CS 350 Spring 2016 **Program 2 (and Lab 7)**

For this assignment, you will emulate a Log Structured File system (LFS). Users will be able to import real linux text files into your file system, and will be able to view, update, and delete them. Your program will be able to exit, restart, and load in the files it had stored previously. Details follow.

"The Hard Drive"

Your file system will use a single linux directory and some contained *segment files*, as its back end disk drive. This drive will consist of 32 segments, each of which contains up to 1MB of data. The directory should be called "DRIVE" and the segment files should be called "DRIVE/SEGMENT1", "DRIVE/SEGMENT2", … "DRIVE/SEGMENT32".

Write a small program that will create the DRIVE directory and the 32 contained segment files, each containing 1MB of NULL data (i.e. zeroes). Segments should contain disk blocks of size 1K.

Your segment files should contain segment summary blocks within the 1MB of data itself.

The File System

Your Log Structured File System (LFS) is a single program that runs and accepts file system commands from a user. In particular, the program should respond to the following commands:

`import <filename> <lfs_filename>`

This command causes the linux file named `<filename>` to be read and stored in your LFS file system, in one or more of the segments. You should use a new inode for your file, and create as many data blocks as necessary to store the contents of the file. Divide the linux file into blocks of size 1K and write them out onto the disk. You will need to keep a mapping of `<lfs_filename>` to the inode that you use for your file, and store that on the drive.

`remove <lfs_filename>`

This command causes `<lfs_filename>` to be removed from the file system. The inode associated with the file should be recycled and reused.

`cat <lfs_filename>`

Display the contents of `<lfs_filename>` from within your file and print them on the screen. (All files that we use will be text files.)

`display <lfs_filename> <howmany> <start>`
Read and display `<howmany>` bytes from file `<lfs_filename>` beginning at logical byte `<start>`. Display the bytes on the screen.

`overwrite <lfs_filename> <howmany> <start> <c>`

Write `<howmany>` copies of the character `<c>` into file `<lfs_filename>` beginning at byte `<start>`. If `<start> + <howmany>` exceeds the current file size, this command should increase the file size appropriately. Don't forget to update the file size in the inode.

`list`

List all the names and sizes of the files in the file system.

`exit`

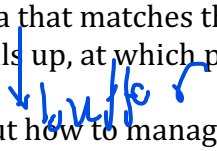
Writes out all the metadata and file system data that is cached in memory and exits.

File System Metadata

Your file system should be structured as a very simplified version of a unix file system. It should include inodes and data blocks, but no indirect blocks. Each inode should store the file's string name and the file's size, and 128 direct data block pointers (as opposed to the 12 that real inodes contain). You file system does not have to support indirect block pointers or double indirect block pointers. Therefore, the maximum file size is only 128K. No directories. No links. For simplicity, inodes are the same size as data blocks (1K) even though they don't really need to be that large. You should support an *imap* structure that should have a home on the disk in the log (that is, within the 32 segments), and you should store a single *checkpoint region* that stores the location(s) of the imap. The checkpoint region may reside outside of any of the 32 segment blocks, in a separate file named CHECKPOINT_REGION. That file should contain a list of block numbers that store parts of the imap, in order. Both inodes and block numbers should be stored in 4-byte unsigned ints. The file system will be limited to 10K files. Therefore the imap can be stored in 80K bytes, which is 80 disk blocks. So the checkpoint region should be represented as a file that contains exactly 320 bytes, no more no less. When your file system process starts up, it should read in this checkpoint region and from it should be able to piece together anything else it needs.

Segment Management and Cleaning

As a *log structured* file system, data blocks, inodes, and blocks that contain parts of the imap should move around on the disk when the change and get written out to the end of the log. You must implement a segment cleaning process that reads segments, determines which of their blocks contain live data (by using the segment summary blocks and inodes), and write out those blocks into some of the segments, freeing up others. The file system should keep a list of free segments and use them as needed. The system should contain a that matches the size of a single segment (1MB), and should operate on blocks in that buffer until it fills up, at which point the entire buffer should overwrite one of the free segments.



For details about how to manage segments and implement the various components of LFS, read the chapter in the book and see the lecture notes. Be aware, however, that you are implementing a subset of the full functionality!

Lab 7 vs. Program 2

For Lab 7, you should support `import`, `remove`, `list`, and `exit` (and restart to read in the files and list them after running your file system process again). You do not have to support `cat`, `display`, or `overwrite`, nor do you have to implement segment cleaning for Lab 7 (so you may assume that importing files does not fill the disk).
The fully functional proram is due as Program 1.