Throughout the next few assignments, we will be working with a file system format which is specified below. 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 in json format, 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 which complies with json format, if you are aware of it:

{"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 a 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":[{"type":"f","name":"foo","location":1234},{"type":"d","name":".","location":102},{"type":"d","name":"..","location":10},{"type":"f","name":"bar","location":2245}]}

Each directory entry in the filename\_to\_inode\_dict specifies three things. The type will indicate that it is a file “f’ directory “d” or special “s”. The name is the filename (or directory name), and the location is the block in which you can find the specification of that file (an inode).

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 refers to the number of times this directory entry is linked into the filesystem (when this is zero, the directory can be removed).

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, and mode) 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 (ie: [2445,2446,2447] is the array, in json form, of three 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.

The index block format and the free space list format will be a simple array 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). There will be nothing in the file other than the open and close square brackets, commas and numbers.