The “file system” is actually created in a number of files named “fusedata.X” and stored on an existing filesystem. The X in the filename corresponds to a simulated “block number” so that if we were to store these files on a block device, fusedata.0 would be in block 0 (sector 0) and fusedata.1 would be in block 1 (sector1). The idea is to simulate working with a block device to store file system data.

We are limited in our block sizes to 4096 bytes, this will not change in the foreseeable future so you can trust this. For now, we will likely use a block device with only 10,000 blocks but this could easily change in the future so you should NOT trust this. We are going to use pointers in a lot of places in the system so I’m going to tell you that a pointer takes up as much as 10 bytes. Which means that the maximum number of pointers you could store in one block is 400 (just trust this for now). All blocks contain text, the only binary data would be the material that is stored in a file.

First, we guarantee that block zero will store the “super block” which tells us about all of the other blocks in the system. The super block has the following format:

{creationTime: 1429434844, mounted: 5, devId:20, freeStart:1, freeEnd:25, root:26, maxBlocks:10000}

Creationtime is the UNIX EPOCH time that the file system was formatted. Mounted is the number of times the file system was mounted. DevID is the type of file system. freeStart is the starting block of the free list (probably 1), and endFree is the ending block. All blocks between freeStart and freeEnd contain the data which make up the free block list. Root is the location of the root directory. MaxBlocks is the number of blocks in the file system. The size of the free block list is directly related to the maxBlocks because we need the ability to indicate that all blocks are free (10000/400=25).

Each directory block contains an directory inode which is in the following format:

{size:1033, uid:1000, gid:1000, mode:16877, atime:1323630836, ctime:1323630836, mtime:1323630836, linkcount:4, filename\_to\_inode\_dict: {f:foo:1234, d:.:102, d:..:10, f:bar:2245}}

The first letter of the name in the filename\_to\_inode\_dict will indicate that it is a file “f’ directory “d” or special “s”. This letter is NOT part of the filename, just a descriptor to tell you what type of inode to expect.

Size, uid, gid and mode are not important as they are for features which will be added later. Atime, ctime and mtime are access time, creation time and modification times stored in UNIX EPOCH format. Linkcount is the number of items in the directory.

Files need to be handled carefully in the file system, so each file will have an inode that describes it. The format for a file’s inode information will be:   
{size:1033, uid:1000, gid:1000, mode:33261, linkcount:2, atime:1323630836, ctime:1323630836, mtime:1323630836, indirect:0 location:2444}

(Again, uid, gid, mode and linkcount are for later use.) This data will be stored in the inode location referenced in the filename\_to\_inode\_dict entry in the parent directory. When a file is created, we must consider how large it is. Since our file system can only handle blocks of 4096 bytes or less, we have to use a level of indirection if we want to store a file larger than that size. Should we need to store a file of between 4097 and 1,638,400 bytes, we need to use an index block. The index block is simply an array of pointers to other blocks. Since we will not be handling anything larger than 1,638,400 bytes, and 400 pointers will easily fit into an index block, we need only a single level of indirection.

We will use the “indirect” field in the inode table to indicate if the “location” field is referencing file data or an index block. (Note, the comma is missing between indirect and location, I know)

The index block format and the free space list format will be a simple CSV format “X, Y, Z” to indicate that X, Y and Z are in the list. (Note: X, Y and Z are numbers like 1042, 5291 and 3).

To do this you need to use the FS Format.doc file to understand how the file system is laid out. You may do this assignment in PHP,Python, C,C++ or Java. You must submit ONLY ONE file, or if necessary a zip file.

As with any filesystem, there exists the possibility that errors will be introduced. In Linux, these errors are resolved using a File System ChecKer (fsck). Each fsck is custom designed for the file system type so that it can examine everything to make sure it is consistent.

For this homework, I would like you to design a file system checker for our file system. You should call it csefsck. It will have to do the following:

1. The DeviceID is correct
2. All times are in the past, nothing in the future
3. Validate that the free block list is accurate this includes
   1. Making sure the free block list contains ALL of the free blocks
   2. Make sure than there are no files/directories stored on items listed in the free block list
4. Each directory contains . and .. and their block numbers are correct
5. Each directory’s link count matches the number of links in the filename\_to\_inode\_dict
6. If the data contained in a location pointer is an array, that indirect is one
7. That the size is valid for the number of block pointers in the location array. The three possibilities are:
   1. size<blocksize if indirect=0 and size>0
   2. size<blocksize\*length of location array if indirect!=0
   3. size>blocksize\*(length of location array-1) if indirect !=0

Your code will be inspected closely for cheating and any cheating will result in a zero grade and be reported to university administration.