Project 3: File Systems
CS 3113

Hard Disks

Hard Disks

- Bytes are organized into blocks
- Blocks are read from / written to the disk at a block level
  - Even if we want a single byte, we have to read the entire block
  - If we want to change one byte on a block, we must read the block, change the byte in memory and then write back out
  - This coarse organization comes from the more general use case: we generallypull large amounts of data off of disks at once

But: we don’t usually think in terms of blocks

File System Abstraction

What are the key features?

File System Abstraction

- File is a long string of bytes
- A file is accessed through a hierarchical directory structure

(and lots of other features)

How do we go from the block to the file system?

Projects 3 / 4

How do we go from the block to the file system?

- We will answer this by implementing our own miniature file system, called “OUFS”
- Project 3: implement directories and directory operations
- Project 4: implement files and file operations

Projects 3 / 4

Virtual Disk:

- A Linux file will act as a virtual disk: 128 blocks of 256 bytes each
- Each block has a unique address: 0 … 127
- Provided: virtual disk operations that read / write one block at a time

OUFS Organization

What do we need? (in terms of data structure)

OUFS Organization

- Inodes
- Content: directory listing or file contents
- Some way of tracking which blocks and inodes are already in use

OUFS Organization: Master Block (0)

Two allocation tables, each is an array of bytes

- Inode allocation table: one bit for every available inode
  - 1 = allocated; 0 = unallocated
- Block allocation table: one bit for every available block on the disk
  - 1 = allocated; 0 = unallocated

Inode

What is the minimal representation?

Inode

What is the minimal representation?

- Type (File, Directory, Undefined)
- Number of references to the inode (number of directory entries that refer to it). For project 3, this will always be one.
- Array of block IDs that are used to store the contents of the inode
  - For our directories: we will only use the 0^{th} block in the array
- Size of the inode
  - Directories: number of contained entities
  - Files: length in bytes

Inode Blocks

- Blocks 1 to N_INODE_BLOCKS are reserved to store only inodes
- A fixed number of inodes fit within each block (INODES_PER_BLOCK)

Data Blocks (remaining blocks)

- Store contents of directories and files

Directory Entries

What do we need to represent a single entry in directory?

Directory Entries

What do we need to represent a single entry in directory?

- Name (string)
- Inode that the name refers to
  - Set to UNALLOCATED_INODE if this inode is not used

Directory Block

- A directory block contains multiple entries
- Each directory inode has exactly one directory block that it uses for contents
  - If all are used, then the directory is “full”

Project 3

Implement:

- An OUFS API (our “system calls”)
- Helper functions for these system calls
- Set of stand-alone executables
  - zformat
  - zfilez
  - zmkdir
  - zrmdir
  - zinspect (provided)