1 Overview

Many file systems provide a `fsck` (file system check) utility to check for and repair errors in the file system at mount time. For this lab, you will create your own rudimentary `fsck` utility to identify, parse, read, and manipulate an on-disk image of an ext2 file system.

Before starting work on this lab, read *Fsck - the UNIX file system check program* by McKusick & Kowalski 1994, revised 1996 (this paper is available in the handout tarball in the doc/ directory). Although this paper refers to the BSD FFS `fsck` utility, most of the concepts hold valid for the ext2 file system and many other file systems.

You will complete this lab on your own. You are required to be familiar with and comply with the policies described on the class web page, including the policies about (not) cheating; please read it again now.

2 Writing your own `fsck`

To help you write your own `fsck` tool, we have broken down the lab in four parts. Completing each part will bring you closer to having a fully functional `fsck` tool. These parts are not independent; each of the parts II, III and IV depend on the previous part.

2.1 Part I: Read the partition table (10 points)

You have been provided with a disk image (see section *Handout Code*). Build a tool to read and *print out* both - the DOS-style partition table located on sector 0, and the extended partition information (pointed to by partition 4) for extended partitions 5 and 6. You can verify that you have done this correctly when your code produces the following values for the provided disk image:

Partition number	Partition type	Start	Length
1	0x83 (ext2)	63	48132
2	0x00 (unused)	0	0
3	0x83 (ext2)	48195	48195
4	0x05 (Extended)	96390	64260
5	0x82 (Linux swap)	96453	16002
6	0x83 (ext2)	112518	48132

As you can see, partitions 1, 3, and 6 contain an ext2 file system.

The executable for your tool should be named `myfsck`. For this part, it should take two input parameters - the partition number, and the path of the disk image.

```
./myfsck -p <partition number> -i /path/to/disk/image
```

The output should be the appropriate start sector and length. For grading, we require you to use ‘‘`0x%02X %d %d\n`’’ as the format-specifier string in `printf`. Any deviation from this format might result in the auto-grader not being able to grade your submission.

```
./myfsck -p 1 -i diskimage
0x83 63 48132
```

Running `myfsck` on a partition that does not exist should return -1.

```
./myfsck -p 10 -i diskimage
-1
```

Your code for this part will be tested against multiple different partition tables.

2.2 Part II: Read the file system structures (0 points)

Though this part will not be graded, completing it will help you develop and test the code infrastructure needed to successfully create the final version of your tool. We recommend that you do not skip this step. In general, each part depends on the functionality of the part before it.

For this part, you should extend your tool from the previous section to accomplish the following tasks -

- Read the ext2 superblock for partition 1. This is located at offset 1024 bytes into the partition (not the disk) and is of size 1024 bytes. *Print out* the superblock magic number and verify that it is correct.
- Determine how to translate an inode number into its equivalent sector number. To do this you will need to read the correct entry in the inode table - which itself is split across multiple block groups. (The first inode in the inode table is inode number 1, not 0.)
- Determine how to locate an inode's (block's) entry in the inode allocation bitmap (block allocation bitmap).
- Locate and read the root inode. Verify that the inode attributes show that it is a directory inode. Verify that this inode is allocated in the inode allocation bitmap.
- Read the directory information pointed to by the root inode (Use `struct ext2_dir_entry_2`). Determine the directory entry that points to the directory `/lions`.
- Read the inode for the directory `/lions` and determine the inode number. This should be inode 4017. Verify the attributes of this inode.
  Continue this process until you find the file `/lions/tigers/bears/ohmy.txt`. Determine the inode number of this file - it should be 4021. Verify that the data blocks for this file are allocated in the block allocation bitmap.

- *Print out* the inode number for the file `/oz/tornado/dorothy`. What is special about this file?
- Determine the inode number for the file `/oz/tornado/glinda`. Verify that this file's type is a symbolic link. What is the name of the file which this link references?

2.3 Part III: Correcting errors on a disk image with well-known errors (50 points)

As you verified in Part I, the given disk image contains ext2 file systems on partitions 1, 3, and 6. The files and the directory structure on these partitions are similar but not necessarily the same.

Partition 1 contains a file system with no errors. Partitions 3 and 6 contain file systems with various errors.

For this part, extend your tool to check for specific file system errors listed below. Your tool should make four "passes", checking for the specified errors in each pass. When an error is found, you should print a description of the error to `stdout` and automatically fix the error. Your tool must only generate output when detecting and repairing errors - in other words, it should generate no output for a correct file system.

- **Pass 1: Directory pointers** (see McKusick & Kowalski, section 3.7). Verify for each directory - that the first directory entry is "." which self-references, and that the second directory entry is ".." which references its parent inode. If your tool finds an error, it should print a *short* description of the error, and correct the entry.
- **Pass 2: Unreferenced inodes** (section 3.5). Check that all allocated inodes are referenced in a directory entry somewhere. If you find an unreferenced inode, place it in the `/lost+found` directory - the name of this new entry in the directory should be the inode number. (i.e., if inode number 1074 is an allocated but unreferenced inode, create a file or directory entry - `/lost+found/1074`.)
- **Pass 3: Inode link count** (section 3.5). Count the number of directory entries that point to each inode (e.g., the number of hard links) and compare that to the inode link counter. If your tool finds a discrepancy, it should print a *short* description of the error, and update the inode link counter.
- **Pass 4: Block allocation bitmap** (section 3.3). Walk the directory tree and verify that the block bitmap is correct. If your tool finds a block that should (or should not) be marked in the bitmap, it should print a *short* description of the error, and correct the bitmap.

For this part, running the following command should fix disk errors on the specified partition.

```
./myfsck -f <partition number> -i <disk image file>
```

If the user specifies `-f 0`, your tool should correct disk errors on every ext2 partition contained in the disk image.

If you run your tool against the file systems on partitions 3 and 6, you should find one of each error (two on one file system, two on the other). To make sure that your tool fixes all errors, run it on these two partitions to fix the errors and then run the version of `fsck` provided by the system on the image (See the Handout Code section). If no errors are returned, your tool works.

This part is worth **50 points**. We will first run your `myfsck` utility on a disk image and expect it to find and correct errors. We will then run the system `fsck` on this corrected disk image to determine if your utility has fixed all errors. You will be penalized 2 points for every error our (system) `fsck` finds in the disk image.

2.4 Part IV: Correcting errors on a disk image with random errors (40 points)

In this part, your tool will be evaluated by running it against the provided disk images, but with random errors inserted. Errors will be inserted using the script `insert_errors.pl` (See section *Handout Code*). All four types of errors from Part III may be inserted randomly.

This part will be graded the same way as Part III. We will first run your `myfsck` utility on a disk image and expect it to fix errors. You will be penalized 2 points for every error that our (system) `fsck` finds on this fixed disk image.

3 Deliverables

All deliverables are due by **11:59 PM** on the due date. The submission **must** be made via **Autolab**. Late submissions will not be accepted. No extensions will be granted.

You should create a tarball - `myfsck.tar` of all your files. The tarball should contain the source code for your tool and a `Makefile` to build your tool. Be sure to include all header files and any supplementary files required for compilation. In order words, given your tarball, a TA must be able to do the following -

```
% tar xf myfsck.tar
% make
% ./myfsck -p 1 -i /path/to/some_disk_image
% ./myfsck -f 0 -i /path/to/some_disk_image
```

Your tool **must** compile on the Linux systems at `unix.andrew.cmu.edu`. Please **do not** include the disk image from the handout code, any binary files or pdfs in your handin tarball.

In order to help you verify that your submission meets the requirement above, we have provided a script - `last_check.sh`. Before submitting your lab, run this script under the same directory as your `myfsck.tar` and make sure it completes successfully.

Once your `myfsck.tar` is ready, you can submit it at `https://autolab.cs.cmu.edu/15746-s14/fscklab`. You can submit as many times as you wish. Your submission will be graded as soon as you submit it. Please let us know immediately if you face any errors while submitting your code.

In your tarball, please include a file `suggestion.txt` to tell us, what you liked and disliked about this lab. Suggestions for changes to the lab (to make it easier to understand or more challenging, etc.) are welcome.

4 Handout Code

The handout code on Autolab contains the following files:

- `genhd.h` and `ext2_fs.h`: These are the relevant header files from the Linux 2.4.17 kernel distribution. These files contain the structures that you will need in order to interpret the on-disk file system organization. You might not need to use all the structures and `#define`s from these files.
- `readwrite.c`: This is a stub program that will read and write one "sector" at at time from the disk file. You should use this program as the basis for your `fsck` tool.
- `disk`: This is a disk image that contains 6 partitions. You should use this disk for your tool development and testing. Assume that the disk sector size is 512 bytes, and that sector numbering starts at 0.
- `run_fsck.pl`: This is a script that will run the system fsck against a partition on the provided disk and report errors. It will NOT modify the disk image. The usage options are:

  ```
  ./run_fsck.pl --partition <valid partition number> --tmp_dir
    /path/to/a/temp/directory --image /path/to/disk/image
  ```

  When <`valid partition number`> is set to zero, the system fsck will be run against all ext2 partitions on the image.
- `insert_errors.pl`: This is a script which will insert random errors to the disk image. By default, insert_errors.pl inserts random errors into partition 1 of the disk. The usage options are:

  ```
  ./insert_errors.pl --config_file /path/to/config --image
    /path/to/disk/image --tmp_dir /path/to/a/temp/directory
  ```
- `p1_files_and_dirs.cfg`: This is the configuration file for `insert_errors.pl`.
- `last_check.sh`: This script will check whether your tarball meets the grading requirement. It only takes one arguement: the absolute path of the disk image. You are strongly recommended to run this script before submitting the tarball to Autolab.

5 FAQ

- **Question:** Why is the partition table I see for the disk image different from that stated in the handout?
  Answer: The partition table listed in the handout does not include an entry for the second extended partition contained in the disk image. An extended partition looks like a linked-list. In each extended boot record (EBR), the first entry describes an information about a current partition and the second entry describes a way to find the next extended partition, if available.

  The complete partition table for the provided disk image is

Partition number	Partition type	Start	Length
1	0x83 (ext2)	63	48132
2	0x00 (unused)	0	0
3	0x83 (ext2)	48195	48195
4	0x05 (Extended)	96390	64260
5	0x82 (Linux swap)	96453	16002
-	0x05 (Extended)	112455	48195
6	0x83 (ext2)	112518	48132

- **Question:** What scheme is followed in assigning numbers to partitions?
  Answer: Partition numbers are assigned as follows - All primary partitions in the master boot record are numbered 1 through 4 (irrespective of whether it is an empty or an extended partition). Next, all logical partitions are numbered starting from 5. All non-primary extended partitions are excluded.

- **Question:** How can I mount the disk image as a normal partition under Linux?
  Answer: If you want to use the provided disk image for debugging, you can mount a specific partition under Linux -

  – Extract the target partition from the disk image:
  Using your part I output, you can find the starting sector of a particular partition and the number of sectors it has. You can then extract this partition using the `dd` command:

  ```
  dd if=/path/to/disk/image of=hda.img bs=512 skip=[starting sector]
  count=[#sectors]
  ```

  – Mount the extracted image in a directory (e.g. `/mnt/test/`)

  ```
  mount -o loop hda.img /mnt/test/
  ```

  You can now access `/mnt/test` as a normal partition under Linux.

6 Resources

You should use the Internet (and the comments in the Linux header files) to figure out how to interpret the on-disk data structures. Here are some suggested starting points (this list is neither exhaustive nor necessarily the best source of information):

- Information on partition tables:
  `http://www.tldp.org/HOWTO/Large-Disk-HOWTO-6.html`
  `http://www.tldp.org/HOWTO/Large-Disk-HOWTO-13.html`
- Information on ext2 file system internals:
  `http://www.tldp.org/LDP/tlk/fs/filesystem.html`
  `http://homepage.smc.edu/morgan_david/cs40/analyze-ext2.htm`
  `http://www.nongnu.org/ext2-doc/ext2.html`
  `http://wiki.osdev.org/Ext2`
- Information about MBR, EBR, EXT2 :
  `http://en.wikipedia.org/wiki/Master_boot_record`
  `http://en.wikipedia.org/wiki/Extended_boot_record`