From CS 537 - Introduction to Operating Systems - Fall 2017

Site: Project5a

Project 5: File System Checker

Update:

- An example image that needs to be repaired (for extra credit part) has been provided. Note this image might not be used in our final tests.

Background

In this assignment, you will be developing a working **file system checker**. A checker reads in a file system image and makes sure that it is consistent. When it isn't, the checker takes steps to repair the problems it sees; however, you won't be doing any repairs to keep your life at the end of the semester a little simpler.

A Basic Checker

For this project, you will use the xv6 file system image as the basic image that you will be reading and checking. The file **include/fs.h** includes the basic structures you need to understand, including the superblock, on disk inode format (`struct dinode`), and directory entry format (`struct dirent`). The tool **tools/mkfs.c** will also be useful to look at, in order to see how an empty file-system image is created.

Much of this project will be puzzling out the exact on-disk format xv6 uses for its simple file system, and then writing checks to see if various parts of that structure are consistent. Thus, reading through mkfs and the file system code itself will help you understand how xv6 uses the bits in the image to record persistent information.

Your checker should read through the file system image and determine the consistency of a number of things, including the following. When one of these does not hold, print the error message (also shown below) and exit immediately.

1. Each inode is either unallocated or one of the valid types (`T_FILE`, `T_DIR`, `T_DEV`). `ERROR: bad inode.`
2. For in-use inodes, each address that is used by inode is valid (points to a valid datablock address within the image). If the direct block is used and is invalid, print `ERROR: bad direct address in inode.`; if the indirect block is in use and is invalid, print `ERROR: bad indirect address in inode.`
3. Root directory exists, its inode number is 1, and the parent of the root directory is itself. `ERROR: root directory does not exist.`
4. Each directory contains . and .. entries, and the . entry points to the directory itself. `ERROR: directory not properly formatted.`
5. For in-use inodes, each address in use is also marked in use in the bitmap. `ERROR: address used by inode but marked free in bitmap.`
6. For blocks marked in-use in bitmap, actually is in-use in an inode or indirect block somewhere. `ERROR: bitmap marks block in use but it is not in use.`
7. For in-use inodes, direct address in use is only used once. `ERROR: direct address used more than once.`
8. For in-use inodes, indirect address in use is only used once. `ERROR: indirect address used more than once.`

9. For all inodes marked in use, must be referred to in at least one directory. `ERROR: inode marked use but not found in a directory.`
10. For each inode number that is referred to in a valid directory, it is actually marked in use. `ERROR: inode referred to in directory but marked free.`
11. Reference counts (number of links) for regular files match the number of times file is referred to in directories (i.e., hard links work correctly). `ERROR: bad reference count for file.`
12. No extra links allowed for directories (each directory only appears in one other directory). `ERROR: directory appears more than once in file system.`

Other Specifications

Your checker program must be invoked exactly as follows:

```
prompt> xv6_fsck file_system_image
```

The image file is a file that contains the file system image. If no image file is provided, you should print the error `Usage: xv6_fsck <file_system_image>.` to stderr and exit with the error code of 1. If the file system image does not exist, you should print the error `image not found.` to stderr and exit with the error code of 1. If the checker detects one of the errors listed above, it should print the proper error to stderr and exit with error code 1. Otherwise, the checker should exit with return code of 0.

Extra Credits

For this project, you can gain some extra points by implementing the following more challenging condition checks:

1. Each .. entry in directory refers to the proper parent inode, and parent inode points back to it. `ERROR: parent directory mismatch.`
2. Every directory traces back to the root directory. (i.e. no loops in the directory tree.) `ERROR: inaccessible directory exists.`

Another way to earn more extra points is to actually repair the "`inode marked use but not found in a directory`" error. We will provide you with a xv6 image that has a number of in-use inodes that are not linked by any directory. An example of such image is available at `~cs537-1/ta/tests/5a/images/Repair`. Your job is to collect these nodes and put them into the `lost_found` directory (which is already in the provided image under the root directory).

By doing so, your will need to obtain the write access to the file system image in order to modify it. This repair operation of your checker program should only be performed when `-r` flag is specified:

```
prompt> xv6_fsck -r image_to_be_repaired
```

In this repair mode, your program should **not** exit when an error is encountered. You can also assume there is no other types of error in the provided image. It should exit only after it has created an entry under the `lost_found` directory for every lost inodes.

Hints

It may be worth looking into using `mmap()` for the project.

It should be very helpful to read chapter 6 of the xv6 book **here**. Note that the version of xv6 we're using does **not** include the logging feature described in the book; you can safely ignore the parts that pertain to that.

Make sure to look at **fs.img** , which is a file system image created when you make xv6 by the tool mkfs (found in the tools/ directory of xv6). The output of this tool is the file fs.img and it is a consistent file-system image. The tests, of course, will put inconsistencies into this image, but your tool should work over a consistent image as well. Study mkfs and its output to begin to make progress on this project.

Tests

You can run the tests at `~cs537-1/ta/tests/5a/runtests`. You can print a list of all available tests and their descriptions using `~cs537-1/ta/tests/5a/runtests -l`.

The error messages expected are at `~cs537-1/ta/tests/5a/err`. The images are at `~cs537-1/ta/tests/5a/images`.

The test cases for the extra credit part will not be provided. You need to create your own images or scripts to test your implementation.

Retrieved from http://pages.cs.wisc.edu/~swift/classes/cs537-fa17/wiki/pmwiki.php/Site/Project5a
Page last modified on December 03, 2017, at 05:47 PM