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.

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, your UID, 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, etc.).

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, noting that the particular numbers shown below are merely examples, and do not correspond to numbers in the actual assigned file system:

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>) ENTRY < entry_num >) * n
```

For example,

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

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>) ENTRY < entry_num >) * 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 > ENTRY < entry_num >) * 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.
`INCORRECT ENTRY IN < inode_num > NAME < entry_name > LINK TO < inode_num > 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.
`INVALID BLOCK < block_num > IN INODE < inode_num > (INDIRECT BLOCK < block_num >) ENTRY < entry_num >`

For example,
`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.

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 **Monday, November 28**, 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

---

[1] If you use Python, please make sure that the version you use (2 or 3) is available on lnxsrv09!