CSC-415 Operating Systems

File System Project

*Milestone 1*

Team:

Diligence

GitHub:

<https://github.com/CSC415-2022-Fall/csc415-filesystem-mkim797>

|  |  |
| --- | --- |
| Name | Student ID |
| Mark Kim | 918204214 |
| Peter Truong | 915780793 |
| Zeel Diyora | 920838201 |
| Chengkai Yang | 921572896 |

Table of Contents

[File System Hexdump: 2](#_Toc117695180)

[VCB Structure: 2](#_Toc117695181)

[Free Space Structure: 2](#_Toc117695182)

[Directory System: 2](#_Toc117695183)

[Table of who worked on what: 3](#_Toc117695184)

[Teamwork 3](#_Toc117695185)

[Issues Faced 4](#_Toc117695186)

# File System Hexdump:

Attached?

# VCB Structure:

// This is the volume control block structure for the file system

typedef struct

{

int number\_of\_blocks; // number of blocks in the file system

int block\_size; // size of each block in the file system

int freespace\_loc; // location of the first block of the freespace

int freespace\_first; // reference to the first free block in the drive

int freespace\_avail; // number of blocks available in freespace

int freespace\_size; // number of blocks that freespace occupies

int root\_loc; // block location of root

long magic; // unique volume identifier

} VCB;

The volume control block is still a work in progress and we will be adding more information as the need arises. Nevertheless, the current structure contains the variables as shown above.

# Free Space Structure:

Our free space is utilizing a File Allocation Table (FAT). The VCB freespace\_first variable is the integer that points to the first free block in the file system. Each subsequent entry in the array then points to the next in the sequence of free blocks available until the end of the file or freespace map. The last block of each file (including the free space map) is set to 0xFFFE to indicate the end of the file. The first block (block 0) is reserved for the VCB, while the second block (block 1) is reserved for the free space map. The free space map then takes up the number of blocks according to the needs of the caller of the file system initialization.

One item of concern is that each element of the FAT seems to be taking up 8 bytes rather than 4 bytes for each entry. This is something to ask about and investigate further.

# Directory System:

// This is the directory entry structure for the file system

// This struct is exactly 63 bytes in size, but is padded by

// 1 byte.

typedef struct

{

char name[32]; // name of file

time\_t created; // time file was created

time\_t modified; // time file was last modified

time\_t accessed; // time file was last accessed

unsigned int size; // size of the file in bytes

short loc; // block location of file

char attr[1]; // attributes of file (1: directory, 2: file)

} DE;

Like the VCB, the Directory Entry struct is a work in progress. Currently, it contains a filename that is 32 bytes long, but it has come to our attention that perhaps the assignment requires a 128 byte long filename. There are three time\_t entries for the times the file was created, modified, and accessed which are 8 bytes each. We used unsigned int for the size since size is never negative and a short for location, with a final byte as a file attribute. A short data type was used to be 2 bytes and the attribute is only 1 byte. By using these values for the directory entry, we come to 63 bytes, but because of padding, exactly 8 entries can be fit in each 512 byte block. We will modify this accordingly if we find that we are required to have a 128 byte filename (or if we find we need more data in the directory entry.

Our directory system is handled similarly to the free space map. The directory entry points to the first block of the file in the FAT and each subsequent block points to the next block in the chain until it reaches the end of file which is flagged as 0xFFFE. A directory is allocated with an initial size of 40 directory entries which currently fills exactly 5 blocks in the file system.

# Table of who worked on what:

|  |  |
| --- | --- |
| *Name* | *Component worked on* |
| Mark Kim | initFileSystem function, free space initialization, root directory initialization, assignment writeup |
| Peter Truong |  |
| Zeel Diyora |  |
| Chengkai Yang |  |

# Teamwork

The team met up approximately 3 times during this portion of the assignment (once or twice a week). We supplemented the meetings with discussions on Discord. We divided the tasks democratically by asking each member to volunteer to different parts. Chengkai was to handle the VCB struct, DE struct, and the logic determining whether the file system needed to be initialized or not. Peter was to handle the initialization of the volume control bock and write the necessary values to it, then LBAwrite the data to block 0. Zeel was to handle free space initialization. Finally, Mark was responsible for root directory initialization. The team was to request assistance as needed from other members if issues were encountered or to clarify needs for each section.

# Issues Faced

The biggest issue we faced was a breakdown in communication. Because each portion of the assignment had dependencies on other portions of the assignment, communication between team members needed to be prompt. Unfortunately, because of this lack of communication, important initial pieces were not being completed so that other team members could start their tasks. In the end, because of the looming deadline (and other pressing upcoming obligations), Mark had to implement everything necessary to get to the portion of the assignment for root directory initialization.