Jacob Saltzman

2 May 2017

CS 4414: Spring 2017

**Project 4: Designing a Virtual File System**

**Introduction**

The purpose of this project was to design an implement a virtual file system with the following specifications:

* 512 bytes of storage
* Support for up to 8 files, max 512 bytes
* 4 ASCII-character filenames
* 4 simultaneously open files
* Block size of 16 bytes
* 1024 bytes / 64 block total size

**Implementation**

The file system was designed following the guidance provided by the instructor with a few small modifications. The basic design utilized a FAT (file allocation table), as well as an open file table.

*Disk Layout*

The bits were stored on disk as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Description** | **Start (incl.)** | **End (incl.)** |
| Superblock | Size: 1 block. Holds basic information about location of disk contents including FAT, directory, and storage | Byte 0  Block 0 | Byte 7  Block 0 |
| Directory | Size: 8 blocks. Each block holds metadata for each file, consisting of:   * Used/Free flag (1 byte) * First block number (2 bytes) * Filename, 4 chars w/ null termination (5 bytes) * File length, in bytes (2 bytes) | Byte 8  Block 1 | Byte 71  Block 8 |
| File Allocation Table | Size: 8 blocks, each holding four 4 byte entries. Used to correlate data blocks to specific files Each FAT entry corresponds to one of the 32 data blocks, and contains:   * Used/Free flag (1 byte) * Index of file in directory (3 bytes) | Byte 72  Block 9 | Byte 255  Block 15 |
| < unused > |  | Byte 256  Block 16 | Byte 511  Block 31 |
| Data | Size: 32 blocks. Used as raw storage with no flags or metadata. | Byte 512  Block 32 | Byte 1023  Block 63 |

*Memory Layout*

When mounted, the filesystem on disk was used to populate corresponding C data structures which could be accessed and mutated via API calls. Additionally, in memory the filesystem contained an OFT (open file table) to keep track of file descriptors and offsets. Serialization methods were written for each of the data structure to translate C structs to raw bytes suitable for writing to disk.

The data structures to implement are shown below.

**typedef** **struct** {

uint16\_t dir\_start\_byte;

uint16\_t dir\_end\_byte;

uint16\_t fat\_start\_byte;

uint16\_t fat\_end\_byte;

uint16\_t data\_start\_byte;

uint16\_t data\_end\_byte;

uint32\_t unused;

} SuperBlock\_t;

**typedef** **struct** {

uint8\_t is\_used;

uint16\_t first\_block\_num;

**char** filename[5];

uint16\_t file\_len;

uint32\_t unused0;

uint16\_t unused1;

} DirectoryBlock\_t;

**typedef** **struct** {

uint8\_t is\_used;

uint16\_t file\_index;

uint8\_t unused;

} FATBlock\_t;

**typedef** **struct** {

**char** data[DATA\_BLOCK\_SIZE];

} DataBlock\_t;

**typedef** **struct** {

SuperBlock\_t super\_block;

DirectoryBlock\_t directory\_blocks[DIRECTORY\_SIZE];

FATBlock\_t fat\_blocks[FAT\_SIZE];

DataBlock\_t data\_blocks[DATA\_SIZE];

} FileSystem\_t;

**typedef** **struct** {

uint8\_t open[OFT\_SIZE];

DirectoryBlock\_t\* dirptr[OFT\_SIZE];

uint16\_t offset[OFT\_SIZE];

} OFT\_t;