CS111 - Project 3B: File System Analysis

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 produced a program to read in a file system image, analyze it, and summarize its contents in several csv files. In part B, you will write a program to analyze these csv files to diagnose the problems in the provided file system image.

Unlike previous assignments, you have the freedom to use whatever programming language you'd like, as long as that programming language is supported by SEASnet servers.

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.
- reinforce the notions of consistency/integrity provided in the text and lecture.

DELIVERABLES:

A single tarball (.tar.gz) containing:

- the source code for Project 3B.
- a Makefile to build the tarball (and compile the code).
- a README file describing each of the included files, both team members' UIDs, the shell command to (compile and) execute your program, 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:

In Project 3B, you will write a program called **lab3b** that:

- Reads in the six csv files your lab3a program produced in Part A from the current directory, checks for certain file system corruptions listed in detail below, and outputs the error reports to a file called “lab3b_check.txt” to current directory.

You can use whatever programming language you prefer, as long as that programming language is supported by SEASnet server lnxsrv09. Please note that we will not install any programming language or libraries for you; you can only use the available programming languages and libraries on lnxsrv09. The shell command to (compile and) execute your program should be provided in both your README file and in Makefile (see the paragraph before Submission section for more detail). For example, if you use Python[1], your filename should be “lab3b.py”, and you should provide a shell command like “python lab3b.py”.

We will test your lab3b program on our own csv files, so bugs/errors in your lab3a program would not affect your Part B score. When testing your code, we will put the six csv files into the same directory as your program, and after executing the shell command provided by you, there should be a file called “lab3b_check.txt” in the same directory. We will use *sort* and *diff* to compare this file with ours. Failing to automatically read in the six csv files or generate the error report .txt file will automatically result in a zero for this project.

Here is the list of errors your lab3b program should check and the corresponding error report format:

1. **unallocated block**: blocks that are in use but also listed on the free bitmap. Here the INODEs should be listed in increasing order of the inode_num.

   `UNALLOCATED BLOCK < block_num > REFERENCED BY (INODE < inode_num > (INDIRECT BLOCK < block_num[2]>) ENTRY < entry_num[3] >)` * n

   For example,

   `UNALLOCATED BLOCK < 1035 > REFERENCED BY INODE < 16 > ENTRY < 0 > INODE < 17 > INDIRECT BLOCK < 10 > ENTRY < 0 >`[4]

[1] If you use Python, please make sure that the version you use (2 or 3) is available on lnxsrv09!
[2] See the definition of block_num in error type 7.
[3] If the block pointer to this block (whose block number is the first block_num your code print in this error type) is within the first 12 direct block pointers of an inode, then the entry_num here should be 0-11. If the block pointer is within a block pointer block, then the entry_num here should be 0-(block_size/4-1). For example, if the block size is 1024 bytes, then entry_num here should be 0-255.
[4] These numbers were handmade to demo the format, not true values.

2. **duplicately allocated block**: blocks that are used by more than one inodes. Here the INODEs should be listed in increasing order of the inode_num.

   `MULTIPLY REFERENCED BLOCK < block_num > BY (INODE < inode_num > (INDIRECT BLOCK < block_num`[5]`>) ENTRY < entry_num`[6] ` >)` * n

   For example,

   `MULTIPLY REFERENCED BLOCK < 613 > BY INODE < 24 > ENTRY < 0 > INODE < 25 > ENTRY < 0 > INODE < 26 > ENTRY < 0 >`

3. **unallocated inode**: inodes that are in use by directory entries (the inode number of the file entry field) but not shown up in inode.csv. Here the DIRECTORYs should be listed in increasing order of the inode_num.

   `UNALLOCATED INODE < inode_num > REFERENCED BY (DIRECTORY < inode_num`[7] `> ENTRY < entry_num >`[8]`)` * n

   For example,

   `UNALLOCATED INODE < 21 > REFERENCED BY DIRECTORY < 2 > ENTRY < 12 >`

4. **missing inode**: inodes that are not in use, and not listed on the free bitmap.

   `MISSING INODE < inode_num > SHOULD BE IN FREE LIST < block_num >`

   For example,

   `MISSING INODE < 34 > SHOULD BE IN FREE LIST < 4 >`

5. **incorrect link count**: inodes whose link counts do not reflect the number of directory entries that point to them.

   `LINKCOUNT < inode_num > IS < link_count > SHOULD BE < link_count >`

   For example,

   `LINKCOUNT < 1714 > IS < 3 > SHOULD BE < 2 >`

6. **incorrect directory entry**: the '.' and '..' entries that don't link to correct inodes.

---

[5] See the definition of block_num in error type 7.
[6] Same as the entry_num definition in error type 1.
[7] This is the parent inode number field for the directory entry.
[8] This is the entry number field for the directory entry.

`INCORRECT ENTRY IN < inode_num[9] > NAME < entry_name > LINK TO < inode_num[10] > SHOULD BE < inode_num >`

For example,
`INCORRECT ENTRY IN < 1714 > NAME < . > LINK TO < 1713 > SHOULD BE < 1714 >`

7. **invalid block pointer**: block pointer that has an invalid block number.[11]
`INVALID BLOCK < block_num > IN INODE < inode_num > (INDIRECT BLOCK < block_num[12] >) ENTRY < entry_num[13] >`

For example[14],
`INVALID BLOCK < 1 > IN INODE < 2 > INDIRECT BLOCK < 3 > ENTRY < 4 >`
or
`INVALID BLOCK < 1 > IN INODE < 2 > ENTRY < 4 >`

In “lab3b_check.txt”, each line represents one error, and should be ended with a single new-line character (‘\n’, 0x0a). Please pay attention to the spaces between words, numbers, and symbols. Incorrect formatting in your “lab3b_check.txt” will be treated as error content.

For your convenience, here is the “lab3b_check.txt” generated by our solution: lab3b_check.txt. Here are the six csv files we used in Project 3A: super.csv, group.csv, bitmap.csv, inode.csv, directory.csv, and indirect.csv.

Same as Project 3A, if you feel confusing about something, please contact Zhaoxing via email. He tried his best to read the sample solution to figure out what you should do in this assignment, what on earth are these entry numbers (there is no document, not even a single line, to describe these entry numbers![15]), and then write up the spec to explain these things! He is adding his

[9] This is the parent inode number field for the directory entry.
[10] This inode_num, as well as the following inode_num, is the inode number of the file entry field for the directory entry.
[11] If a block pointer points to block number 0, or a block number greater than the largest block number in the file system, then this an error in this format should be reported.
[12] If this invalid block pointer is stored in an indirect pointer block (a block that stores block pointers), then the output the block number here. The block_num here should always be non-zero, and your code only prints INDIRECT BLOCK < block_num > when block_num is non-zero.
[13] Similar as the entry_num definition in error type 1. However, besides the previous mentioned cases, here your code detects the validity of last three block pointers in inode’s i_block array. If any of them points to block 0, then outputs an error in this format, sets the indirect block number as 0 (so your code does not print INDIRECT BLOCK < block_num>), and sets the entry_num as 12, 13, or 14 (the index in the i_block array).
[14] The example numbers here are not real values, so you will not see this from the provided csv files.
[15] There is no explanation in the spec draft, and the reader who wrote the sample solution did not give an explanation either.

understanding as footnotes, but his understanding may not be correct. So the sample output file should be considered as the only official source to understand this assignment.

UPDATE: The error formats for many error types were changed on Thursday and Friday (the last modification time is Friday 1:53 AM), so please modify your code according to the new format. As a result, the sample lab3b_check.txt has been changed, so please re-download it (the latest version was uploaded at Thursday 11:10 PM).

In your Makefile, the default action should compile your code (if you have to do so). Also, we will use "**make run**" to execute your program, so please make sure your Makefile does support this.

SUBMISSION:

Project 3B is due before midnight on **Wednesday, May 25**, 2016.

Your tarball should have a name of the form lab3b-*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 (13%)
3%	untars expected contents
6%	correct Makefile to (compile and) execute your program with no warning/error
2%	Makefile has clean and dist targets
2%	reasonableness of README contents
	Code review (10%)
5%	overall readability and reasonableness
5%	correct program name, csv filenames, and the error report file name
	Results (77%)

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).

11%	unallocated block
11%	duplicately allocated block
11%	unallocated inode
11%	missing inode

11% incorrect link count
11% incorrect directory entry
11% invalid block pointer