File Systems Exercise 1

Due Friday by 10pm **Points** 0.5

Introduction

For Assignment 4, you will be manipulating an Ext2 file system image. The next three exercises are intended to help you get started on the assignment by helping you understand the structure of the file system. Please start early to fully take advantage of the tutorial time to ask questions.

Each of the three file systems exercises counts as one tutorial exercise mark.

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 manually 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

Please read the Assignment 4 handout. It has links to some resources that you will need to read to fully understand the ext2 structure, especially sections "Learning about the Filesystem" and "Mounting a filesystem".

You are given two files as starter code:

- ext2.h
- readimage.c

There are also a bunch of disk images available to you in /u/csc369h/winter/pub/a4/images on teach.cs.toronto.edu

- emptydisk: An empty virtual disk.
- onefile: A single text file has been added to emptydisk.
- deletedfile: The file from onefile has been removed.
- onedirectory: A single directory containing a text file has been added to emptydisk.
- hardlink: A hard link to the textfile in onedirectory was added.
- deleteddirectory: A recursive remove was used to remove the directory and file from onedirectory.
- twolevel: The root directory contains a directory called `level1` and a file called `afile`. `level1` contains a directory called `level2`, and `level2` contains a file called `bfile`.

- largefile: A file larger than 13KB (13440 bytes) is in the root directory. This file requires the single indirect block in the inode.

Note that `readimage` uses `mmap` to map the disk image file into memory. The superblock is in the second block on the disk starting at byte 1024, so we can interpret these bytes as the super block struct.

You should also look at these bytes of the one of the images, for example emptydisk.img by running `xxd emptydisk.img > emptydisk.txt` and using an editor to view the contents of the file. Try the following:

- Figure out where each block starts.
- See what the inode bit map and block bitmaps look like.
- Find other non-empty blocks to see if you can see what might be in them.

Your task for this exercise is simple. Add to `readimage` to print out the following fields from the block group descriptor. For `emptydisk.img` your output should look *exactly* like 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
```

Submission

Add, commit, and push `readimage.c` and `ext2.h` to your fs1 directory. You may include a Makefile if you want, but you should not commit any disk images.