CS111 - Project 3A: File System Dump

INTRODUCTION:

Project 3 is expected to be the most difficult project, where you will develop programs to analyze file systems and diagnose corruption. In part A, you will produce a program to read in the image of a file system, analyze it, and summarize its contents in several csv files. These csv files will be used in part B, where we will diagnose the problems.

Part A can be broken into two major steps:

1) Mount the provided image file on your own Linux and investigate it with *debugfs(8)*.
2) Write a program to analyze the image file and output a summary to six csv files (describing the super block, cylinder groups, free-lists, i-nodes, indirect blocks, and directories).

Project 3 may be done by a team of up to two students. We will use some specialized software to detect cheating for Project 3.

RELATION TO READING AND LECTURES:

This project more deeply explores the file and directory concepts described in chapter 39.
This project is based on the same EXT2 file system that is discussed in sections 40.2-40.5.
This project goes much deeper than the introductory discussion of integrity in sections 42.1-2.

PROJECT OBJECTIVES:

- reinforce the basic file system concepts of directory objects, file objects, and free space.
- reinforce the implementation descriptions provided in the text and lecture.
- gain experience with the examining, interpreting, and processing information in binary data structures.
- gain practical experience with complex data structures in general, and on-disk data formats in particular.
- (in 3B) reinforce the notions of consistency/integrity provided in the text and lecture.

DELIVERABLES:

A single tarball (.tar.gz) containing:

- a single C source module that compiles cleanly (with no errors or warnings).
- a Makefile to build the program and the tarball.
- a README file describing each of the included files, both team members' UIDs, and any other information about your submission that you would like to bring to our attention (e.g. limitations, features, testing methodology, use of slip days).

PROJECT DESCRIPTION:

To avoid having to deal with kernel code, this project will be done entirely in user mode. Linux supports the creation, mounting, checking, and debugging of file systems stored in ordinary files. In this project, we will provide EXT2 file system images in ordinary files. Because they are in ordinary files (rather than protected disks) you can access/operate on those file system images with ordinary user mode code.

To complete this assignment, you will need to learn a few things:

- *debugfs(8)* (http://man7.org/linux/man-pages/man8/debugfs.8.html)
- *pread(2)*
- csv format
- EXT2 file system (http://www.nongnu.org/ext2-doc/ext2.html)

Step 1 - study a file system image

Your program will be tested on multiple (broken) file systems. Here we provide one as an example. SEASnet servers do not support commands like *sudo(8), mount(8), or debugfs(8)*, so to play with our provided file system image, you will have to install a Linux distribution (if you do not have one) on your own computer. For simplicity and convenience, you may choose to install Linux inside a virtual machine, for example VirtualBox (it's free!).

Then download this image file, and mount it onto your **own** Linux, with the following commands:

```
mkdir fs
sudo mount -o loop disk-image fs
sudo chown -R $USER fs
```

Now, you can navigate the file system, just like an ordinary directory, with commands like *ls(1)*, *cat(1)*, and *cd(1)*. After you are done with it, you can unmount with the following command:

```
sudo umount fs
```

Before you start writing your C program to interpret the "disk-image" file, you can explore it further using *debugfs(8)* (in your own Linux). You may find many useful commands in its man page. Some particularly helpful ones are: *stats*, *stat*, *bd*, *testi*, and *testb*. If you have problems interpreting parts of the file system, you can use the *debugfs* program to help you understand its contents.

Here is a sample of things you may see in the image file:

- sparse files
- large files
- allocated data blocks full of zeroes

- unallocated blocks containing valid data
- files with data beyond their length
- files with long names
- files with syntactically strange names
- directories that span multiple blocks and have old entries for deleted files.

Step 2 - write a program to summarize the file system's contents

In this step, you will write a program called **lab3a** that:

- Reads in (only) one file system image according to the provided file name. For example, we may run your program with the above file system image using the following command: ./lab3a disk-image.
- Analyzes the provided file system image and outputs six csv files to the current directory. The contents of these csv files are described below. Your program **must** output these files with **exactly the same formats** as shown below. We will use sort and diff to compare your csv files with ours, so a different format will make your program fail the test and the content in the output csv file will be treated as error.

Please note that, although you cannot mount the provided image file on SEASnet servers, your lab3a program should be able to run on SEASnet servers, just like previous assignments.

There are to be six csv-format files, each summarizing a different part of the file system. Remember, you can always check your program's output against debugfs's output. All the information required for the summary can be manually found and checked by using debugfs.

If you feel confusing about certain fields' meanings, contact Zhaoxing so he can clarify that on this page[1].

1. **super block:** A basic set of file system parameters from super block.

fields(9)	format
magic number	hex
total number of inodes	dec
total number of blocks	dec

[1] These field names were chosen by other people, so don't blame him. ;-) In fact he has to read the reader's sample solution to understand what are these fields… And he is trying his best to help you guys by explaining how the sample solution processes the entries in those tables. Keep an eye on the footnotes, they are key information for this assignment. However, please keep in mind that he may be wrong in understanding the code, so the sample csv files should be the only source that could be considered official. Zhaoxing has asked the reader to explain the code, but no reply yet.

block size	dec
fragment size	dec
blocks per group	dec
inodes per group	dec
fragments per group	dec
first data block	dec

2. **group descriptor:** Information for each group descriptor.

fields(7)	**format**
number of contained blocks	dec
number of free blocks	dec
number of free inodes	dec
number of directories	dec
(free) inode bitmap block	hex
(free) block bitmap block	hex
inode table (start) block	hex

3. **free bitmap entry:** A list of free inodes and free blocks[2].

fields(2)	**format**
block number of the map	hex
block/inode number	dec

4. **inode:** Key information for each allocated inode[3].

fields(11+15)	**format**
inode number	dec

[2] Your code shall output all the entries marked as 0 in the bitmap, no matter they are really free entries or not.
[3] By “allocated”, we mean that inode is marked as 1 in the bitmap, no matter it is a valid inode or not.

file type	char[4]
mode	oct
owner[5]	dec
group	dec
link count	dec
creation time	hex
modification time	hex
access time	hex
file size	dec
number of blocks[6]	dec
block pointers * 15	hex

5. **directory entry:** The valid/allocated entries in each directory[7].

fields(6)	format
parent inode number	dec
entry number[8]	dec
entry length[9]	dec

[4] We only recognize a (small) subset of the file types: 'f' for regular file, 'd' for directory, and 's' for symbolic links. For other types, use '?'.
[5] The owner id and group id could be changed after using command *chown*.
[6] This means the number of **file system** blocks, with the size specified in super block.
[7] Here by "valid" we mean the inode number of the file entry is not 0. The sample solution's implementation for this table is very interesting, and it's hard for you guys to come up with the same code. So your kind TA Zhaoxing decided to help again. The sample solution uses a for loop to iterate through all inodes outputted in previous step. If the inode's file type is directory, then uses a second for loop to traverse all blocks (here the number of blocks needs to be traversed is the block number registered in the inode) for that inode. Here the code traverses all blocks given by the pointers, no matter they are true data blocks or not. The code always counts the entry numbers no matter the directory entry is valid or not (can you think about why the code does this way?), but only outputs valid directory entries to the csv file. If the code needs to access some data block x through an indirect block, but the indirect block pointer's value is 0, then the code just assumes the data block x's number returned from the indirect block is 0.
[8] Entry numbers always start from entry number 0 within each directory. So in each directory, first directory entry has entry number 0, second directory entry has entry number 1. When traversing the data blocks of one directory, do not forget to check the data block pointer (see if it is zero) before your code dereferences that pointer!
[9] This is the rec_len variable.

name length	dec
inode number of the file entry	dec
name	string[10]

6. **indirect block entry:** These are all the **non-zero** block pointers in an indirect block[11]. The blocks that contain indirect block pointers are included[12].

fields(3)	**format**
block number of the containing block	hex
entry number[13]	dec
block pointer value	hex

If you are confused about the above fields, you may refer to the EXT2 documentation linked to above or contact Zhaoxing so he can clarify that on this page.

For each different kind of entry, output its summary in a separate file. The names of your files should be:

- super.csv ...for super block;
- group.csv ...for group descriptors;
- bitmap.csv ...for free bitmap entries;
- inode.csv ...for inodes;
- directory.csv ...for directory entries; and
- indirect.csv ...for indirect block entries.

[10] Double-quoted.

[11] Here your code only needs to investigate the indirect block pointers corresponding to the number of blocks registered with the inode. So if the inode says its number of blocks is only 10, then your code should just skip this inode.

[12] You kind TA Zhaoxing believes it is impossible for you to write the same code as the sample solution, so he decided to help again. If the block pointer value is 0 (guess why 0 is not a valid value, even when block size is greater than 1KiB?), then do not output this entry. Here if the indirect block pointer (index 12-15 in block pointers array)'s value is 0, then the code just reads from block 0, assuming it is a block containing block pointers… However, if the code (thinks) it is reading a block containing single indirect block pointers (this block is pointed by a double indirect block pointer), and if the current single indirect block pointer (remember this pointer points to a block containing block pointers) is 0, then the code just skips this pointer instead of treating it as a block containing block pointers. The same rule also works for triple indirect block pointer. So if the code (thinks) it is reading a block containing double indirect block pointers (this block is pointed by a triple indirect block pointer), and if the current double indirect block pointer (remember this pointer points to a block containing single block pointers) is 0, then the code just skips this pointer.

[13] Entry number starts from 0 within each indirect block. So the first entry in each indirect block has entry number 0. Although your code shall not output entries whose pointer value is 0, these entries still have corresponding entry numbers.

In these csv files, each line represents one entry, and should be ended with a single new-line character ('\n', 0x0a). The field orders in those csv files should be the same as listed above, and your program should **use and only use** a comma to separate each field.

Here we provide one sample output line for each csv file, the data here may be different from what your program outputs:

1. super block
   ef53,5136,20480,1024,1024,8192,1712,8192,1
2. group descriptor
   8192,1796,1682,2,4,3,5
3. free bitmap entry
   3,253
4. inode
   1,?,0,0,0,0,572eabbd,572eabbd,572eabbd,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5. directory entry
   2,3,12,2,12,"C1"
6. indirect block
   e8,0,2a3

If your program encounter an obvious inconsistency, it may output an error message to stderr. However, we will only be grading the above-mentioned 6 files: we will run your program on multiple file system images, and check your results with the correct answers.

For your convenience, here are the exact 6 csv files generated by our solution: super.csv, group.csv, bitmap.csv, inode.csv, directory.csv, and indirect.csv.

SUBMISSION:

Please carefully check the Rubric section (especially the **Code review** subsection) and **all (latest) footnotes** before submitting your code.

Project 3A is due before midnight on Monday, May 16, 2016.

Your tarball should have a name of the form lab3a-*studentID*.tar.gz and should be submitted via CCLE. If you did this project with another student, then either one's UID is acceptable. A team may **only** submit one tarball. We will deduct penalty points for double submission.

We will test your work on a SEASnet GNU/Linux server running RHEL 7 (this is on lnxsrv09). You would be well advised to test your submission on that platform before submitting it.

RUBRIC:

Value	Feature
	Packaging and build (10%)
3%	untars expected contents
3%	clean build w/default action (no warnings)
2%	Makefile has clean and dist targets
2%	reasonableness of README contents

Code review (15%)

You can check header files like ext2_fs.h to have an idea about the variable types. HOWEVER, you shall write up your own code for the calculations like block/inode (number), indirect block pointer, various different kinds of offset, and so on. So in sum, do all the calculations with your own code, do not use macros/functions provided in header files like ext2fs.h.

5%	overall readability and reasonableness, correct program name and csv filenames
10%	image file investigation correct (use pread() and write your own code for calculations)

Results (75%)

⅔ points shall go to outputting the required fields correctly, and ⅓ points go to correct lines (penalty will be given for extra/missing lines).

10%	super.csv
10%	group.csv
10%	bitmap.csv
15%	inode.csv
15%	directory.csv
15%	indirect.csv