



Precept 6: File Systems

COS 318: Fall 2019

Project 6 Schedule



- **Precept:** Monday 12/09 & Tuesday 12/10, 7:30pm - 8:20pm
 - (You are here)
- **Design Review:** N/A
- **Due:** Tuesday 01/14, 5:00pm (Dean's Date)
 - No late submissions!

Design Document



- No design review!
- Submit pdf describing design decisions + implementation details instead
- Submit with project on Dean's Date
- See project spec for more info

Project 6 Overview



- **Goal:** Implement simple UNIX-like file system
- Manage disk space with dynamic file sizes
- Implement system calls and shell commands to interact with the file system
- Don't worry about concurrency, permissions, or performance



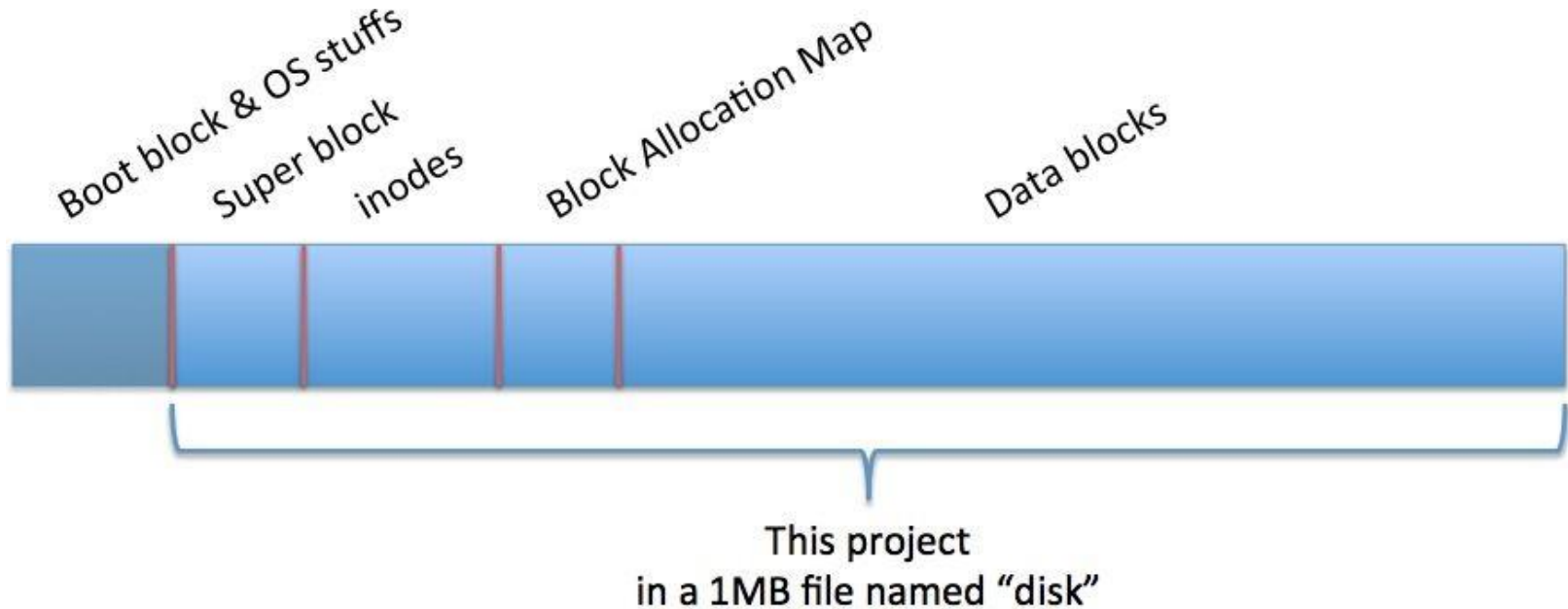
Project Description

API



- Format disk
- File
 - open, close, read, write, seek
 - link and unlink
 - stat
- Directory
 - make, remove, stat, etc.
- Shell commands
 - ls and chdir (cd)

Disk Layout



(Space between divisions not representative of actual size)

Superblock: Disk Metadata



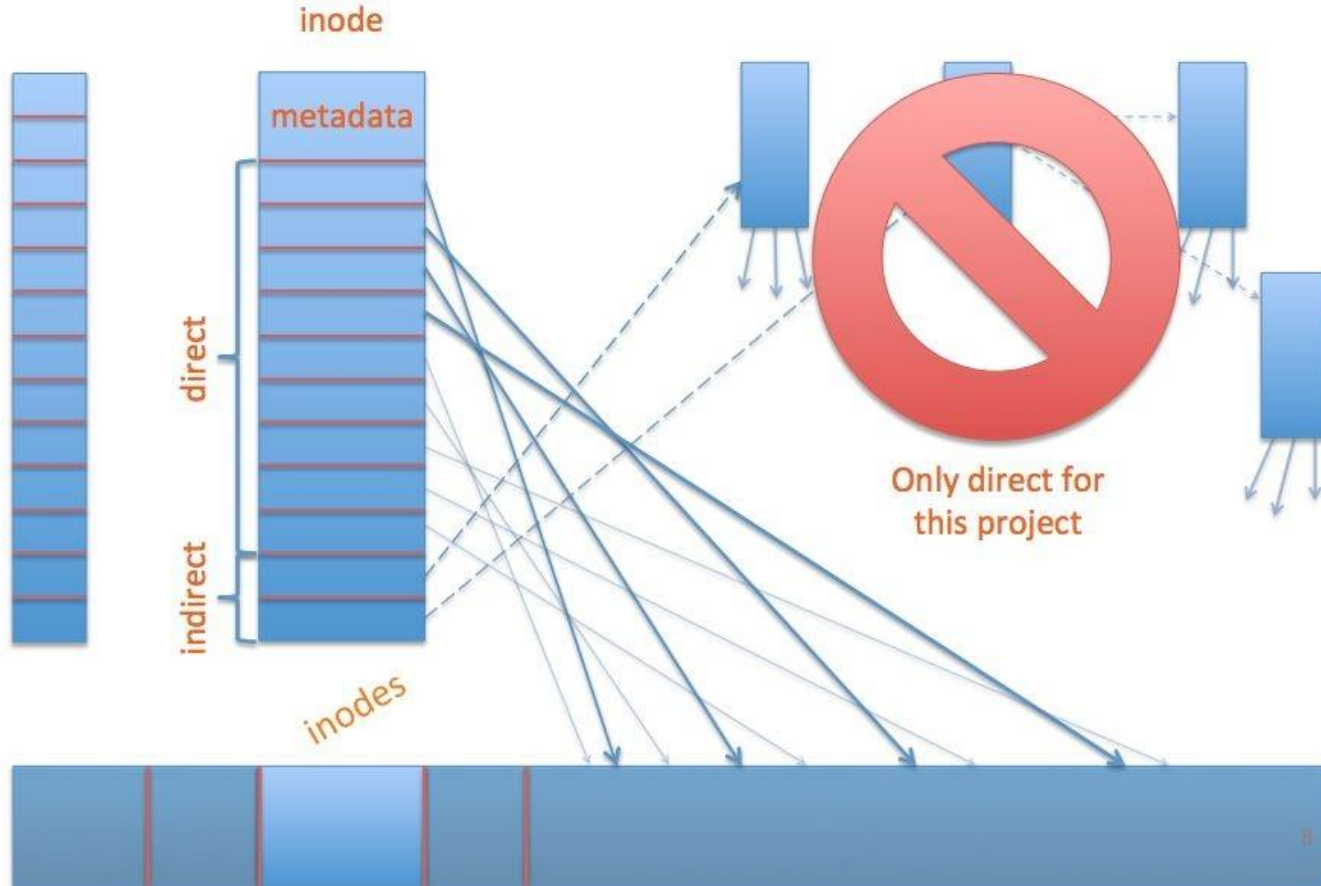
- Examples:

- Size
- Inode /DB start
- # inodes /DBs
- Magic number

Super block



Inodes: File Metadata



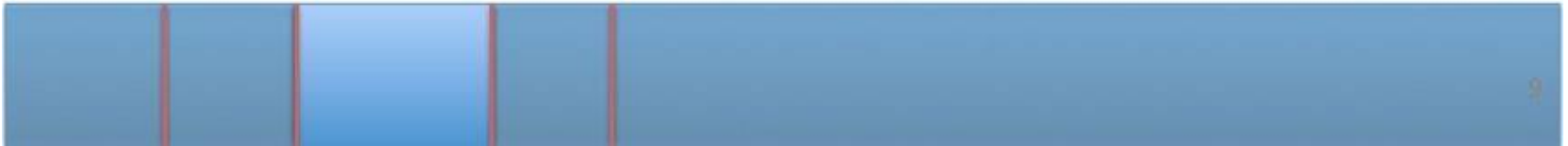


Inodes: File Metadata

- Examples:

- File or dir.
- Link count
- Size
- etc.

inodes



fs_init



- “Constructor” for the FS
- Call `block_init()` to initialize the device
- Init resources used by the FS
- Format disk or mount if already formatted
 - How will you know if disk is formatted?



- **Formats the disk**
 - Write the super block
 - Mark inodes and data blocks as free
 - Create root directory
 - Initialize file descriptor table

File Creation and Deletion



- `fs_open()`: Create a new file if it does not exist
- `fs_link()`: Hard link to an existing file
- `fs_unlink()`:
 - Delete a file if link count == 0
 - Delete directory entry
 - Special behavior if file is still open (look at the project description)

File Access



- `fs_open()`: Open an existing file (allocate file descriptor)
- `fs_read()`: Read bytes from an open file
- `fs_write()`: Write bytes to an open file
- `fs_lseek()`: Change position in a file
- `fs_close()`: Close an existing file (free file descriptor)

`fs_lseek()` Semantics



- In this project, `fs_lseek()` takes only two arguments:
 - file descriptor and offset
- In Unix, `lseek()` takes three arguments:
 - file descriptor, offset, and whence (`SEEK_SET`, `SEEK_CUR`, `SEEK_END`)
- `fs_lseek()` will assume whence == `SEEK_SET`
- What if `fs_lseek()` tries to seek past end of file?
(look at the project description)

Directories - Part 1



- Like a file, but contains a list of files and directories (name to inode number mapping)
- Can read it like a file:
 - Use your file I/O functions (`fs_*`) to do directory manipulation
- Always has at least two entries:
 - Current directory: “.”
 - Parent directory: “..”

Directories - Part 2



- `fs_mkdir()`: **Make a directory**
 - Create a directory entry in parent directory
 - Create the two directories “.” and “..”
- `fs_rmdir()`: **Remove directory if empty**
- `fs_cd()`: **Change the current directory**
 - Only need to implement for relative path names

fs_mkdir() Example



```
int fs_mkdir(char *fileName)
{
    if (fileName exists) return ERROR;
    // allocate inode
    // allocate data blocks
    // set directory entries for "." and ".."
    // set inode entries appropriately
    // update parent
    return SUCCESS;
}
```

Miscellaneous



- You don't need to support absolute path names
- You don't need to support recursive directory removal
- Implement a file system check (`fsck`) tool for debugging that verifies integrity of:
 - a. Superblock magic number
 - b. Block allocations
 - c. Inode allocations
 - d. Block allocation map
 - e. Directory content
 - f. Etc.

Implementation



- In Linux:
 - Uses a file to simulate a disk
 - Code is provided
 - Execute `./lnxsh`
- Shell supports:
 - System calls for file system
 - Commands: “ls”, “cat foo”, “create foo 200”
- You will have to write a lot of code (1,000+)

Testing



- A python script for testing is provided
- Multiple tests that each:
 - Execute the shell
 - Open an existing file system (or format a new one)
 - Write commands to the shell (i.e. “`cat foo`”)
 - Read output from the shell (i.e. ABCDEF)
 - Exit
- You should also write your own test cases
- Submit them with your code



Questions?
