Lab5 – Implementing A Simple Filesystem

Due: 11:59 pm Tuesday, December 05

Requirements: In this lab, you will learn about the Linux VFS by implementing a simple filesytem, which has the following features:

- an on-disk layout similar to a single block group of the ext2 filesystem, including a superblock, directory, inodes, and data blocks;
- a single directory containing zero or more files;
- hard links (multiple directory entries pointing to a single inode);
- storage for noncontiguous (fragmented) files, each at least 8 KB in size;
- at least 1000 files and 8 MB total size;
- direct references to data blocks in the inode;
- preservation of ownership, permissions and dates.

You do *not* need to implement the following features:

- storage for subdirectories, symbolic links, pipes, devices or other special files;
- dynamic allocation of inodes;
- indirect blocks;
- variable block sizes;
- multiprocessor safety and locking;
- detection of filesystem corruption;
- correct *statfs* output (from *df*, etc.), which means that it is OK for your filesystem to crash or behave unexpectedly if and only if the filesytem is corrupt; however, no user-level operations should cause a crash.

Implementing the filesystem:
Each value in [square brackets] below indicates the maximum number of points your group can receive on the corresponding portion of the assignment. The maximum score for this lab assignment is 100 points.

<u>The module</u>
Filesystems can be developed entirely outside the kernel, and loaded as modules. In this lab, you'll be writing the *lab5fs* filesystem as a module for the Linux 2.6.9 kernel.

Do not try to write your filesystem all at once, or even develop major components all at once. You'll find that the kernel will immediately oops, and you'll have no idea where the problem

lies. Instead, add a small piece of functionality, test, and isolate which one of your recent changes is causing the problem before continuing.

Here is a list of milestones you might find helpful:

- [5] The filesystem mounts.
- [5] The filesystem unmounts cleanly.
- [5] You can cd into the filesystem's mount point; it contains "." and ".." entries.
- [5] The filesystem contains one or more files.
- [5] You can add and remove files.
- [5] You can add and remove hard links to files.
- [5] You can set permissions and ownership on files.
- [10] You can read and write data from files.
- [10] Inodes and data blocks are properly freed when no longer in use.

And some conditions to check:

- [10] The filesystem preserves changes between mounts.
- [10] The filesystem properly handles errors.
- [5] The filesystem does not corrupt data.

Creating the filesystem:

[10] The mkfs (make filesystem) utility is the typical Unix way to create an initial filesystem structure on a device. Actually, there's one mkfs for each filesystem. Write mklab5fs, a user-level C program which creates a lab5fs filesystem on the device or file it's given.

Testing your filesystem:

Normally, filesystems live on disk partitions, exposed to the VFS layer as block devices. Since repartitioning your virtual machine's hard disk would be annoying, we will use the loop device. The loop device provides the means by which a file can emulate a block device.

First, create an empty disk image file:

```
$> dd if=/dev/zero of=image bs=1024 count=1024
```

This produces an empty, 1 MB file named image. Reduce the count value to make a smaller image. The bs option to dd does not have to match your filesystem's block size; it just indicates how large a piece dd writes at a time.

When you have an mklab5fs written, you can issue:

```
$> ./mklab5fs image
```

```
./mklab5fs: created LAB5 filesystem on 'image'
```

Your mklab5fs should output a similar message on success, or an error message on failure (if image does not exist, or is too small, for example).

Once you've written a filesystem implementation (or part of it), you can try installing the module and mounting the filesystem:

```
$> insmod lab5fs.o
$> mount -t lab5fs image /mnt -o loop
```

When you are done testing your filesystem, unmount it:

```
$> umount /mnt
```

Make sure you're not inside /mnt when you try to unmount your filesystem, or you won't be able to do so.

References:

Section 10.6 of the Tanenbaum book covers generic Unix filesystem concepts. If you're not familiar with the basics of Unix filesystem design and semantics, review this section.

Understanding the Linux Kernel contains a lot of information on filesystem implementation, but it is not everything you need to write your filesystem.

- Read Chapter 12 to get an overview of the virtual filesystem (VFS) abstractions, but do not try to understand everything the first time.
- Learn Block Device Drivers from Chapter 14. Since your filesystem will reside on a block device, you'll need some of these functions to manage its contents.
- Read Page Cache and Buffer Head Data Structures from Chapter 15. You won't need to worry about the internals of these structures, but they may help you understand what you're doing as you handle struct buffer_head objects in your filesystem.
- Chapter 16 is useful for understanding the mechanisms of reading/writing a file.
- Read Chapter 18's section on the ext2 filesystem. This chapter should make it clearer how the VFS layer, the page cache, and other kernel features fit together in a real filesystem.

Linux 2.6.9 contains a multitude of filesystem implementations, from simple to complex. Since the implementations all fit into the VFS layer, at least at the surface they will have similar designs. If you're confused about how to do something, examine the way in which other filesystems implement it.

Hints:

Make sure to back up your source code frequently to a location off the virtual machine, preferably before each attempted installation of your kernel module. Do not assume that potential damage will be limited to the contents of your filesystem. The virtual machines are not backed up, so if you don't have a copy of your source code somewhere else, it could be lost.

Once you see an oops or other indication something went wrong, reset the system; don't try to do more work in the face of potential corruption. Similarly, if your filesystem becomes unmountable for any reason, reboot.

There are lots of examples of filesystems in the Linux kernel source tree. Ramfs is a good example of a small filesystem (but beware, it does not store any persistent state). cramfs has persistent storage, but it is read-only, so it is only a little more complex, and ext2 is a reasonable example of a large filesystem.

As usual, the kernel provides many abstractions and data types for you to use. Feel free to use any of these as you wish; however, do not directly copy the internals of other filesystem implementations.

Submission: you need to assemble the following files to hand in:

- lab5fs.c, which contains the C source code for the lab5fs module
- mklab5fs.c, which contains the C source code for the mklab5fs utility
- lab5fs.h, which declares data structures shared between the filesystem module and mklab5fs
- Makefile, which should produce lab5fs.o and mklab5fs.
- README, which consists of two parts:
    - [5] create a text file that briefly describes what you did, and how the pieces of your filesystem implementation fit together. If there are any known bugs in your implementation that you were unable to fix, or you could not fully complete the assignment, please describe what does and doesn't work correctly in this file as well.
    - [5] Describe your filesystem's on-disk and in-memory layout in this file. Document the limits of your filesystem (number of files, number of inodes, file size, number of disk blocks, and so forth).

All the above files should be in a new directory “lab5” relative to the root directory of the kernel source code “linux-2.6.9lab5”. You should tar the directory “lab5” as follows

```
$> tar zcf cse5433lab5.tar.gz lab5
```

Then submit the tarball file using the following command:

```
$> submit c5433aa lab5 cse5433lab5.tar.gz
```

Cleanup: Since we do not have enough disk space in each virtual machine, you need clean up the virtual machine after my evaluation for each of your lab solutions. First, delete the lab working directory at /usr/src/kernels/linux-2.6.9lab#. Then, delete the directory of installed modules at /lib/modules/linux-2.6.9lab#.