Project 5: Filesystems

~~Deadline: May 12 11:59 PM~~

Deadline: May 14 11:59 PM

Weightage fsck:lfsreader 70:30

Congratulations! You have made it this far into CS 537!

The good news is that the project will be done entirely in Linux rather than xv6.

The bad news is there are two parts.

The good news is the project should be fun!

The bad news is the project can be a little maddening because you're searching for errors in part 1 (but just searching for something in part 2).

The good news is it's a puzzle!

The bad news is it's a puzzle!

Part 1: Saving the xv6

You will be given a number of broken xv6 file system images. You will write fsck to fix them where possible or report that the file system is, alas, no more.

Remember, that an xv6 file system image to Linux, is just a file. You can open it, read it and write to it. It's just a file. The relevant file system related data structures are found in xv6.

Few broken xv6 images are available at ~cs537-1/public/p5/fsck

Part 2: LFS Reader

EDIT : The output format has been updated. Please check the new format required by 'ls'

You will be given file system images based on LFS. You will have to write a simple "ls" command and a simple "cat" command. You'll have to use the file system image (it's just a file to Linux) and tease out the contents of the specified directory (for ls) or dump the contents of the specified file (for cat).

File System Information
Block Size for LFS is 4096
Inode number for the root directory is '0'
Invalid directory entry has an inode number -1
Note: These information holds good only for LFS and not for the fsck part

- The file system related structures can be found in this header file.
- Your binary should be named "**lfsreader**"
- Invocation format of commands
  - lfsreader cat 'absolute pathname for a file' 'lfs image'
  - lfsreader ls 'absolute pathname for a directory' 'lfs image'
- Output Format
  - Incuring any form error, you should print "Error!\n"
  - **ls**: List contents of the directory in the order they appear in individual lines. Using the command on a file should display an error. If the line item is a directory, suffix the entry with a trailing '/'. Note that this trailing slash is only for directories and not files.
  - **cat**: Output the contents of a file. Using the commanf on a directory should print an error.
  - Error if directory or file not found.

A sample LFS image can be found at ~cs537-1/public/p5/LFS/image/example.img

Notes for part 1

This is not an exhaustive list of what can go wrong. Remember, that sometimes fsck must be run more than once to fully clear a file system.

Superblock consistency checks: As you know, a file system's superblock contains basic global information about the entire file system. If these values are corrupted, there may be no way to salvage the file system. The values in the superblock may not be able to be vetted individually. Taken together though, there are some consistency checks that you can perform.

For example, the size of the inode list may be specified as being larger than the file system itself. This is a problem.

You will make sure the superblock passes "the sniff test". Should it fail, if the error can be overcome you will do so. If not, you will inform the user that their terrible bad awful day is about to get worse.

Basic of free map checks: Enumerate the free map, checking as you go that the values you are finding all seem OK. Once enumerated you can determine if the superblock entries referring to the free map also make sense.

Basic inode checks: Enumerate all the inodes in the file system performing various sanity checks as you go. Keep track of which inodes are free and which are marked as allocated. Sanity check fields such as file size for the allocated inodes. For example, is there a file size marked as being so large as to exceed the space used by the superblock, inodes and supposedly free blocks? Are the file types valid? That is, a file type can include regular, directory, device, etc. Do the reference counts (link counts) make sense? When you encounter a bad inode, the only thing you can do is clear it.

Data block checks using both the enumerated inodes and free map: Now that you have all the free blocks enumerated and a list of all the inodes in memory, compare the two. No block pointed to an inode should appear in the free map, for example. No block anywhere should be pointed to twice. When you get this far you should be able to repair any damage in the free map. For blocks referred to more than once, a real fsck might ask which inode to toss away. For this project, toss away all inodes involved in the overlap so as to avoid user interaction.

Directory Checks: Sniff all data contained in all directories. Do they seem correct? For example, do they contain dot (.) and dot dot (..)? You know for sure that dot and dot dot should refer to directories. Do they? You've enumerated all the inodes so you can tell. For that matter, you can determine if every allocated inode is referred to by a reasonable number of directory entries. For example, you know that every inode marked as a directory should have at least a minimum number of references in other directories. Do the reference counts to all files (no matter the type) correspond to the reference counts found in their inodes? Is every valid inode referred to by a directory?

If you find a valid inode with no directory references you have no idea what to call the file. We will have created a directory entry in the root directory of the file system called "lost+found".

If a directory entry refers to an invalid inode, just clear the directory entry. If you determine that a directory is not referred to, put it in the "lost+found".

Are there any directory entries that are clearly bad such as an inode number larger than the number of inodes in the file system? Do all the file names in directory entries make sense? For example, suppose a file has only a three character name. Can you tell?

Other data block and indirect block checks: By now you have determined that every indirect block seems reasonable, haven't you? And, you already determined that the number of blocks referred to by each inode is consistent with its size, right?

Blocks marked as being data blocks (for non-directories): Obi Wan Kenobi says "These aren't the droids you are looking for."

Helpful Resources

Here is a helpful link, it refers to a FFS based file system so some of the information isn't applicable, but a large amount is.
The OSTEP chapter on what can go wrong with a file system is also helpful.

Submission Instructions

Please copy your source code and makefile for the LFS reader under p5/linux.
For FSCK, please submit your sourcecode and makefile under p5/xv6.