Operating System
Project 3 LFS Design Document

Gaogao Pan[*], Si Jiang[†], Zeying Cai[‡]

Introduction

The whole LFS implementation with FUSE 3.1is explained elaborately in **LFS.h** head file (operation logic, macro definitions, data structures and functions) . Especially for macro definitions:

- The LFS has 100MB **LOG-SIZE**, support file and dir's name with max length **MAX-FILE-NAME** = 124, longer name will return a error indicates name is too long.
- A file or dir can have at most **MAX-FILE-SIZE-INBLK** = 10240 data blocks, larger file will return a file too large error. (This is constrained by our simple inode structure, which will be taken away by a multi-level inodes structure when we finish proj4).
- The LFS can have at most **MAX-INODES** = 2048 inodes. (This is constrained by predetermined size super block structure, more inodes lead to larger super block).

data structure

The data structure is defined and in **LFS.h** and **inode.h**

- **LFS superBlock**: SuperBlock that record next inode number, log tail and imapTable. SuperBlock has pre-determined size, store in the head of the log. Each time the LFS is mounted, load the SuperBlock into Mem and modify it in the Mem. Only push the newest SuperBlock into the log after each operation is commited into the disk, this gurantees the transcation.
- **ImapTable**: Used in super block. Record the disk addr according to inode number. Also record create, access, modify time, so we do not need to create new inode only when time part is changed.
- **DirtDataBLK**: The data structure used in dir's data block. Record the name and corresponding inodeNum of files under the dir.
- **Inode**: Record uid, gid, mode, size, number of data blocks, number of links, data blocks' address. Time part information is saved in superBlock's imap to save operations making new inodes

[*]Contributes all tests
[†]Implements the whole LFS
[‡]Contributes :-)

code organization

All **.c** files are in the **src** subdir and all **.h** files are in **include** subdir. The LFS can be compiled by **make all** and destroied by **make clean**. After make, one can run **./bin/lfs.o -f $MOUNTPATH$** to mount the LFS. The log is saved in **log/LFS_Log**. In the **bin** path, one can run **./logReadout.o** to readout the metadata in the log.

details about 7-10

- 7. Implemented by function do_chown and do_chmod. First exam the authority of current user (whether the owner, whether the root), if have the authority, then make a new inode with new permissions.
- 8 and 10. Threads safe and random crash-recovery are guranteed by our super block: since the super block is first loaded and then modified in the memory, and only writes into disk after all other data and metadata are written into the disk, we can have transcation.
- 9. For now, we do not implement segment and only cache the super block. Also, the super block is written into the head of the disk after each operation finished. This can be really inefficient and will be improved in proj4.

Testcases

The test examples under folder **tests** are written in shells with prefix "test". We have tested the following instructions: mkdir, rm, cd, touch, echo, cat, cp, mv, chmod, ln, tree. We have also tested crashings, big files and adding lines to files. To run these shells, first run **./bin/lfs.o -f tests/testfolder/MyLFS**, this mount the LFS in path MyLFS and some tests will compare the behaviour of LFS and original OS's FS by diff, tree MyLFS and OSFS

- test: The combination of test cases. It just run all other test shells listing bellow in order.
- test_mkdir.sh: It create a1,a2,a3,a4 four directories and check their properties, then remove them all. This shell test cd, mkdir and rm on directories.
- test_rm.sh: It create directories a1,a2 and files t1,t2,t3, them remove them. This shell tests rm, touch.
- test_readwrite.sh: It create, write and read file t1. Then it move t1 to directory a1. This shell tests echo (to file), cat, mv.
- test_chmod.sh: It create a txt file, an executable file and a directory (with another txt file inside it). The shell use chmod to control the authorities. For each type (txt, executable file, directory), the shell tests 3 cases (777,444,111) and read/execute/write them, to check if chmod works.
- test_ln.sh: It simply creates a file t and a hard link a/t1 to it (a is a directory). This shell tests ln. (Create hard link between different file systems are not allowed.)

Testcases continued on next page...

- test_threadsafe.sh: It creates two threads trying to read, write to the same file (threadsafe.txt). Each thread try 5 times. The output shows that context switch happens between read and write code, but the contents are not ruined.
- test_tree.sh: It just simply create many directories and tree them. (Our ln does not allow hard link between different file systems, so this test case is simply under this file system)
- Crashing: we have tried crashing during a thread using this file system (e.g., vim), which would left an extra swap file in log.
- Big files: copying a big file into the file system has been tested and proved correct.
- Adding lines to files: we use **echo 'xxx'** >> **a.txt** to add line to files, proved correct.
- chown: change the owner of a file, tested by first create a file by a normal user, then chown by root.

Github repo

Since proj1 and proj2 are both for nachos, we make a new repo. for LFS implementation with url: `https://github.com/Su-Li-Fuwa/LFS-fuse`