Precept 5: File Systems

COS 318: Fall 2017

Project 5 Schedule

- Precept: Monday 11/27, 7:30pm
  - (You are here)
- Design Review: Monday 12/04
- Due: Sunday, 12/10, 11:55pm

Precept Overview

- File System Support
- Shell Support
- Design Review

File System Review

File System Layout

- Unix-like file system based on xv6
- Disk Layout:

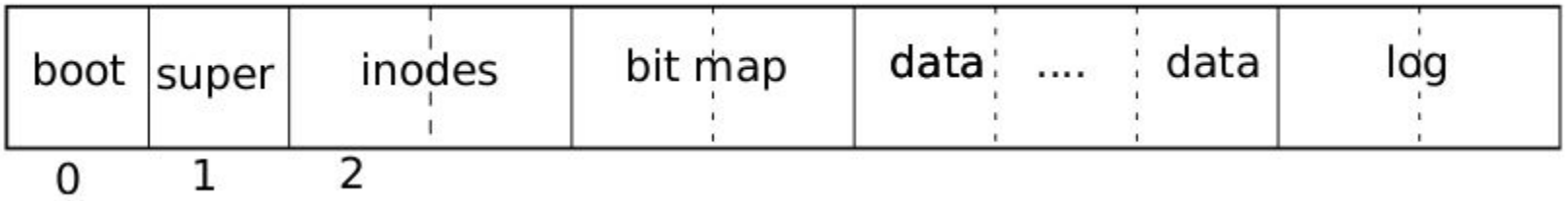

Image Source: xv6 book, Chapter 6

File System Organization

File Descriptor Layer
Path Layer
Directory Layer
Inode Layer
Logging Layer
Block Layer
Buffer cache Layer
Disk Driver Layer

Disk Driver Layer

- Implements driver for disk operations
  - Located in `kern/dev/disk/ide.c`
- `inb`, `outb`: perform low-level port I/O
- Don’t worry too much about this

Buffer Cache Layer

- In-memory cache for disk blocks
  - Disk blocks are held in memory buffers while being used
  - Get written back to disk during recycle
- In comments: `brelse` = `bufcache_release`

Block Layer

- Implements operations on blocks
  - `read_superblock`
  - `block_zero`, `block_alloc`, `block_free`
- mCertiKos specific layer

Log Layer

- Implements logging for crash recovery
  - System calls log all disk write ops on disk
  - Logs “complete” after all expected ops are logged
  - Wipes the log after disk ops actually happen
- On boot: check logs + redo committed operations

Inode Layer

- Inodes hold file metadata
  - E.g. size, # of links, data block addrs, etc.
- `dinode` = on-disk inode, `inode` = in-memory inode: holds additional kernel info
- Indexed by inode number: unique identifier

Directory Layer

- Directories are implemented as inodes
- Its “data” = sequence of directory entries
  - Directory entries: name + inode number
  - Can hold files, or other directories

Path Layer

- Fake Layer
- Helper functions for converting paths to inodes
- You will be writing most of this

File Descriptor Layer

- File Descriptors: Inode wrappers with additional info (e.g. type, R/W, etc.)
- Global table of open files + each process has list of open files
- Layer most of the kernel interacts with

What you are responsible for

Scheduler Changes

- Need a way for threads to sleep on arbitrary resources
  - Each kernel variable has unique address
  - Channel = address of variable / resource
- Implement: `thread_sleep` and `thread_wakeup`

File System Changes

- `dir_lookup` and `dir_link` and `skipelem`
  - Straightforward: follow the directions
- `namex`: ensure you handle locks and edge cases
- Several system call handlers: also straightforward
  - Use `tcb_get_openfiles` as necessary

Inline ASM

- First colon: outputs,
  Second: inputs,
  Third: clobbered registers
- a, b, c, d are x86 registers
- Use simplest solution:
- See this and this for more information

Example:

```
asm volatile("int %2"
    : "=a" (errno),
      "=b" (ret)
    : "i" (T_SYSCALL),
      "a" (SYS_write),
      "b" (fd),
      ...
    : "cc", "memory");
```

Shell Support

Shell Command

- All shell commands are based on the implementation of file system.
- Shell is like a wrapper of the file system.
- Comman: ls, pwd, cd,cp,....
- Should support -r option for cp and rm.

Procedure

- Create a new user process called “shell” in kern/init/init.c.
- Implement system calls to read inputs from users. Feel free to use functions in kern/dev/console.c;
- Then implement each command depending on the input from users.

Design Review

Design Review

1. Explain why not having any locks around the global in-kernel IO buffers is an issue. Clearly explain potential issues with the provided implementation.
2. The core unix shell commands are implemented on top of the underlying file system. Briefly explain what the following commands do, in terms of the file system interface implemented in mCertiKos: `ls`, `cd`, `mv`, `cat`

Questions?