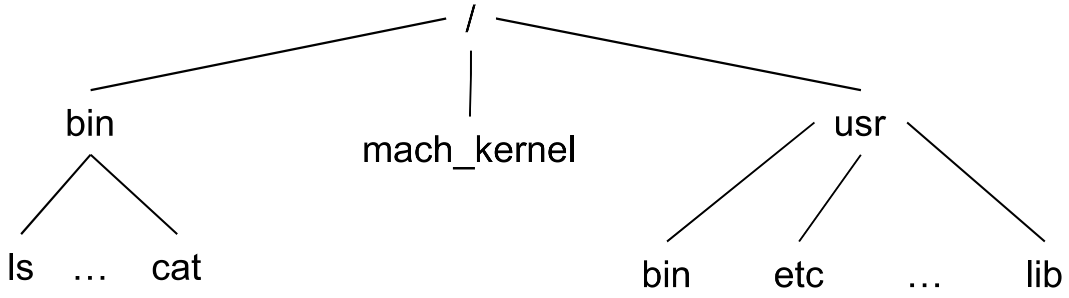
- Let’s think about how to implement a hierarchical name space (i.e., directories & files).
  - How will you represent a directory?
  - How will you find the root directory (“/”)?
  - How will you support traversing up a directory tree (cd ..)?
  - Be as specific as you can.

Hierarchical Naming

- Generalized tree structure
  - Directories are regular files with a special format.
  - A bit in the file meta-data indicates that a file is of type directory.
  - A directory entry is simply a mapping between names and a file index (a collection of name/value pairs).
    - User programs can read directories just like they read files.
    - Only the operating system can write directories (wouldn't want a user to corrupt the directory structure)

- Pros:

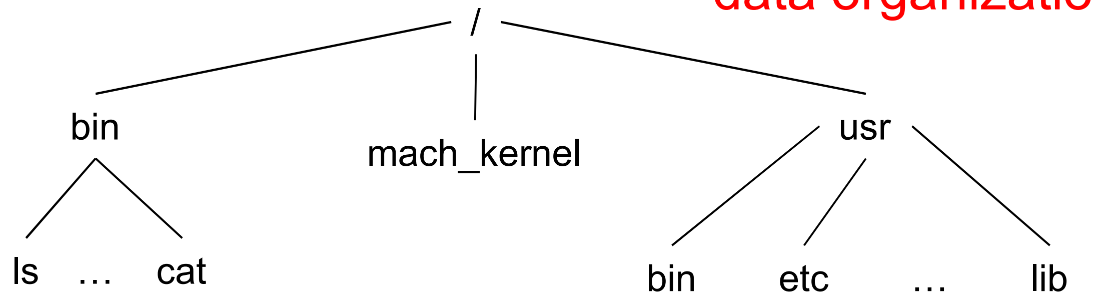
- Cons:



Hierarchical Naming

- Generalized tree structure
  - Directories are regular files with a special format.
  - A bit in the file meta-data indicates that a file is of type directory.
  - A directory entry is simply a mapping between names and a file index (a collection of name/value pairs).
    - User programs can read directories just like they read files.
    - Only the operating system can write directories (wouldn't want a user to corrupt the directory structure)

- Pros:
  - Reuses file implementation
  - Mimics how people used to organize files in file cabinets.
- Cons:
  - Doesn't natively provide the kind of searching that is commonplace today in data organization.



Traditional Directory Implementation

- Directories are represented like files.
- Contents of directories are structured (`dirents`).
  - Name
  - Inode number
  - Type
- Directories grow in chunks of `dirents` that fit on a single disk block.
- Root directory has a designated inode.

The Root Directory

- This is the contents of the "/" directory on my machine.

Name	inumber	Name	inumber	Name	inumber
Applications	113	Desktop Folder	844727	Developer	844731
Documents	937803	Library	213	Marketocracy	937813
Network	84416	System	37	Updaters	937816
Users	38892	Volumes	26447	bin	24377
cdrom	937840	cores	84418	dev	296
etc	25116	home	5	mach_kernel	552433
net	3	opt	937844	private	214
sbin	4512	sw	1024168	tmp	25155
usr	40	var	25156	.	2
..	2				

Walking a Directory Path

- For historical reasons (because original versions of UNIX did this) we call:
  - File index structures: inodes
  - References to file index structures: inumbers
- Given a path /C1/C2/C3 …
  - Start at the root directory (a designated directory with a designated inumber).
    1. Let `inum` = root directory inumber; `current component` = C1
    2. Read the directory data for `inum`
    3. Find the entry with the name equal to the `current component`
    4. Fine the associated inumber
    5. Read the inode for that inumber
       - If it's not a directory, this is a bad pathname
       - If it is a directory, set inum to the inumber; set current component to next part of path and iterate back to step 2.

Directory Example

Assume:

- **Inode 2 is in disk block 100**
- **Inodes fit 8 to the block**
- **Block 100 contains inodes 0-7, 101 contains 8-15, etc.**
- **There are 100 blocks of inodes**

Exercise:
List all the blocks, in order that you need to read to open /usr/lib/libc.a

The number in these inodes is what is found in daddr[0]

	Disk block number	**Contents**							
Inodes	100			200					
	101	202	203						
	102	204	205						
	…								
Data Blocks	200	., 2	.., 2	bin, 8					
		usr, 16	boot, 35	kadb, 27					
	201	., 11	.., 2	Some text					
		is in	this file						
	202	., 8	.., 2	ls, 91					
		csh, 105							
	203	., 9	.., 16	libc.a, 55					
		font, 77							
	204	., 16	.., 2	lib, 9					
		share, 52	ucb, 15	old, 66					

Directory Example

The number in these inodes is what is found in daddr[0]

	Disk block number	Contents							
Inodes	100			200					
	101	202	203						
	102	204	205						
	…								
Data Blocks	200	., 2	.., 2	bin, 8					
		usr, 16	boot, 35	kadb, 27					
	201	., 11	.., 2	Some text					
		is in	this file						
	202	., 8	.., 2	ls, 91					
		csh, 105							
	203	., 9	.., 16	libc.a, 55					
		font, 77							
	204	., 16	.., 2	lib, 9					
		share, 52	ucb, 15	old, 66					

Assume:

- **Inode 2 is in disk block 100**
- **Inodes fit 8 to the block**
- **Block 100 contains inodes 0-7, 101 contains 8-15, etc.**
- **There are 100 blocks of inodes**

Exercise:
List all the blocks, in order that you need to read to open /usr/lib/libc.a

More Directory Fun

- In POSIX, every directory has two special entries "." and "..".
  - The "." directory refers to the directory itself.
  - The ".." directory refers to the parent directory.
  - This is how the file system implements paths such as ../asst2.
- It is possible for more than one directory entry to refer to a single file.
  - *Hard link*: the same inumber appears in two different directories. The reference count for the inumber is incremented.
    - Could you create a hard link between two directories in different file systems?
    - When you remove (unlink) a file, you decrement its reference count and remove a name from a directory. When the reference count goes to zero, the file's blocks are actually freed.
  - *Soft link* (symbolic link): file that contains the name of another file.
    - Files of this sort are identified by a bit in their file descriptor.
    - When the OS encounters a symbolic link, it continues pathname resolution using the pathname that appears in the file.
    - Can you create a soft link between two directories?
- What is the minimum link count for a directory?

Working Directory

- It is cumbersome (and inefficient for the OS) to use full pathnames every time you reference a file.
- POSIX maintains a single “current working directory” (cwd) for each process. The inumber of the cwd is stored in the user structure.
- When the OS wants to translate a name to an inumber, it looks at the first character in the path. If that character is “/”, the OS begins looking at the root. If it is not a path, the OS begins looking in the current directory.
- Some systems allow you to have more than one current working directory. The list of directories that are in the “current working directory set” are called a search path.

The VFS/Vnode Layer

- Context:
  - The year is 1985 (do not remind me that you weren’t even born or your parents weren’t even married yet).
  - Personal computers are fairly new toys.
  - Single-user workstations (powerful personal computers) are a new thing.
  - Workstation users need to store files reliably.
  - Workstations have local disks, but they aren’t terribly reliable, and if a workstation crashes, its disks (and data) are unavailable.
  - It’s difficult to access one workstation’s files from another workstation.
- So, what do you do?

The VFS/Vnode Layer

- Context:
  - The year is 1985 (do not remind me that you weren't even born or your parents weren't even married yet).
  - Personal computers are fairly new toys.
  - Single-user workstations (powerful personal computers) are a new thing.
  - Workstation users need to store files reliably.
  - Workstations have local disks, but they aren't terribly reliable, and if a workstation crashes, its disks (and data) are unavailable.
  - It's difficult to access one workstation's files from another workstation.
- So, what do you do?
  - You build a network file system! (NFS)

The Engineering Challenge

- Traditionally a system had only a single file system.
- Now you want to add a second one.
- How do you do it?
  - Hand code everything?
  - Think about a generic interface to the file system code and rewrite the existing system to use it.

The Engineering Challenge

- Traditionally a system had only a single file system.
- Now you want to add a second one.
- How do you do it?
  - Hand code everything?
  - Think about a generic interface to the file system code and rewrite the existing system to use it.
- Enter VFS: Virtual File System: one VFS/file system
  - Operations performed on entire file systems, e.g., :
    - unmount
    - root – returns the root of the file system
    - statfs – returns file system statistics
    - sync – write all dirty data to disk

Vnodes

- One vnode per file
  - Abstract representation of a file
  - To get real work done, you have to pass the operation to the underlying implementation.
- Vnode operations:

- open
- close
- rdwr
- ioctl
- select
- getattr
- setattr
- lookup
- create
- remove
- link
- rename
- mkdir
- rmdir
- readdir
- symlink
- readlink
- fsync
- inactive
- bmap
- strategy
- bread
- brelse

And Speaking of SFS…

- How does SFS represent a file?
  - `kern/include/kern/sfs.h`

```
struct sfs_inode {
    uint32_t sfi_size;            /* File size in bytes */
    uint16_t sfi_type;            /* File or directory */
    uint16_t sfi_linkcount;       /* #hard links */
    uint32_t sfi_direct[15];      /* Direct blocks */
    uint32_t sfi_indirect;        /* Indirect block */
    uint32_t sfi_dindirect;       /* Double indirect */
    uint32_t sfi_tindirect;       /* Triple indirect */
    uint32_t sfi_waste[108];      /* Pad to 512 bytes */
}
```

SFS Free Space Management

- See `kern/include/sfs.h`
- A `struct sfs_fs` is the in-memory representation of an SFS file system.
- That structure has the following fields:

```
struct bitmap *sfs_freemap;
bool sfs_freemapdirty;
struct lock *sfs_freemaplock;
```

- The file `kern/include/kern/sfs.h` defines the following constants:

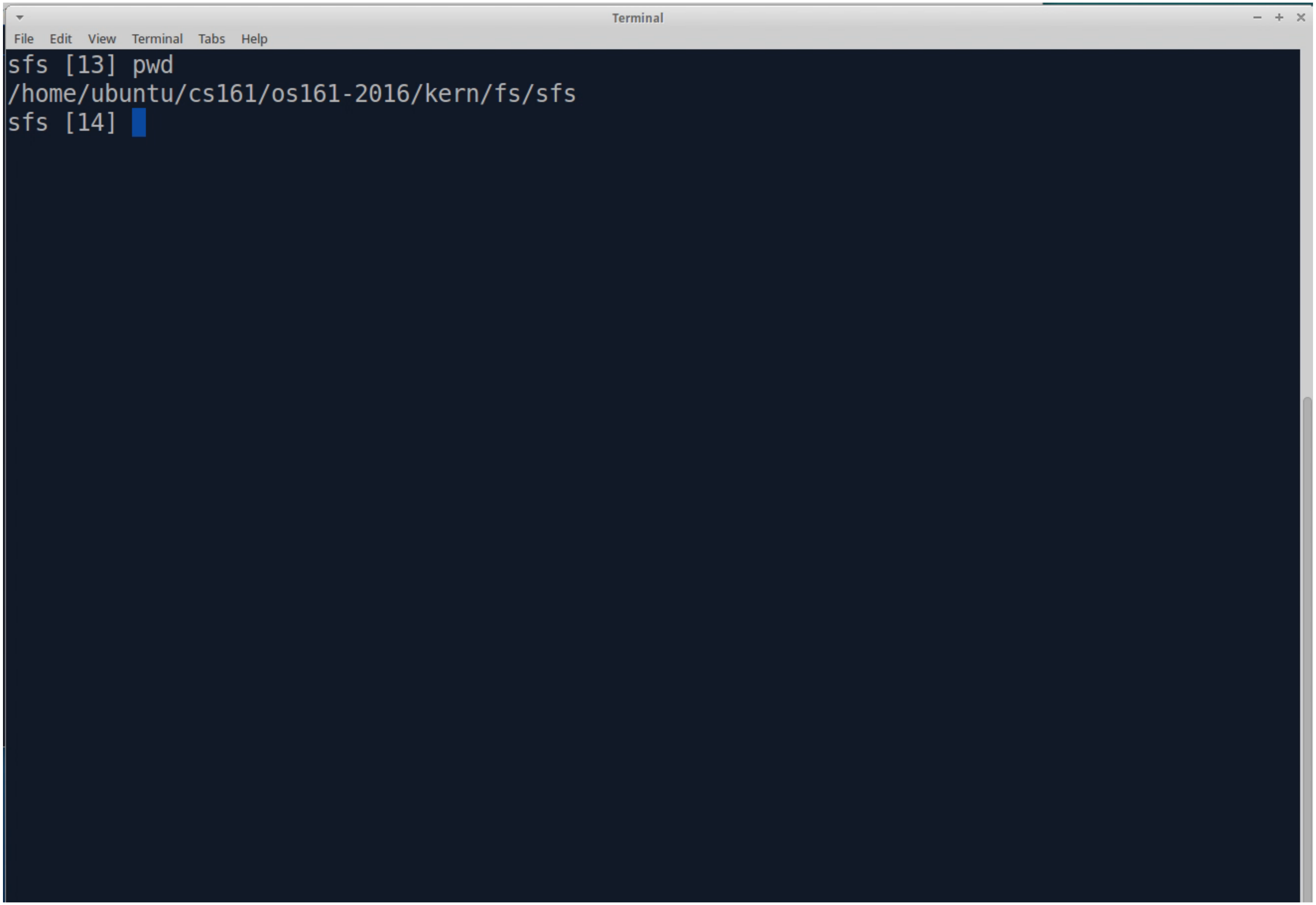
```
#define SFS_FREEMAP_START 2
```

SFS Directories

- See `kern/include/kern/sfs.h`

```
struct sfs_direntry {
        uint32_t sfd_ino;          /* Inode number */
        char sfd_name[60];         /* Filename */
};
```

- So, how large is each entry? 64
- And how many entries do you fit in a block? 8
- Can you think of two different ways we might keep track of how many valid entries are in a directory?



```
sfs [13] pwd
/home/ubuntu/cs161/os161-2016/kern/fs/sfs
sfs [14]
```

SFS Idiosyncracy

- Where do inodes live?
  - Anywhere they want!
- Since an inode consumes an entire block, we allocate an inode by asking the allocator for a free block.
- The block number is the inode number.
- The root directory has a designated inode (#1).

Implementing File System Operations

- Given the data structures used to implement SFS, you should now be able to construct most file system operations.
- Let's walk through creating a file.
  - Assume that you have a vnode for a parent directory and you are asked to create file "newfile"
  - What do you have to do?

① Read directory – check for newfile

- error
- not there - byte offset or dirent offset

② Allocate a block for inode ⇒ inode #

(update bitmap)

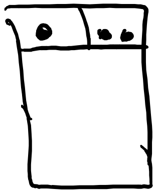
③ Init inode

④ Write dir entry.

Other Operations

1. How would you modify the previous operation to create a directory instead of a file?

   + block allocation
     - initialize block as directory
   + record block in dir inode
   ⊕ write directory entry



2. What do you have to do to delete a file?

   + free disk blocks } update bitmaps.
   free inode
   remove directory