Lab 4
File System Design

Darrin McAdams
TUH37195
CIS3207-004
Nov. 28, 2018

Table of Contents

1. Introduction

This project is to design and implement a simple file system in C. The file system should be able to run in user mode and a premade .dat file will be used to emulate the disk. The program must implement the disk and file structure as well as the I/O control for the disk.

1.1. Project Outline

This is an outline of the planned components for the project. It will be used as a guideline during development, but is subject to change during the development phase.

- filesystem.c
  - The core program that will create the file system.
  - Will provide basic functionality for users to test and use the file system

- file.h
    - Responsible for implementing basic file structure and functionality.
    - Has the inode and directory structs
    - Implements the create_file() function
- disk.h
    - Responsible for implementing the disk structure
    - Contains the I/O Control System functions
- README.txt
    - Contains everything a user should need to know to run the file system program
    - Will also contain some useful information about known bugs and limitations

1.2. Project Description

This file system should meet the following requirements:

- The File System:
    - The file system should support directories and sub directories.
    - Files should be able to have at least 8 character names and 3 character extensions.
    - Files can be up to 4kb in size.
    - Files should not have to be contiguous in memory.
    - Memory should be allocated in 512 byte blocks.
- I/O Control System:
    - The I/O controls system should have the following functions:
        - Create File
        - Read File
        - Write File
        - Open File
        - Close File
        - Delete File
    - These functions should be the primary means of manipulating files within the file system.

2. The Disk

			DATA BLOCKS							
SUPERBLOCK	INODE TABLE	DIRECTORY TABLE	1	2	3	4	5	6	7	8
			9	10	11	12	13	14	15	16
			17	18	19	20	21	22	23	24

2.1. Overview

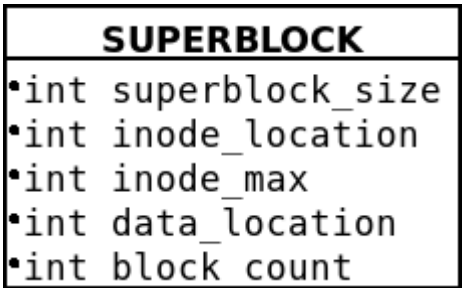
The disk should be comprised of the following parts:

- The Superblock
  - Max Inodes
  - Total available blocks
  - Superblock, Table, and Data starting locations
  - Other information needed to properly use the disk
- The Inode Table
  - Contains all of the Inodes in the file
- The Directory Table:
  - Holds all of the directories and their links
- The Data Region
  - Contains all of the blocks used for data storage

2.2. Design Guidelines

The disk will actually be a file where all of the bytes are sequential. This means that concepts like disk rotation and seek time can mostly be discarded. The general layout of the disk must still be enforced by the file system program. This can be achieved by knowing the exact size of each piece (blocks, inodes, superblock, etc) and having a predefined byte range for those objects to exist in.

3. The Disk



3.1. Overview

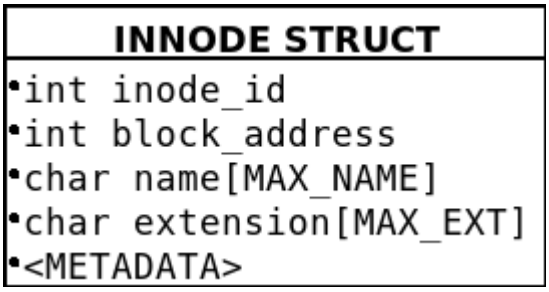
The superblock should contain the following attributes:

- int superblock_size
  - Contains the size of the superblock in bytes
- int inode_location
  - The byte location for the start of the inode table
  - should be equal to the superblock_size + 1
- int inode_max
  - Contains the maximum number of inodes allowed on the disk
- int data_location
  - Contains the byte location for the start of the data blocks
  - should be equal to superblock_size + (inode_count * the size of the inode) + 1
- int block_count
  - Contains the number of blocks on the disk

3.2. Design Guidelines

The superblock contains all of the basic information required for the disk to operate. This includes the starting location for each portion of the disk and number of blocks available. It might also be necessary to implement an array of bytes to represent each block on the disk for checking if a block is in use. This would be added to superblock.

4. The Inode



4.1. Overview

For this project the Inode struct should contain the following attributes:

- int inode_id
  - A unique identifier for the Inode
- int block_address
  - The address of the first data block used by the file
- char name[MAX_NAME]
  - Contains the name of the inode
  - MAX_NAME must allow for names to be at least 8 characters long
- char extension[MAX_EXT]
  - Contains the extension used by the file
  - MAX_EXT must allow for extensions to be up to 3 characters long
  - The .DIR extension will be given special treatment for organizing files.
- <METADATA>
  - Other attributes will be used to track data about the file, such as the date created or last modified.

4.2. Design Guidelines

The Inode is a struct that contains all of the necessary information about a file. The most important of these is the address for the starting block for the file. The first block should contain pointers for all of the other blocks used by the file. Since each block has 512 bytes, and each int is 8 bytes, the first block should be able to hold the address for another 64 blocks, which meets the 4kb file requirement. For inodes with the .dir extension, the first block should point to other Inodes.

5. I/O Control System

5.1. Overview

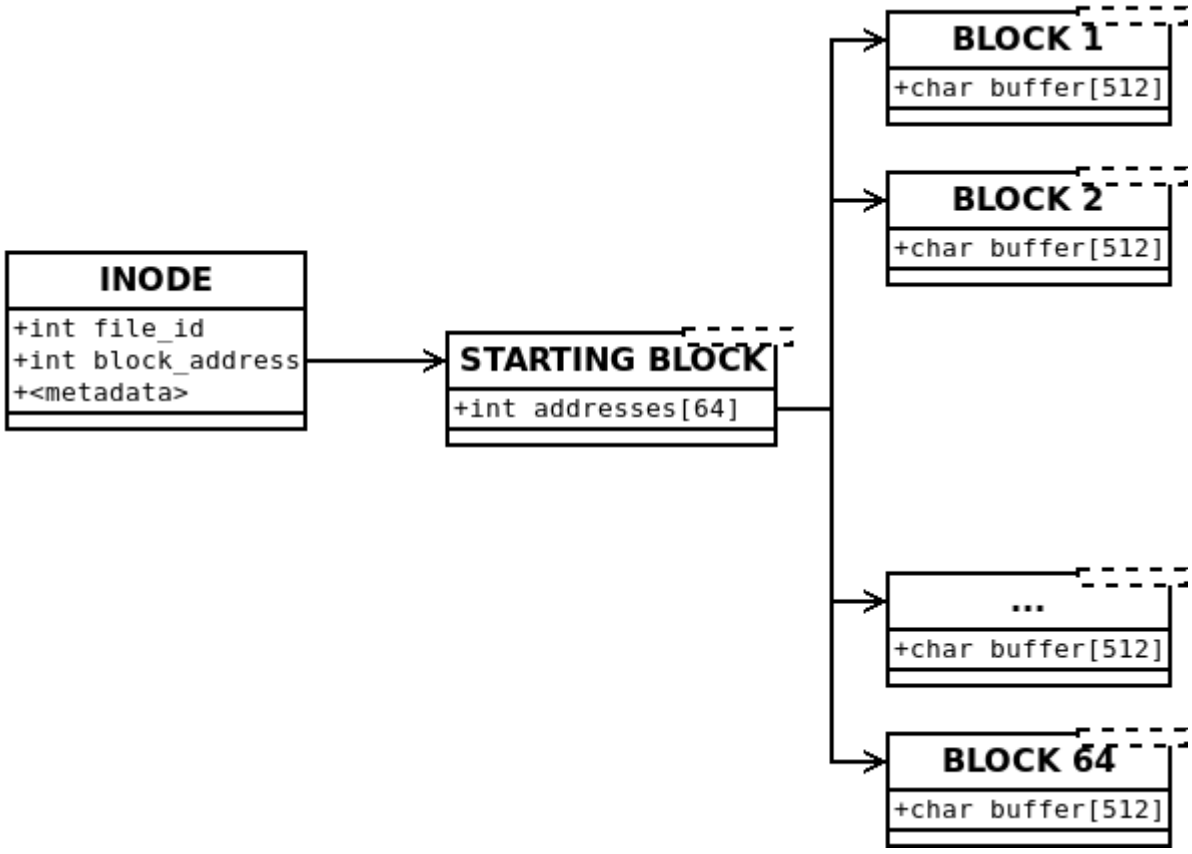
The I/O Control System should contain the following functions:

- fs_open
  - Finds and opens a file stored on the disk
- fs_close
  - Closes an opened file
- fs_delete
  - Deletes a file found on the disk
  - fs_delete should not be able to delete files that are currently opened
- fs_read
  - Reads the file's data stored on the disk
- fs_write
  - Write changes made to the file to the disk.
- misc. helper functions
  - helper functions that are needed for the core functions in the Control System
  - Possible functions:
    - fs_search: searches for a file on the disk
    - fs_block_read: reads individual blocks and returns them as a string.
    - fs_block_write: takes a string and writes it to a block

5.2. Design Guidelines

The I/O Control System is responsible for manipulating data on the disk and maintaining the basic structure of the disk. Special care should be used here to ensure that data is handled properly and that the disk's design constraints are strictly enforced.

6. File Storage



6.1. Overview

File storage is an essential part of this project. The basic premise is that each inode contains a pointer to a single starting block. This starting block will contain the address of every other block that contains the file's data. Since each block contains 512 bytes and an int takes up 8 bytes, this starting block should be able to contain the address of up to 64 data blocks. This also means that a file can have up to 4kb of data, which meets the minimum requirements for this project. This concept can be extended out so that the starting block can point to a set of address blocks, increasing the maximum file size to 256kb. Using this technique the maximum file follows the following equation: $m = 4kb * 64^n$ where m is the max file size and n is the number of nested address blocks.

6.2. Directories

Directories will also be stored as an inode. There are only two key difference between a directory and a file. The first one is that the directory will be identified by the .dir extension. The second key difference is that the starting block will point towards other inodes instead of data blocks. This means that the directories will be affected by the same limitations as regular file storage, and can only hold up to 64 files. Like with regular file storage, this can also be expanded on by nesting address blocks.

7. Psuedocode

This is the psuedocode for various parts of the the files system program. It should be used as a starting template and reference when writing the code for the file system.

7.1. file.h

```
typedef struct Inode{
        int inode_id;
        int starting_block;
        char *name[MAX_NAME]
        char *extension[MAX_EXT}
        int date_created;
        int date_modified;
        //other metadata here
} inode;

inode *create_file(<attributes>){
        //allocate the memory for the new inode
        inode *n = malloc(sizeof(inode));

        //set the supplied <attributes> here

        return n;
}
```

7.2. disk.h

```
typedef struct Superblock{
        int inode_location;
        int inode_count;
        int inode_max;
        int data_location;
        int block_count;

} superblock;

//create a superblock

superblock *create_super(<attributes>){
        //allocate the memory for the superblock
        superblock *sb = malloc(sizeof(superblock))

        /*
        set the predefined <attributes> here
        */

        /*
        calculate the rest of the required attributes here
        Ex: blocks = (disk_size – superblock_size – inode_table_size) / block_size)
        */

}
//open a file
fs_open(char *disk_name, int file_location){
        //open the disk, don’t forget to check the return value
        FILE *disk = fopen(disk_name, “r”);
        //move to specified location. Again, don’t for get to check the return value.
        fseek(disk, location, SEEK_SET);
        //create an inode to hold the file information
        inode *n = malloc(sizeof(inode));
        //read the file from the disk
        fread(n, sizeof(inode), 1, disk);
        //close the disk file
        fclose(disk);
        //return the file
        return n;
}
```

```
//close a file
void fs_close(inode *n){
	//free the memory for the inode
	free(n);

}

fs_delete(int location){
	/*
	1.) use file location to get the starting block
	2.) use starting block to get a list of blocks used by the file
	3.) set all blocks to NULL
	4.) set inode to NULL
	5.) return
	*/
}
//write a file to the disk
fs_write(inode *n, int location, char *buffer){
	/*
	1.) find or add inode to the inode table as needed
	2.) get size of buffer
	3.) update inodes's size and blocks used if needed
	4.) if needed, add or remove block addresses from the file's starting block
	5.) break up buffer into 512 byte chunks
	6.) write each chunk to a block.
	*/
}

fs_read(inode *n){
	*/
	1.) get the inode's start block
	2.) get all data block addresses from the start block
	3.) allocate memory for a buffer to hold a file's data
		3a.) size should be equal to 512 * number of blocks
	4.) copy each block byte by byte into the buffer
	5.) return the buffer
	*/
}
```

7.3. filesystem.c

```
//add all required definitions here
int main(){
	//initialize the disk
	init();
	//start main user interface
	start();
}
//handles all of the required setup for the program
init(){
	/*
	1.) check for superblock on the disk
	2.) if no superblock, create one
}

start(){
	/*
	This function should provide users with a basic command line prompt to
	navigate and use the file system. Ideally, this would use some code from Lab
	3: Shell to provide a nice interface for the user. All other functions should be
	accessed, either directly or indirectly, from here.
	*/
}
```