File System Description

(Bret Chalpin and Devon McDonald)

Key limits:

- Maximum number of files is 75
- Maximum filename size is 32 characters
- Maximum blocks per file is 56
- Maximum bytes per file is 28,672 (could be 28,672 if overflow on just one file)

We allocated 5 blocks in our filesystem to hold our File Allocation Table. Each block has 15 table entries (15 files). Each table entry consists of a 32-byte name and a 2-byte inode pointer. Each entry takes up 34 bytes, so the table in each block is up to 510 bytes (just under the 512-byte limit). Since there are 5 blocks used, our filesystem can have up to 75 files.

We have one block each for our two bitmaps. The first bitmap block is for the data. It keeps track of 3,975 data blocks. Since our block size is 512 bytes, there are up to 4,096 bits available. Therefore, we can keep track of all of our data blocks. Our second bitmap is used to keep track of the used and unused inodes (up to 75 since one inode per file created). For both bitmaps, when a bit is 1, the block has data in it. When a bit is 0, there is a free block. We simply traverse through the bitmap and find the next free block.

Our five inode blocks hold up to 15 inodes each. Each inode contains 12 direct block numbers, 1 indirect block number, and the filesize. Each number is 2 bytes, so each inode can be up to 28 bytes large (12*2 + 2 + 2 = 28). 15 inodes at 28 bytes each is 420 bytes - well under the maximum block size of 512 bytes.

We also allocated 75 blocks to be our indirect blocks (one block per file for up to 75 files). Given our constraint to use just one data bitmap and our 5,000-block limit, the indirect blocks can point to up to 43 data blocks each. Combining that with our 12 direct blocks and 1 indirect block, the maximum blocks per file is 56 blocks. This gives us 3,975 data blocks (75 files at 53 blocks), as mentioned earlier.

In total, we use 3,987 blocks out of a possible 5,000 blocks given in this assignment.