Garrett Bowser
CIS3207 Section 04
Project 4: Simple File System
Due date: 5 December 2019

Overview

This project is a simple, inode-based file system. It writes inode structures to a virtual disk and is able to reconstruct them from the disk them after changes have been made. It supports a small directory hierarchy, the creation, truncation, and deletion of files and directories, among other things. It uses a block size of 4096 bytes and supports a total of 16384 blocks.

Data Structures

Inode

Inodes are the core feature of this project. An inode represents a “file” which can be an actual file or a directory that contains other files. Inodes have several member variables that represent meta-information about the file they represent, such as the file path, size, and the number of blocks that are allocated to it, as well as an array of the block numbers themselves. Inodes that represent directories make use of an array of integers that represent the Inode-ID’s of the files that they contain. Whenever a new file or directory is created, its parent is updated with the new file as a child. New files are assigned a parent upon creation. Each inode can be allocated a total of 256 blocks or contain a total of 256 directory entries. This inode implementation differs from a traditional inode as it does not include indirect pointer blocks. All blocks allocated to the Inode are contained by ID in the Inode’s direct_blocks array.

Design

When a filesystem is created on a clean virtual disk, 256 blank, invalid Inodes, as well as a valid root directory are created in a region of the disk that is equivalent to the “superblock” of a FAT based file system. These Inodes are updated with meta-information and block locations as files are created, deleted, and written to. As new data is written to existing files, empty blocks that exist outside the Inode-region of the disk are allocated to said files. When a new block is allocated to a file, the direct_blocks array in the file’s Inode is updated to reflect the new block that has been allocated to it. Blocks are read from and written to in the order that they appear in the Inode’s direct_blocks array. Upon reading, the contents of each allocated block are concatenated to represent the file’s contents. When an amount of data is written to a file that exceeds the capacity of the last block allocated to the file, whatever amount of data that can be written to the last block is written, and a new block is allocated to which the rest of the data can be written. This procedure continues until all data is written, or the file reaches the maximum number of blocks that can be allocated to it.

Testing

Project 4 was tested by creating a set of test instructions in the fs executable distributed with the project. The tests demonstrate most tasks the filesystem is capable of and illustrate how it can be used to read and write to “real” files on the OS’s disk rather than the virtual disk. Persistence can be verified by either examining the contents of the virtual disk with “hexdump -c vsda” after

running the executable, or by running the executable twice and seeing that errors are displayed where duplicate files are created. Various information is printed as the filesystem allocates new blocks and creates files that show what exactly it is doing.