Precept 6: File Systems

COS 318: Fall 2020

Project 6 Schedule

- **Precept:** Monday 11/23 and Tuesday 11/24, 7:30pm – 8:20pm

  Last Precept of the semester 😢

- **Due:** Tuesday 12/08, 5:00pm (Dean’s Date)
  - No late submissions (due to University Policy)

Design Document

- No design review 😮!
- Submit **pdf** describing design decisions+ implementation details instead.
- Submit with project onDean’s Date.
- See project spec for more info
- **5 page LIMIT**

Project 6 Overview

- **Goal:** Implement simple UNIX-likefile system
- Manage disk space with *dynamic file sizes*
- Implement system calls (+1 shell command) to allow shell to interact with the file system.

- Don’t worry about **concurrency, permissions, or performance**

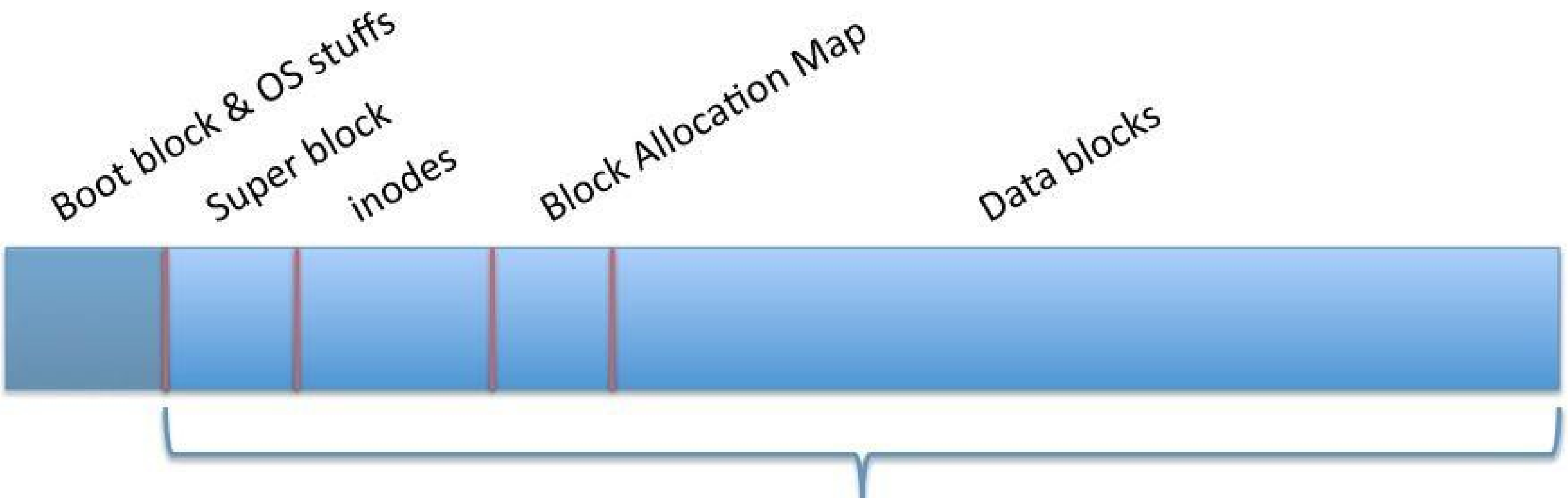
Project Description

API

- Format disk
- File
  - open, close
  - read, write, seek
  - link and unlink

- Directory
  - mkdir, rmdir
  - cd, stat
- “`ls`” shell command

Disk Layout

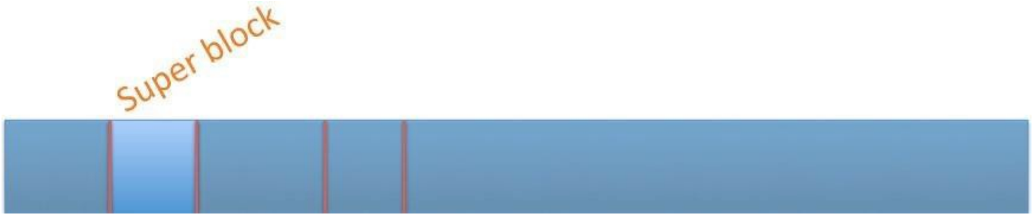


In this project, a FS_SIZE byte file named “disk”

Order and size of sections may change in your system EXCEPT for the Super Block!

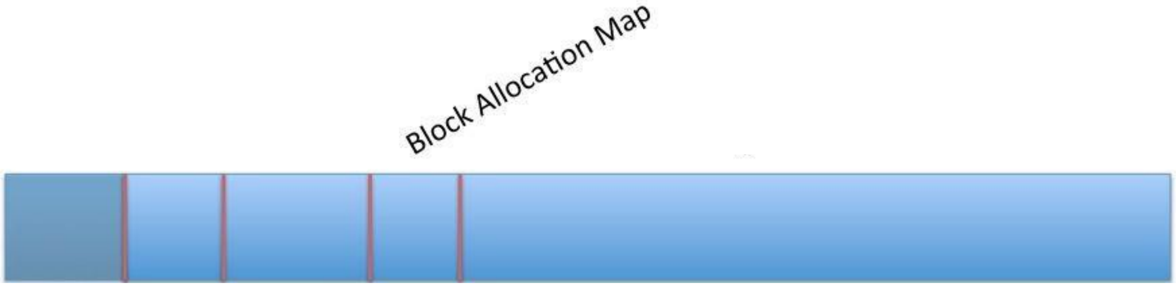
Superblock: Disk Metadata

- First block in the File System (0 in your image)
- Should keep track of:
  - **Magic number**
  - **File System Size**
  - Section information
  - Other info you may find useful.

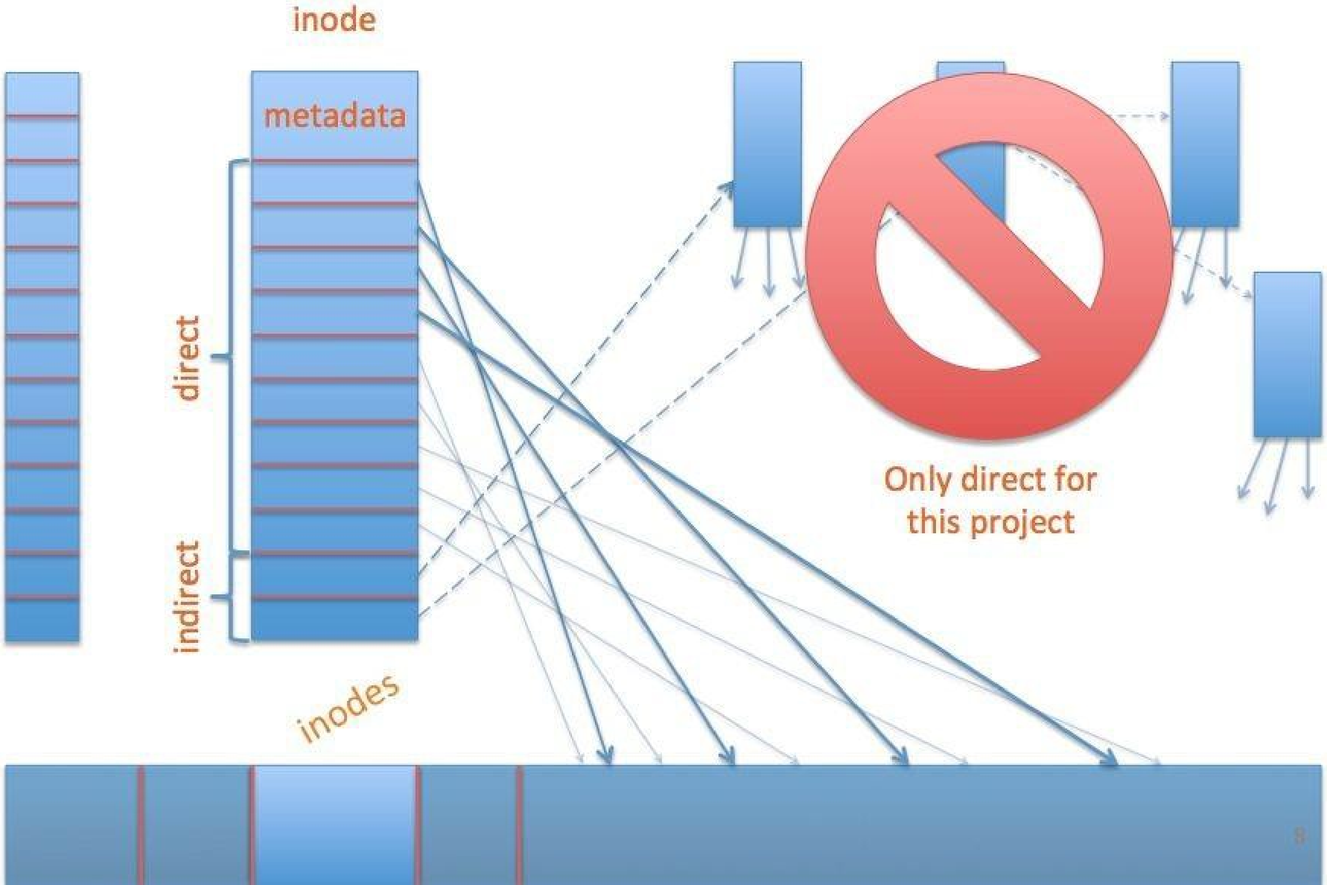


Block Allocation Map

- Keep track of available data blocks.
- Data Structure *(Describe in your Design Document!)*
- HINT: *For every block, you only need to know if it is FREE or IN_USE.*



Inodes: File Metadata

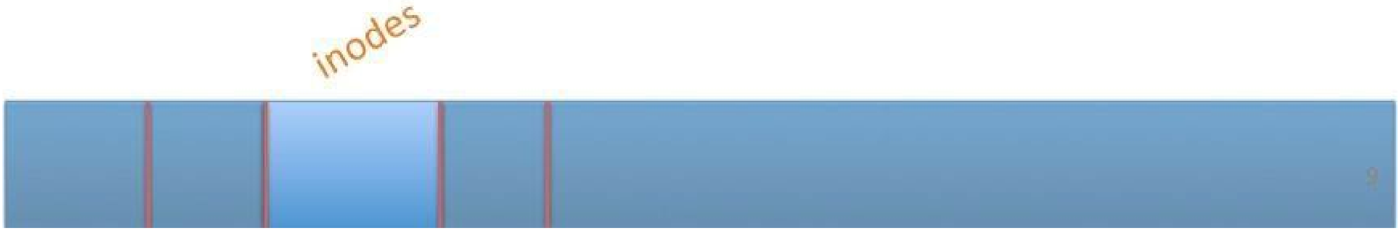


Only direct for this project

Inodes: File Metadata

- Examples:
  - File or dir.
  - Size
  - Link count
  - and possibly more...

(for example, is the inode free?)



System Calls

`fs_init`

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

`fs_mkfs`

- Formats the disk
  - Write a new super block
  - Mark inodes and data blocks as FREE
  - Create root directory (/)
  - Clear File Descriptor Table
- May be called by `fs_init` or directly by the shell!

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()`:
  - Decrease the link_count
  - Remove directory entry
  - Delete file if link_count == 0 and file is not open
  - Open files with no links will be deleted when CLOSED!

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?

  Then move offset. Pad with 0s on `fs_write` past the end.

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: “..” (root points to itself!)

Directories - Part 2

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

Directories - Part 3

- Directories in this assignment can span MULTIPLE blocks.
- Your directories should take the smallest #blocks possible!

- On `fs_unlink` and `fs_rmdir`, if the number of blocks needed goes down, you should change your directory to take less blocks!

- Example, if each entry is 50-bytes and each block is 512-bytes, 11 entries should fit in 2 blocks. Deleting an entry should resize the directory and free 1 block!

fs_mkdir() Pseudo-code

```
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

- All path names are filenames! All operations within current directory
- 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.

Shell

- In Linux:
  - Uses a file to simulate a disk
  - Code is provided
  - Execute `./lnxsh` directly! (That is **NO BOCHS!** ☹)

- Shell supports:
  - System calls for file system
  - Commands: “`ls`”, “`cat foo`”, “`create foo 200`”
  - All comands implemented for you EXCEPT “`ls`”
- You will have to write a lot of code (1,000+)

Testing

- A python script for testing is provided (test.py)
- 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?