PROJECT 5B:
FILE SYSTEM - WRITING

Minimum Requirements: None

TEST DISTRIBUTION

Public tests – 34 tests | 167 points

Release tests – 0 tests | 0 points

Secret tests – 0 tests | 0 points

Yes, no secrets, you have all the test code to debug.

NEW TEST FILE

This project relies on project 5a but no other projects. There is some updates in the lfstst.c file, please download it and place it under src/user, replace the old one.

The same disk image is used from previous project.

FUNCTIONS TO IMPLEMENT

- static int LFS_Write (struct File *file, void *buf, ulong_t numBytes)
- static int LFS_Open (struct Mount_Point *mountPoint, const char *path, int mode, struct File **pFile)
- static int LFS_Create_Directory (struct Mount_Point *mountPoint, const char *path) static
- int LFS_Delete (struct Mount_Point *mountPoint, const char *path, bool recursive) static int
- LFS_Sync (struct Mount_Point *mountPoint)

SYSTEM CALLS TO MODIFY

- Sys_Delete
- Sys_CreateDir
- Sys_Sync

FILE SYSTEM & SEGMENTS

Get_FS_Buffer
/* modify buffer->data */
Modify_FS_Buffer
Release_FS_Buffer

However, keep in mind for LFS, you need to buffer in term of segment. That means you need to have one segment buffered in memory all the time and flush this segment to disk once it is full or sync is called.

LFS_OPEN

- If a file does not exist and (mode & O_CREATE) == True, create it
- If a containing directory does not exist throw an error (ENOTFOUND)
- Split the path to directory name and file name
- Retrieve the directory inode
- Find a free writable inode (address in imap is 0) for the file. Set up the parameters for our new file and write the inode on the disk.
- Update imap entry.
- If inodes are used up, return ENOMEM
- Create a dirent representing the file and write it in the dirent blocks in the directory inode's data block.
    - You can use a helper function from both this and LFS_Write here
    - Remember to update parent inode, write the new inode to disk and update the imap entry
    - Since LFS is always appending to the disk, the old inode will be there and wait until a garbage collector cleans it. (not required for this project)

LFS_CREATE_DIRECTORY

- If the directory already exists return EEXIST
- If the parent directory doesn't exist return ENOTFOUND
  - Directories should not be created recursively.
- Split the path and get the parent directory's inode
- This should be the same routine for LFS_OPEN, make a helper function
  - Find a free writable inode for the new directory
  - Create a dirent for the new directory and add it to the parent directory inode's extents
- Make dirents in the new directory for '.' and '..'
  - This is the only difference when creating a file and creating a directory.

LFS_WRITE

- [RECOMMENDED]
  - Create a helper function to write to data blocks
    - Let's you bundle the same work between writing files and directories
  - Create a helper function that updates inode for you.
  - Set a threshold for the max data a data block can hold.
- Use File→filePos to determine the starting location
- Once you get to the location, you can either read the data block content, append the new data, and write the entire data block back, then update the inode.
- Or you can use a new data block. Indirect blocks are not required but it can be used, if you want to use a new data block, think about what happens if all the data blocks are used up.

LFS_DELETE

- Delete the dirent entry from parent directory
  - Take that dirent out and write to a new one without this entry
  - Make sure how your code for finding dirents from a path works with how you delete
- Decrement inode reference count
- If inode reference count drops to zero (always true in the scope of this project)
  - Remove the reference to the inode (clear imap entry)

LFS_SYNC

- Return Sync_FS_Buffer_Cache() (already provided to you, check bufcache.c)
- Also write the segment to disk, and start a new segment in buffer.
- Pass the cache you created during mount as the parameter

NOTE

- Remember to do sanity checks and free memory when necessary
- Don't forget to use Release_FS_Buffer after using a buffer, especially now that we are writing
- If a file has size expect there are some data blocks.
- For anything you think is reasonable but not mentioned in this slides, it should be correct. This does not cover all the cases in all the functions.