Prepared by Daniel Atkinson and Brandon Norman

We decided that each inode would contain 9 unsigned short ints that give the block index of a data block, 1 unsigned short int for the indirect data block index, and an unsigned long integer specifying how many bytes total the file occupies. This makes each inode occupy 28 bytes of storage, so 18 inodes can fit into a single block.

We also decided that we wanted to support up to 1008 files at a time. So for this to work, we needed 1008 inodes, and thus, we also needed 56 blocks to contain these inodes. We also needed 1008 directory blocks, since each file must also have a directory item.

Free inodes and free data blocks are tracked using bitmaps. Each inode has an index associated with it, and the position of a bit in the inode bitmap corresponds to the index of that inode. A 0 bit indicates the inode is free, and a 1 bit indicates the inode is already in use. Similarly, data blocks are tracked using the bits within the data block bitmap. Each bit position corresponds to a data block index, with 0 and 1 values for the bit indicating whether or not the data block is free. To check if a directory block is free, a function simply loops through all directory block indexes and finds the first directory where the "allocated" variable is set to 0.

The way we decided to partition our software disk was to:

1. Make the first 2 blocks of the software disk dedicated to 2 different bitmaps. One that keeps track of free inode blocks, and one that keeps track of free file data blocks.
2. Make the next 56 blocks dedicated to holding inodes.
3. Make the next 1008 blocks dedicated to holding a single directory each.
4. The rest of the 3891 blocks are dedicated to holding file data.

Some of the limitations of our filesystem include only allowing up to 128 characters for a file name, only supporting up to 1008 files, and the max size of a file being 135,680 bytes.