Operating Systems : Project 3 - Filesystem

DUE DATES

1. **DESIGN: Submitted to Canvas as PDF by Friday 03/25 @ 5:00 PM**
2. **Milestone 1: Friday 04/08 @ 11:59 AM**
   Base: P3M1_BASE **Submission: P3M1**
3. **Milestone 2: Friday 04/15 @ 11:59 AM**
   Base: P3M2_BASE **Submission: P3M2**
4. **Milestone 3: Friday 04/22 @ 11:59 AM**
   Base: P3M3_BASE **Submission: P3M3**

1. Introduction:

The task for this project is to use the block storage library created in Assignment 2 as a logical storage device, on top of which you are now going to build the **S16 Filesystem (S16FS)**. **S16FS** will closely model the original Unix Filesystem and the Unix Fast Filesystem.

Please review your class notes. Additionally, these references may be useful:

- https://www.cs.berkeley.edu/~brewer/cs262/FFS.pdf
- http://en.wikipedia.org/wiki/Extended_file_system
- http://www.cs.cornell.edu/courses/cs6410/2010fa/lectures/04-filesystems.pdf
- https://en.wikipedia.org/wiki/Ext2

2. Filesystem Design:

Files and filesystems are designed to operate using an abstraction of the underlying storage hardware. You will implement a filesystem that interfaces to the ***Back Store*** of Assignment 2. Recall the *Back Store* library specifications; specifically the purpose of the library is to simulate a block storage device that provides access to storage blocks based on a block index.

The following are the sizing constraints of the *Back Store*:

1. The total number of blocks is 2^16.
2. The size of blocks is fixed at 1 KB, i.e., 1024 bytes.
3. Some of these blocks are pre-allocated to the free block map.

3. Milestones:

This project will be divided into four submissions:

1. **Design**,
2. **Milestone 1**,
3. **Milestone 2**, and
4. **Milestone 3**.

After Milestone 1 and 2, example code will be provided for you to use as you wish.

Design:
A failure to plan is a plan to fail. This will be a large project; understanding the basic concepts, planning your structures, functions, and what issues can occur is **vital** to any programming project.

You are to draft pictorial representations of (drawings on paper and scanned/photographed are sufficient):

1. Inodes and their block references
2. The filesystem layout, and how its layout relative to Back Store.

You are to put together your proposed layout (read: structure definitions) for:

1. S16FS objects
2. Inodes
3. Directory files

You are to write pseudocode for the following problems given the specified information, and list at least three errors that can occur during (other than allocation failing):

1. File/Directory Creation given an absolute path to the file/directory and a flag indicating file or directory
2. File Writing given a **file descriptor,** a buffer, and a byte count
3. File/Directory Deletion given an absolute path to the file/directory

Once you have completed your draft designs, structure definitions, and pseudocode; collect them into a PDF document and submit them to Canvas by **Friday 03/25 @ 5:00 PM**.

Milestone 1:
You are to implement formatting, mounting, and unmounting (fs_format, fs_mount, and fs_unmount) of your S16FS filesystem. In addition to this, you will implement the creation of empty files and directories (fs_create).

Once you have completed your implementation, be sure to commit all changes, tag your commit with the appropriate submission tag (**P1M1**), and push to GitHub by **04/08 @ 11:59 AM**.

Milestone 2:
You are to implement the opening, closing, removal, and writing of files (fs_open, fs_close, fs_remove, fs_write).

Once you have completed your implementation, be sure to commit all changes, tag your commit with the appropriate submission tag (**P1M2**), and push to GitHub by **Friday 04/15 @ 11:59 AM**.

Milestone 3:
Your final milestone will implement the reading and seeking of a file as well as the enumeration of directories (fs_read, fs_seek, and fs_get_dir).

Graduate students will also need to implement moving files and directories (fs_move).
This will also be bonus for undergrads.

Once you have completed your implementation, be sure to commit all changes, tag your commit with the appropriate submission tag (**P1M3**), and push to GitHub by **Friday 04/22 @ 11:59 AM**.

4. Filesystem Specification, Implementation, and API:

Your implementation of the SP16 Filesystem must be a CMake system library project. A header with the expected functions will be made available after the Design due date.

Specifications:

Your S16FS implementation must be capable of:

- Format an S16FS file
- Mount and unmount an S16FS file
- Create directories and regular files
- Open, close, and seek files
- Read and write to files
- Remove files
- List directory contents

The S16FS filesystem will incorporate file descriptors, and they will be used much like they are in the OS. Open, close, seek, read, and write, instead of working on file names, will work on descriptors maintained by your implementation.

Structuring the S16FS

Regarding inodes:

Each inode will be exactly 128 bytes. The general inode structure is as follows:

Filename: bytes 0-63
Metadata / ACL: bytes 64-112
Data Block Pointers: bytes 112-127

The inode table will be 32 kilobytes in size, i.e., 32 data blocks worth of inodes. Inodes will use 16-bit addressing for block references. Your filesystem root directory must be located at the first inode.

Regarding block pointers:

Inodes contain 8 block pointers: 6 direct, 1 indirect, and 1 double indirect. Direct pointers, as the name suggests, are the indices of the data blocks that make up the file. Indirect pointers point to a data block that is made of direct pointers. A double indirect pointer points to a block of data that is made of indirect pointers. Indirect and double indirect pointers greatly increase the maximum size of a file while the direct pointers allow for simple access to small files.

Regular File Block Structure:

The regular file blocks have no structure, they are pure data. Be sure to keep track of the file's size somewhere in your inode metadata!

Directory File Block Structure:

To simplify the filesystem, directory files will be limited to 15 entries. Each entry will consist of:

1. The file's name: 64 bytes
2. The file's inode number: 1 byte

This allows a directory to be contained on a single data block, the remaining space on the directory's data block should be allocated to Metadata / ACL as you see fit.

File Descriptors:
The S16FS filesystem will be using file descriptors to manage open files. The filesystem will be limited to **256 file descriptors**. Each file descriptor will track a single read/write position for that descriptor. How you manage this is up to you. Files should be able to be opened multiple times, meaning your implementation should support multiple descriptors to the same file, but with different read/write positions. Seeking allows the user to move the read/write position to where they want. Be sure to read the header for additional notes or requirements on how a function affects a file descriptor.

Testing:
Your implementation should be capable of doing all operations listed above cleanly and safely. Like all projects, a tester will be provided to test the frontend of your implementation. You will be expected to deliver a functional solution for each milestone. Be sure to be comfortable with various debugging and file inspection tools, and they will prove vital to writing a correct implementation.

System Library:
The library should be named: `libSoneSixFS.so` or `libSoneSixFS.a`

Be sure the install process installs the library module as well as the interface header.

5. Rubric:

Design	30 points
Milestone 1	70 points
Milestone 2	110 points
Milestone 3	90 points
Total	300 points
fs_move	+15 points

For all students, for all milestones:

Insufficient Parameter Validation: up to -25% of rubric score
Insufficient Error Checking: up to -25% of rubric score
Insufficient Comments: up to -25% of rubric score
Incorrect implementation: up to -50% of rubric score
Memory Leaks: up to -20% of rubric score
Submission compiles with warnings: -90% of rubric score
Submission does not compile, or refuses to build due to cmake issues: -100% of rubric score