LFS - Phase 1
Authors: Xueheng Wan, Wenkang Zhou

Required completion of components:
mklfs, lfsck, prototypes of Log and File layers, simplified Directory layer

File: mklfs.c
Description: This file handles mklfs function that formats and creates a new LFS file system using arguments passed from the command line. The creation involves super block initialization and writing it into the Flash. In addition, with the scope of phase 1, it also writes ifile and creates root directory to the first segment (Note: the first segment is not super block, it is the first segment can be accessed by user).

File: log.h log.c
Description: This file implements the log layer of the LFS. Log layer only handles log cache and I/O. It decides placement of blocks. Any logical operations on file-level are handled in File layer or upper layers, thus it implies that data passed to log layer with a unit of block and log layer only read/write data from/to Flash and update inode of owner of input blocks. By doing this, we simplified the design of log layer.
Functions:
Log_Read: Reads blocks on the given address; An address consists of a segment number and a block number; Segment cache for recent N segments is not supported in this phase as it is not needed, we decided to leave its development for phase 2. This function only reads data in unit of block.
updateInode: Update Inode with a given inum, which results a change of ifile; The modified block of ifile is pushed onto the current log, meanwhile the inode of ifile is also updated with the address of the new position of the modified block.
Log_Write: Push blocks onto log segment cache whenever it is called from File Layer; Inode of files owning input blocks will be automatically updated as blocks are pushed onto log; Address of new block placement will be returned to caller; the log segment is written onto Flash once the log segment cache reaches full;
Log_Free: Need Further Development. This function is out of scope of phase 1, we decided to leave it for phase 2.

File: file.h file.c
Description: This file implements the file layer of the LFS. File layer handles all operations in file level.
Functions:

File_Create: An inode is created and initialized with the given inum during a call of this function; This also results an update of ifile. Every time it calls Log layer, it requires data of a size of block.

File_Write: This function implements write operation of a file. The operation is actually divided into multiple operations in log layer by File_Write_Helper if written data is cross-blocks. The File_Write calls its helper for writing a new block. File write operation does not remove data in blocks, it overwrites or appends data in blocks.

File_Read: This function implements read operation of a file. Similar to File_Write, this operation is divided into multiple operations in log layer by its helper. We only read a block each time until we finish the written data reaches input length.

File_Free: Need Development in Phase 2; Not in the scope of Phase 1.

Read_Inode_in_Ifile: Extract an inode from ifile.

Extent Inode: Assign a new block to a file (update its inode).

FIle: lfs.h lfs.c

Description: This file implements a simplified directory, root directory, and our implementation of FUSE functions. This layer is the place for the LFS directly communicating with FUSE.

Functions:

Init: Read configuration of the file system into memory so that the file system can be accessed and operations on it can be executed.

Init_sb: Load the superblock from Flash into memory in order to get properties of the file system.

Lfs_getattr, lfs_open, lfs_read, lfs_write, lfs_readdir, lfs_destroy, lfs_create, lfs_truncate: A list of functions implementing interfaces of FUSE, and handle communications between LFS and FUSE.

Other FUSE functions: To be developed in phase 2.

File: lfsck.c

Description: This file is used for debugging. It is a reporter that reports all states of the file system we need during the development. Due to this nature, we decided to build features only for what we needed in our development stage. Thus, it may not cover all features in LFS handout, such as incorrect segment summary information (segment summary not used in phase 1). However, this function will be further developed with all required features and used in debugging for phase 2 development.

Compile:

make mklfs
make lfs
make lfsck

Test Result:

In testing stage, we learned that different environmental configurations in our macbook and the Lectura made difficulties to test our project. We have built our own test cases covering almost all developed features for every layer, and improved the implementation to pass our own tests before we tested the given python file lftest.py.

When it ran in our local macbook, our implementation has **passed all test cases** of lftest.py when we performed test cases one or two at each time. And the tests were failed due to "Input/output error: 'a/foo'", in which 'a' is the mount point, when we performed all test cases together. We believe this involves environment setup issues.

Running in Lectura was completely a different story. Our implementation has passed all test cases excepting for the last one in Lectura when we performed all test cases together. This was kind of strange for us since our debugging functions showed that it should have passed the last test case as it passed in our local macbook. And later on, Lectura was down while we were investigating this issue, therefore we lost our chance to find it out what happened to the last test case.