CS/SE 4348 F18 Operating Systems

Project 5: File System Checking

Due date 30th November, 2018

In this project, you will be developing a working file system checker. A checker reads a file system image and checks it's consistency. 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.

You can do this project individually or with a partner. You cannot share your work with anyone other than your project partner.

This project can be done in **csjaws.**

A Basic Checker

For this project, you will use an 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 the 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 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.

Hints

It may be worth looking into using `mmap()` for the project. Using `mmap()` to access the file-system image will make your (kernel programming) life so much better.

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 progress well on this project.

xv6 Source code

The xv6 source code for this project is /cs4348-xv6/src/xv6.tar.gz. You do not need any of the code that you implemented in Project 2. Copy this file to your local working directory for this project and extract the source code tree, and run 'make' to create the fs.img.

Testing

Make sure you compile your checker as follows:

```
gcc xcheck.c -o xcheck -Wall -Werror -O
```

Sample file images with inconsistencies will be made available in the directory `/cs4348-xv6/src/testscripts/p5/`

Submission

Copy your entire source code (xcheck.c and any other include files) to the directory /cs4348-xv6/xxxyyyyyy/p5/.

If you have worked with a partner, only one of you need to submit the files. But, both of you should create a text file named PARTNER in /cs4348-xv6/xxxyyyyy/p5 and save your partner's name and netid in the file.