Project 4: Filesystems

November 22, 2013

1 Overview

In this project, you will implement a read-only driver for ext2, a standard filesystem supported by the Linux kernel. Starting from a reference implementation provided in binary form, you will reimplement the driver one piece at a time until you do not link against the original code at all. The reference blob and your code will both be used by ext2cat, a simple program included with the project that retrieves the content of a file from an ext2 image and prints it to stdout.

2 Provided code

The project tarball contains a number of files. We recommend examining them in roughly the following order:

- `eecs343.img` - An ext2 filesystem image. This is a regular file which has been formatted as if it were a real disk partition. It contains some nested directory structure with a handful of files scattered throughout.
- `reference.o` - A binary blob implementing an ext2 driver. You want to make this file unnecessary.
- `ext2cat.c` - The final user of your code. Immediately after extracting the project, you should be able to build `ext2cat` with Make and use it with the provided filesystem image by running

  ```
  ./ext2cat eecs343.img /README.txt
  ```

  You will see a turkey.
- `lib/ext2_access.c` - The file where you should put all of your work. You will notice that almost all of the functions inside are simple wrappers around reference versions (e.g., `get_inode()` does nothing more than hand off to `_ref_get_inode()`). Your task is to reimplement these functions so that they do not invoke any of the `_ref` code.
- `include/ext2fs.h` - A header taken from the Linux implementation of ext2. Contains a plethora of useful struct definitions and macros. It's long, but you would be well-advised to give the entire thing a once-over before you start. You shouldn't waste time writing anything this file already contains.

After checking out all of those, you need one more thing before you can start: a specification of the ext2 data structures and disk layout. This information is available from a number of sources, but we recommend using this site:
`http://www.nongnu.org/ext2-doc/ext2.html`

3 Your filesystem

As you will not be able to mount `eecs343.img`, you need to know what you're looking for. These are the contents of the filesystem:

```
<root>
|-- code
|   |-- ext2_headers
|   |   `-- ext2_types.h
|   |-- haskell
|   |   `-- qsort.hs
|   `-- python
|       `-- ouroboros.py
|-- lost+found
|-- photos
|   |-- corn.jpg
|   |-- cows.jpg
|   `-- loons.jpg
`-- README.txt
```

README.txt will be the easiest to access, followed by the pictures, followed by the code. Be aware that `ext2cat` will happily print a jpg (or random data, if you've got a bug) to stdout, which may cause your terminal to behave strangely. As such, you may need to redirect its output to a file. If you do mess up your terminal, don't panic! Just hit enter, type `reset`, and hit enter again, even if the letters don't show up correctly.

4 Plan of attack

You may implement the necessary code in any way you like, but we recommend a bottom-up approach: start with the simpler functions and work your way up to the more complicated ones. The order in `ext2_access.c` serves as a good roadmap for this. Refer frequently to the spec and be confident you've got a handle on the superblock before moving on to block group descriptors.

5 Debugging

- As always, you should make frequent use of `gdb` to make sure your code is actually doing what you think it does.
- There is a utility called `dumpe2fs` that will give you a great deal of useful info about the layout of your filesystem. Much of this is read directly from fields in the superblock and block group descriptors, so it's a good way to sanity-check your code in the early stages of the project. (Note that this utility is located at `/sbin/dumpe2fs`, which may not be in your `PATH`. Invoke it with its absolute path or just configure your shell to search `/sbin`.)
- You need to eliminate the reference implementation to get full credit, but that doesn't mean you can't use it as an oracle! Running your code and ours side-by-side may help you determine if and how yours is screwing up.

6 Spoilers and simplifying assumptions

- The superblock is always at the same offset from the start of the disk. The first block group descriptor is not: it begins with the first whole block after the one that the superblock lives in.

- The inode table begins from 1. This is because an inode number of 0 is always invalid.
- Real ext2 filesystems have many block groups. You may assume that the disks you deal with only have one.
- Real ext2 inodes have blocks in several levels of indirection. You do not need to implement indirect blocks. (See *Grading* for an extra credit opportunity.)
- The spec describes linked list and "indexed" (hash table) directory entry formats. You only need to implement the linked list kind.

7 Grading

If your submission compiles into a functional ext2cat and contains no references to the `_ref` functions, you will receive full credit. If you do not eliminate all of them, you will receive partial credit for each function you did remove. As always, you will lose two points per compiler warning up to a maximum of 16 lost points.

The test script decides whether your ext2cat is functional by using it to retrieve a number of files in `eecs343.img`. Since the skeleton is fully functional when you get it, you will be assessed a penalty of 100 points if your handin cannot correctly extract all of the following files. You will be able to verify most of them visually, but the most reliable method (and the one we will be using) is a checksum. You can get an MD5 checksum with:

```
./ext2cat eecs343.img <file> | md5sum
```

These are the checksums of the files in eecs343.img:

file	md5sum
`/code/ext2_headers/ext2_types.h`	`730cc429e8d0ab183380ab14f51f2912`
`/code/haskell/qsort.hs`	`a7b79363f8645e4722a5d3d114311709`
`/code/python/ouroboros.py`	`ecd524e78b381b63df3d4bfcf662ce0d`
`/photos/cows.jpg`	`3f19778ecb384018ea53869313023d8b`
`/photos/corn.jpg`	`dc049b1372bf3a6a1aea17be9eca5532`
`/photos/loons.jpg`	`96a1f79091ef9eacc621d2495246043a`
`/README.txt`	`c092359845cf0c2859ea9a532b7764ca`

Note that you can also directly compare the output of `md5sum` of your code versus the reference implementation.

Eagle-eyed filesystem hackers will also notice that /photos/loons.jpg is too large to fit entirely in direct blocks, and that ext2cat will produce a truncated file of extremely low resolution that will remind some of you of the dial-up days. For 10% extra credit, modify ext2cat to successfully retrieve the data stored in indirect blocks. You will receive the points if running

```
./ext2cat eecs343.img /photos/loons.jpg | md5sum
```

produces the correct checksum for the entire file, which is

`eb5826a89dc453409ca76560979699bb`.