Filesystem Design Document

CSC 4103: Operating Systems

Programming Assignment 4 – Filesystem Design

Authors: Carter Mauer & Cole Heausler

Instructor: Prof. Aisha Ali-Gombe

Date: May 6, 2025

Overview

This document describes the layout and design of our simple inode-based filesystem implemented in `filesystem.c`, `filesystem.h`, and `formatfs.c`. The filesystem operates on top of a simulated software disk and manages file metadata, data blocks, and directory entries through a series of block allocations and bitmaps.

Disk Block Layout

The software disk is organized into 4096 blocks, each 4096 bytes in size. The layout of the disk is as follows:

Block Range	Purpose
0	Data Block Bitmap
1	Inode Bitmap
2–5	Inodes (4 blocks, 128 inodes/block, 512 total)
6–69	Directory Entries (64 blocks, 8 entries/block, 512 total)
70–4095	Data Blocks for file contents

Key Filesystem Structures

Inodes

Each inode represents a file and contains:
- 13 direct block pointers (16-bit each)
- 1 single indirect block pointer

- A file size field (32-bit unsigned)
This layout supports file sizes up to approximately 8 MB.

Directory Entries

Directory entries are stored in blocks 6–69 and provide the mapping between filenames and inode numbers. Each entry includes:
- A filename (up to 507 characters + null terminator)
- A 16-bit inode number
- A 'used' flag to indicate if the slot is occupied

Implementation Limits

- Maximum number of files: 512 (based on inode and directory entry limits)
- Maximum file size: ~8 MB (13 direct blocks + 2048 indirect blocks)
- Maximum filename length: 507 characters
- Flat directory structure: no subdirectories
- Not thread-safe (single-process access only)
- No support for file permissions or timestamps

Error Handling

The filesystem uses a global error variable `fserror` to record and report the last error. All file operations set this variable, which can be printed using `fs_print_error()`.

Conclusion

This design provides a compact, educationally useful file system that adheres to all assignment requirements, including bitmap tracking, persistent storage, and inode-based block allocation.