CS 537 Spring 2019, Project 5: File System Checking

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 this project a little simpler (well, unless you want a bit challenges and to earn some extra credits).

Background

Some basic background about file system consistency is found in the textbook chapter on Crash Consistency: FSCK and Journaling (http://pages.cs.wisc.edu/~remzi/OSTEP/file-journaling.pdf)

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.c` 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 a problem is detected, print the error message (shown below) to **standard error** and exit immediately with **exit code 1** (i.e., call `exit(1)`).

1. Each inode is either unallocated or one of the valid types (`T_FILE` , `T_DIR` , `T_DEV`). If not, print `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. If not, print `ERROR: root directory does not exist.`
4. Each directory contains `.` and `..` entries, and the `.` entry points to the directory itself. If not, print `ERROR: directory not properly formatted.`
5. For in-use inodes, each address in use is also marked in use in the bitmap. If not, print `ERROR: address used by inode but marked free in bitmap.`

6. For blocks marked in-use in bitmap, the block should actually be in-use in an inode or indirect block somewhere. If not, print `ERROR: bitmap marks block in use but it is not in use.`
7. For in-use inodes, each direct address in use is only used once. If not, print
   `ERROR: direct address used more than once.`
8. For in-use inodes, each indirect address in use is only used once. If not, print
   `ERROR: indirect address used more than once.`
9. For all inodes marked in use, each must be referred to in at least one directory. If not, print
   `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. If not, print
    `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). If not, print `ERROR: bad reference count for file.`
12. No extra links allowed for directories (each directory only appears in one other directory). If not, print
    `ERROR: directory appears more than once in file system.`

Other Specifications

Your checker program, called `xcheck`, must be invoked exactly as follows:

```
prompt> xcheck 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 usage error shown below:

```
prompt> xcheck
Usage: xcheck <file_system_image>
```

This output must be printed to standard error 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 standard error and exit with the error code of 1.

If the checker detects any one of the 12 errors above, it should print the specific error to standard error and exit with error code 1.

If the checker detects none of the problems listed above, it should exit with return code of 0 and not print anything.

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 used but not found in a directory"` error. An xv6 image that has a number of in-use inodes that are not linked by any directory. An example of such image is available: `/u/c/s/cs537-1/tests/p5/images/Repair`. Your job is to collect these nodes and put them in `lost_found` directory. The `lost_found` directory is already present in the root directory of the provided image.

For doing this you will need to obtain the write access to file system image in order to modify it. The repair operation of the program should only be performed when `-r` flag is specified.

```
prompt> xcheck -r image_to_repair
```

Hints

It may be worth looking into using `mmap()` for the project. Like, seriously, use `mmap()` to access the file-system image, it will make your life so much better.

It should be very helpful to read Chapter 6 of the xv6 book here (https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf). 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

It is a good habit to get basic functionalities working before moving to advanced features. We have provided a few tests for you, which can be run through the command:

```
/u/c/s/cs537-1/tests/p5/runtests
```

You can print the descriptions of these tests using the `-l` flag. The raw xv6 images are available in `~cs537-1/tests/p5/images`.

The test cases for the extra credit part will not be provided. One of challenging parts to earn extra credits is that you will need to create your own images and scripts to test your implementation.

Due Date

29th April, 11:59 PM

Submitting your implementation

The handin directory is `~cs537-1/handin/<cs-login>/p5/` . To submit the solutions copy all necessary source and headers files to the handin directory. One way to do this would be to navigate to your solution and execute

```
cp -r . ~cs537-1/handin/<cs-login>/p5/
```

Consider the following when submitting the solution:

- If you use any slip days you have to create a `SLIP_DAYS` file in the `/<cs-login>/p5/` directory otherwise we use the submission on the due date.
- Your `xcheck.c` and other required header files which should be directly copied under the `~cs537-1/handin/<cs-login>/p5/` directory.
- For `types.h` and `fs.h` you should use the one provided in xv6.
- You are free to create any extra header files. As long as the code compiles with `gcc xcheck.c` . Meaning no other `c` files should be needed to create the binary.
- Submitting types.h and fs.h is also okay. But these will be supplied when the tests are run.

If you choose to work in pairs, only one of you needs to submit the source code. But, both of you need to submit an additional partner.login file, which contains one line that has the CS login of your partner (and nothing else). If there is no such file, we are assuming you are working alone. If both of you turn in your codes, we will just randomly choose one to grade.

Collaboration

This project is to be done in groups of size one or two (not three or more). Now, within a group, you can share as much as you like. However, copying code across groups is considered violation of plagiarism policy.

If you are planning to use `git` or other version control systems (*which are highly recommended for this project*), just be careful **not** to put your code in a public repository.

Acknowledgement

The project uses material created by Prof. Remzi for his OS class offered in Spring 2018 and Prof. Michael Swift for his OS class in Fall 2017.