File Systems Exercise 3

- Due Mar 29 by 10pm
- Points 0.5
-

Introduction

This exercise follows on from Tutorial Exercise 8. **If you wait until the deadline to do this exercise it is unlikely that you will be able to finish the assignment 4.**

You may work in pairs for this exercise, with your A4 partner, since some code will potentially be common with the assignment. MarkUs will only create the appropriate directory in your repository when you log into MarkUs and either create your group, or declare that you will work alone. The groups will get a new shared repository, and the students working solo may also get a new repository. Please log into MarkUs well before the deadline to take these steps. (If you create the directory in your repo yourself, then MarkUs won't know about it and we won't be able to see your work.)

It is your responsibility to log into MarkUs *before* the exercise deadline to ensure that you know where to commit your work, and so that MarkUs can connect your work to the grading system.

Requirements

Your final task in this series of exercises is to print the **directory block** contents. While you are doing this, you should work out how the directory structure creates a linked list of directory entries by using `rec_len` to show where the next directory entry begins. Note that `rec_len` in the last directory entry takes you to the end of the block.

When printing the blocks, first print the root directory. Then for each inode that is in use, check if it is a directory. If it is a directory, print the contents of the directory block. You do not need to print anything for file blocks.

For `emptydisk.img` your output should *exactly* match the following. In other words, we should be able to use `diff` to compare your output to this and see that it is identical (the indentation below uses 4 spaces).

```
Inodes: 32
Blocks: 128
Block group:
    block bitmap: 3
    inode bitmap: 4
```

```
    inode table: 5
    free blocks: 105
    free inodes: 21
    used_dirs: 2
Block bitmap: 11111111 11111111 11111100 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000001
Inode bitmap: 11111111 11100000 00000000 00000000

Inodes:
[2] type: d size: 1024 links: 3 blocks: 2
[2] Blocks:  9

Directory Blocks:
   DIR BLOCK NUM: 9 (for inode 2)
Inode: 2 rec_len: 12 name_len: 1 type= d name=.
Inode: 2 rec_len: 12 name_len: 2 type= d name=..
Inode: 11 rec_len: 1000 name_len: 10 type= d name=lost+found
```

Note: The original `twolevel.img` was a bit confusing, so we uploaded a `multilevel.img` to the images directory and changed the output below.

For `multilevel.img` your output should match the following:

```
Inodes: 32
Blocks: 128
Block group:
    block bitmap: 3
    inode bitmap: 4
    inode table: 5
    free blocks: 102
    free inodes: 17
    used_dirs: 4
Block bitmap: 11111111 11111111 11111111 00000000 10000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000001
Inode bitmap: 11111111 11111110 00000000 00000000

Inodes:
[2] type: d size: 1024 links: 4 blocks: 2
[2] Blocks:  9
[12] type: d size: 1024 links: 3 blocks: 2
[12] Blocks:  23
[13] type: d size: 1024 links: 2 blocks: 2
[13] Blocks:  24
[14] type: f size: 0 links: 1 blocks: 0
[14] Blocks:
[15] type: f size: 39 links: 1 blocks: 2
```

```
[15] Blocks:  33

Directory Blocks:
   DIR BLOCK NUM: 9 (for inode 2)
Inode: 2 rec_len: 12 name_len: 1 type= d name=.
Inode: 2 rec_len: 12 name_len: 2 type= d name=..
Inode: 11 rec_len: 20 name_len: 10 type= d name=lost+found
Inode: 12 rec_len: 36 name_len: 6 type= d name=level1
Inode: 15 rec_len: 944 name_len: 5 type= f name=afile
   DIR BLOCK NUM: 23 (for inode 12)
Inode: 12 rec_len: 12 name_len: 1 type= d name=.
Inode: 2 rec_len: 12 name_len: 2 type= d name=..
Inode: 13 rec_len: 1000 name_len: 6 type= d name=level2
   DIR BLOCK NUM: 24 (for inode 13)
Inode: 13 rec_len: 12 name_len: 1 type= d name=.
Inode: 12 rec_len: 12 name_len: 2 type= d name=..
Inode: 14 rec_len: 1000 name_len: 5 type= f name=bfile
```

For `largefile.img` your output should match the following:

```
Inodes: 32
Blocks: 128
Block group:
    block bitmap: 3
    inode bitmap: 4
    inode table: 5
    free blocks: 90
    free inodes: 20
    used_dirs: 2
Block bitmap: 11111111 11111111 11111100 00000000 00010000
00000000 00000000 00000111 00000010 00000000 00111111 11110000
00000000 00000000 00000000 00000001
Inode bitmap: 11111111 11110000 00000000 00000000

Inodes:
[2] type: d size: 1024 links: 3 blocks: 2
[2] Blocks:  9
[12] type: f size: 13440 links: 1 blocks: 30
[12] Blocks:  62 63 64 83 84 85 86 87 88 89 90 91 36 0 0

Directory Blocks:
   DIR BLOCK NUM: 9 (for inode 2)
Inode: 2 rec_len: 12 name_len: 1 type= d name=.
Inode: 2 rec_len: 12 name_len: 2 type= d name=..
Inode: 11 rec_len: 20 name_len: 10 type= d name=lost+found
Inode: 12 rec_len: 980 name_len: 13 type= f name=largefile.txt
```

Note that block list above for the file prints only the contents of the blocks array. An alternative approach would be to print the indices of the data blocks for the file: `62 63 64 83 84 85`

```
86 87 88 89 90 91 92 71
```

Submission

Ensure that you have a repository directory created by MarkUs for this exercise (fs3), then add and commit `readimage.c` and `ext2.h` to this directory.