David A. Fontana 29 November 2017

FILE SYSTEM IMPLEMENTATION

I. DISK LAYOUT

- The File System scales its reserved partition based on Disk and Block Sizes.
- The FAT uses 4N/BLOCK_SIZE blocks, allowing for addressing of entire disk
- File Records use 1% of Disk Space

- Block 0 is Reserved for File System metadata which includes:

numFatBlocks (bytes 0-3) - # of disk blocks allocated to the FAT
numRecordBlocks (bytes 4-7) - # of disk blocks allocated to File Records
numDataBlocks (bytes 8-11) - # of disk blocks allocated for Data
firstFatBlock (bytes 12-15) - Block Index of the first FAT Block
firstRecordBlock (bytes 16-19) - Block Index of the first File Record Block
firstDataBlock(bytes 20-23) - Block Index of the first Data Block

II. FILE ALLOCATION STRATEGY

- The File System utilizes a 32-bit (4 Byte) FAT allocation strategy.
This allows for easy indexing with 32-bit unsigned integers
(All FAT entries are 32-bit wide, allowing for addressing of 2^32 blocks)

Two special FAT entries (0x0000000 and 0xFFFFFFFF) represent free blocks and terminating blocks, respectively.

- File metadata (including name) is held in 32 BYTE wide File Entries and has layout:

Byte 0 (leftmost) – File Attributes
Bit 0 – File Record is Present (Set on Create, Unset on Delete)
Bit 1 – File Record is Parent (Concurrent non-parent records used for File Names)
Bit 2 – File is Open (Set on Create, Open. Unset on Close.)
Bit 3 – Not Used
Bits 4-7 – Used for concurrent record indexing.

Byte 1 – Unsigned Integer Index of first FAT entry for File
Byte 2 – Unsigned Integer of File Size
Bytes 9-32 – File Name (23 Characters)

- Long Filenames are accommodated by allocating contiguous record clusters, up to 15 (345 character filenames)

- File Entries are located on disk in the Block immediately behind the FAT table.

III. Design Considerations

- File System Information

As covered above, these are fields that keep track of filesystem metadata, including offsets to the FAT, File Entry, and Data sections.

Currently FSInfo is implemented as a non-global, accessible only through the get_fs_info function. Given more time I would change this to allow easier access across functions.

- File Internals

Each FileInternals struct holds the following fields:

unsigned int recordNumber - Used for quickly accessing File Attributes
unsigned int fileSize; - Stores File Size while in-use
unsigned int filePos; - File Position Pointer (absolute, in-bytes)
unsigned int startingBlock; - First Data Block (First FAT Index) of File
unsigned int currentBlock; - Index number of Block into which filePos points
FileMode mode; - READ_ONLY, READ_WRITE enum

- Read/Write

The project instructions and documentation did not specify the particular functionality of the Read / Write functions. Unfortunately, my original implementation did not increment the File Position pointer after Read/Write functions (similar to Project 1 from this semester).

As a result, the Read/Write function implementation is still very buggy.

IV. Internal Function Overview

- A Quick Overview of the internal FileSystem functions that have been implemented:

struct FSInfo get_fs_info();
Returns a pointer to the FSInfo structure, as covered above

unsigned int find_file(char *name);
Used as an extension of the file_exists() function, this provides the Record Number of a file matching ‘name’. Useful for quick Open/Create requests. Returns 0xFFFFFFFF if File Does Not Exist.

unsigned int is_open(unsigned int recordNumber);

Used to quickly determine if a File is currently open, given a Record Number (available in the FileInternal included with most API calls)

unsigned int get_free_data_block();

Returns first empty FAT entry (0x00000000). Very simplistic, could be optimized.

Unsigned int get_free_record(unsigned int length);

Needed due to tricky nature of long filenames with FAT. Scans File Entry records looking for ‘length’ number of records beginning with a 0x00000000 FileAttribute byte. Returns the Parent (top-most record) index number.

Unsigned get_next_data_block(unsigned int parentIndex);

Essentially just following the FAT pointer to next FAT block, but useful when looking for EOF during Read/Write/Seek.

Unsigned int update_file_size(unsigned int recordNumber, unsigned int entryValue);

Lazy write to the FileSize field in the File Record ‘recordNumber’

unsigned write_fat_entry()

Unused, previously for testing

unsigned write_record_entry(char *name, unsigned int dataBlock)

Constructor for a File Record entry. Responsible for building the FileAttribute byte and appending correct sub-indexes. “dataBlock” is the index of the first Data Block for file.

unsigned int allocate_data_block(int *parentFatIndexPtr, int targetFatIndex);

Most useful private function. Since Data blocks are only ever allocated during a Create, Seek, or Write this allows quick updating of the “parent” or “head” block to point to the “target”.