Design Document of Project 3

Yushi Bai, Ruixiao Yang, Zewen Fan

December 2020

1 README

Our test files are in the folder `tests`, where `test_echo.sh` is used to test the functionalities of our echo file system, `test_lfs.c` is used to test the basic file operations of our LFS implementation (to make it more convenient, one can run `test_lfs.sh` to autorun this test), `test_concurrent.c` is used to test on concurrent processes accessing the file system. To run all tests once and for all, we provide a script `test_all.sh` in folder `tests`.

Our code runs successfully on a linux OS (ubuntu) with fuse version 3.10.0. We only used common libraries in .c programs.

IMPORTANT: for our file system to run correctly, there are two places where absolute location must be used: one is the 5th line of function `yrx_init_lfs` in file `metadata.c`, indicating the location of the 100MB file, while the other one is the 1st line of `block_dump` function in file `metadata.c`, indicating the location to store the pretty print content.

2 File system structure

The whole file system maintain a few metadata including inode map (mapping inode from id to block index), next block to write, a buffer for temporary storing, and the size of it. Also we construct inode and directory as the paper required. To handle large file, we create indirect blocks, but the implementation of write file seems to still have some bugs. In our implementation, the directory only contains files'(directories') names, and their corresponding inode. The inode contains the uid, gid, permission, file size, file name, file locations. It also has indicators showing whether it is a directory, how much child it has if it is a directory and where the indirection block is. For these structures, we provide some operations. The basic operations are read/write from/to buffer and disk of these structures. The high level operations including create, delete, modify.

3 Code organization

`echo.c` implements the echo file system, which prints out the name of the file system operation in `op.log`. `metadata.c` implements the log structured file system and `main.c` is modified from `echo.c` and use the log structured file system to implement the functions required by FUSE. `buildfs.c` is the code to generate the 100 MB file system file, whose content does not matter because our file system cannot be recovered from files.

4 Advance functionalities

To configure owner and group, we use fuse operation textttfuse˙get˙context(). Each inode store the uid, gid, and its mode. We compare these two attribute when loading inode to verify the permission.

Our concurrency is implemented by a general lock called transaction, only one process can get the lock and operate on the file system.

5 Test details

5.1 Testing echo file system

In `test_echo.sh`, we call on `ls, cat, chmod, mv, cp, ln, rm` regular linux commands on our mounted echo file system, which prints out into a file `op.log`. Then we run `readlog.py` to get the outputs from the file and check their validity. A right echo file system should pass all the commands with "SUCCESS" output on screen.

5.2 Testing basic LFS functionalities

In `test_lfs.c`, we tests on basic file operations including `cp, cat, mv, mkdir, ln, touch, ls, rm, chmod, ls -l` and one that read and write to a file (the size is so large that indirection should be used) in our LFS file system. We compare between the output from our file system and the output from linux file system to check our LFS implementation.

5.3 Testing advance LFS functionalities

In `test_concurrent.c`, we test under the circumstance where concurrent processes access the file system. We check when two concurrent processes read and write from the file system and see if they are consistent, and we also check when there are multiple concurrent processes and see if every operation gets executed correctly eventually.

We also include a `block_dump` class in `metadata.c` which calls the `pretty_print` functions to print out all metadata blocks (including the superblock for the file system and inode for each file/directory).

Our LFS also support tree command, printing the true tree structure of the mounted file system.

6 Known bugs and limitations

The implementation of reading or writing large files is not correct. Also, each time the file system will be empty when mounted again, meaning that our file system cannot recover from a crash.