CSC 4103 Program 3
Design Document

Sam Dowd | Ben Graham

Physical Layout

The physical layout of the file system is split up in terms of blocks. The software disk consists of 5,000 blocks of size 512 bytes, numbered from 0 - 4999. Our file system only uses the first 4355. This is how our system utilizes those blocks:

Block 0: Free Space Management – File Memory

The first block is used for our free space management system. As mentioned, the block has 512 bytes, or 4096 bits. Our file system is implemented with 4096 blocks allocated to file memory, so this Is the perfect number of bits to represent each block of file memory. Each bit in this block is set to 0 if the corresponding file memory block is free, and set to 1 if the corresponding file memory is in use.

Blocks 1 – 128: File Name Directory

The next 128 blocks are used for the file name directory, used to associate file names with the associated inode. Each block is split into sections. The first section TODO

Blocks 129 – 256: Inode List

som

Block 257: Free Space Management – File Name Directory

This block is used to manage free and used space in the file name directory. As the file name directory portion is significantly smaller than the file memory portion of the disk, the entire block is not used. The structure is the same though. Each of the first 128 bits is set to 1 when the corresponding block of the file name directory is in use and 0 if it is free.

Block 258: Free Space Management – Inode List

This block does the same thing as block 257 but for the portion of the disk allocated to the list of inodes. If there is no inode in a block, the corresponding bit in this block is set to 0, otherwise it is set to 1.

Block 259 – 4354: File Memory

This is the meat of the filesystem, where all the data in the files is actually stored. When a new file is created, or more data is written to a file, the filesystem finds the first available block in this portion by consulting block 0. The filesystem then fills that block with data. If there is more remaining, the filesystem will repeat this process, finding the first available block, until all data is stored. There is no need for the data to be stored in sequential blocks as the index of each block is added to the inode of the file.

Implementation Design

Creating a File

After determining that no file with that name previously exists, the first step to creating a file is finding a place to put it. Our implementation uses the inode structure, so the first goal is finding open memory for the name directory entry and the inode itself. Our implementation starts by checking the free space management blocks to find the first free block allocated to the name directory. It then does the same for the inode list. Armed with two free blocks, now marked as in use, the system creates an entry in the file name directory with the given file name and the index for its inode. It then opens the file.

Opening a File

When a file is opened, a flag in the inode is set to true, indicating that the file is open. This prevents the same file from being opened twice and is also used in the inverse to error out when a closed file gets closed again. The information from the inode is stored in a File object that is used within the session to track information about the file.

Closing a File

When a file is closed, a flag in the inode is set to false, indicating that the file is closed. This tells the system it is okay to open the file again but it cannot currently be closed. This flag is checked against before opening or closing a file.

Deleting a File

Deleting a file really boils down to freeing the space the file was using. This is done in three steps: first by flipping all 1's in the memory free space management block to 0's where those blocks were linked by the file's inode, second by setting the bit in the inode list management block corresponding to the file's inode to 0, and third by setting the bit in the file directory management block corresponding to the file's inode to 0.

Reading a File

Reading a file is done by using the disk read to read each block of the file in the order it is presented in the inode.

Implementation Limits

Maximum File Name Size

The maximum file name size is 507 characters. This limitation is brought on because each entry in the file directory is limited to one block. A block has room for 512 characters, but the remaining 7 characters are used to store the mode information and the block number of the associated inode.

Maximum Number of Files

Because the inode list is 128 blocks in size, the maximum number of files our system can accommodate is 128.

Maximum Storage Space

Because our free space management system can only track 4096 blocks, the most data (not including metadata such as filenames) it can store is 2MB.

Maximum Size of File

Each inode has 10 direct pointers, 1 indirect pointer, and 1 double indirect pointer. Each pointer is a standard c int, stored in 4 bytes, and each block contains 512 bytes. Because of this, the maximum file size would normally be 10 * 512 + 128 * 512 + 128 * 128 * 512 or just shy of 8.5MB. But because our file system only has 2MB of storage space, the maximum file size is 2MB.