CS 111: Operating System Principles
Lab 4

Hey! I'm Filing Here 1.0.6

Jon Eyolfson
May 24, 2021
Due: June 4, 2021 at 8 PM PST

In this lab you'll be making a 1 MiB `ext2` file system with 2 directories, 1 regular file, and 1 symbolic link. You'll be given the `ext2` structures and some initial skeleton code which creates a file called `cs111-base.img` in the current working directory. You're expected to create a valid `ext2` filesystem. You'll mount it with `sudo mount -o loop test.img mnt`. After, when you run `ls -ain mnt/` you should get the following (I omitted the fields that depend on your machine or date):

```
total 7
       2 drwxr-xr-x 3    0    0 1024 .
                                     ..
      13 lrw-r--r-- 1 1000 1000   11 hello -> hello-world
      12 -rw-r--r-- 1 1000 1000   12 hello-world
      11 drwxr-xr-x 2    0    0 1024 lost+found
```

This lab **will be time consuming**, so please **start early**. Afterwards you will have experience with real file system internals.

Additional APIs. All of the header files you'll need are included for you. You should read and understand the macros.

Starting the lab. Run the following command to get the skeleton for Lab 4: `git pull upstream main`. You should be able to run `make` in the `lab-04` directory to create a `ext2-create` executable, and then `make clean` to remove all binary files. When you run the executable it creates `cs111-base.img` as described at the beginning. You can then run `fsck.ext2 cs111-base.img`, and will likely be asked to fix (many) errors. At the end of this lab you're expected to have no errors after running `fsck.ext2`. In addition to the course materials, you'll find extra resources here and here.

Files to modify. You'll be writing the following functions in `lab-04/ext2-create.c`:

```
write_superblock
write_block_group_descriptor_table
write_block_bitmap
write_inode_bitmap
write_inode_table
write_root_dir_block
write_hello_world_file_block
```

Finally, like always, you should modify `README.md` in `lab-04`.

Your task. Using wrapped system calls, along with the provided `ext2` structures you'll be creating a valid `ext2` filesystem image that the kernel could mount. You'll be connecting the dots and learning that filesystems, like everything on your computer, are just a bunch of numbers with structure. You should try not to hard code as much as possible. I have set up defines for you to use for block numbers and inode numbers. The root directory is always inode 2 for 2 is defined by `ext2`. We'll be creating a 1 MiB `ext2` file system with 1 KiB sized blocks and space for 128 inodes. You'll be creating 4 inodes: the root directory, the `lost+found` directory, a regular file named `hello-world`, and a symbolic link named `hello` which points to `hello-world`. You are provided with the inode for `lost+found` along with its directory block. The root directory and the `lost+found` should be owned by uid 0 and gid 0 (root). The owner should have read, write, and execute permissions. The group and other should have read and execute permissions. The `hello-world` file and `hello` symlink should be owned by uid 1000 and gid 1000 (typically the number of the first "normal" user). The owner should have read and write permissions. The group and other should only have read permission. Your `hello-world` file should only be 12 bytes long and contain "Hello world" followed by a newline.

Errors. For any wrapped system calls, you should check for errors. If there is an error, you may exit with the error number (`errno`). There is now an `errno_exit` macro to make your code more readable.

Tips. This lab will be frustrating to start, as you won't have much to show for it. However, after you're done with the superblock and your first inode, you'll make progress much faster. Note that the skeleton code creates a 1 MiB image file for you, which is initialized to all zeros. Also, when you assign `{0}` to a `struct` in C, it will zero initialize it. You should zero initialize everything. Remember that blocks start from 0, and inodes start from 1.

Running. These are all the commands you'll likely want to use (minus the normal clean command):

```
make # compile the executable
./ext2-create # run the executable to create cs111-base.img
dumpe2fs cs111-base.img # dumps the filesystem information to help debug
fsck.ext2 cs111-base.img # this will check that your filesystem is correct
mkdir mnt # create a directory to mnt your filesystem to
sudo mount -o loop cs111-base.img mnt  # mount your filesystem, loop lets you use a file
sudo umount mnt # unmount the filesystem when you're done
rmdir mnt # delete the directory used for mounting when you're done
```

You can find example output of both `dumpe2fs` and `fsck.ext2` on the last page. If you think it may be easier to read the binary of your filesystem, or you're interested, you can use `hexdump -C cs111-base.img`.

Submission. Simply push your code using `git push origin main` (or simply `git push`). For late days will we look at the timestamp on our server. We will never use your commit times as proof of submission, only when you push your code to the course Git server.

Example output of `dumpe2fs cs111-base.img`. Note that your output may by slightly different. You should understand these values and use macros that make sense, and not just hard code them. However, you should get something like:

```
dumpe2fs 1.46.2 (28-Feb-2021)
Filesystem volume name:   cs111-base
Last mounted on:          <not available>
Filesystem UUID:          5a1eab1e-1337-1337-1337-c0ffeec0ffee
Filesystem magic number:  0xEF53
Filesystem revision #:    0 (original)
Filesystem features:      (none)
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              128
Block count:              1024
Reserved block count:     0
Free blocks:              1000
Free inodes:              115
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         128
Inode blocks per group:   16
Last mount time:          n/a
Last write time:          Sun May 23 19:35:00 2021
Mount count:              0
Maximum mount count:      -1
Last checked:             Sun May 23 19:35:00 2021
Check interval:           1 (0:00:01)
Next check after:         Sun May 23 19:35:01 2021
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)

Group 0: (Blocks 1-1023)
  Primary superblock at 1, Group descriptors at 2-2
  Block bitmap at 3 (+2)
  Inode bitmap at 4 (+3)
  Inode table at 5-20 (+4)
  1000 free blocks, 115 free inodes, 2 directories
  Free blocks: 24-1023
  Free inodes: 14-128
```

Example output of `fsck.ext2 cs111-base.img`. You need to make sure you get the following output:

```
e2fsck 1.46.2 (28-Feb-2021)
cs111-base has gone 0 days without being checked, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
cs111-base: 13/128 files (0.0% non-contiguous), 24/1024 blocks
```