CSE 325 Lab Project 6

Andrew (AJ) Burns, William Rosenberger

Introduction:

For this project we made a simple version of a file system. For this, we designed, implemented and tested storing data in a virtual disk. This included implementing functions for create, open, read, write, seek, truncate, delete, and close.

Data Structures:

The following sections describe the data structures used to represent the virtual file system.

Storage:

For our file system, we stored each file in blocks of size 4096 bytes. These blocks are not necessarily contiguous. To deal with this, we made the first 4 bytes of the block metadata that is the index of the next block of the file— essentially, we created a singly linked list on the harddrive, rather than in main memory.

Since each file is just a list of blocks, we have a master table that stores the index of the first block of each file that is stored within our file system. This means when you “open” a file, it looks up that file in the table, and the returned descriptor points to the first block of that file. Since the first block is linked to the next block, and so on, it is easy to find all of the data of each file from the beginning.

Meta Block:

The first block in the virtual disk is reserved for storing key-value mappings between filenames and the block number of the first block in the file. This is used to find the start of files when they are opened.

File Block:

All blocks except the first one are available for data storage. The first 4 bytes of each block are reserved as a reference to the next block in the file. If these reserved bytes are “-100”, then the block has not been allocated to a file. If they are “0000”, then this block is the last block in the file. Otherwise, they store the block number for the next block in the file.

Open File descriptor table:

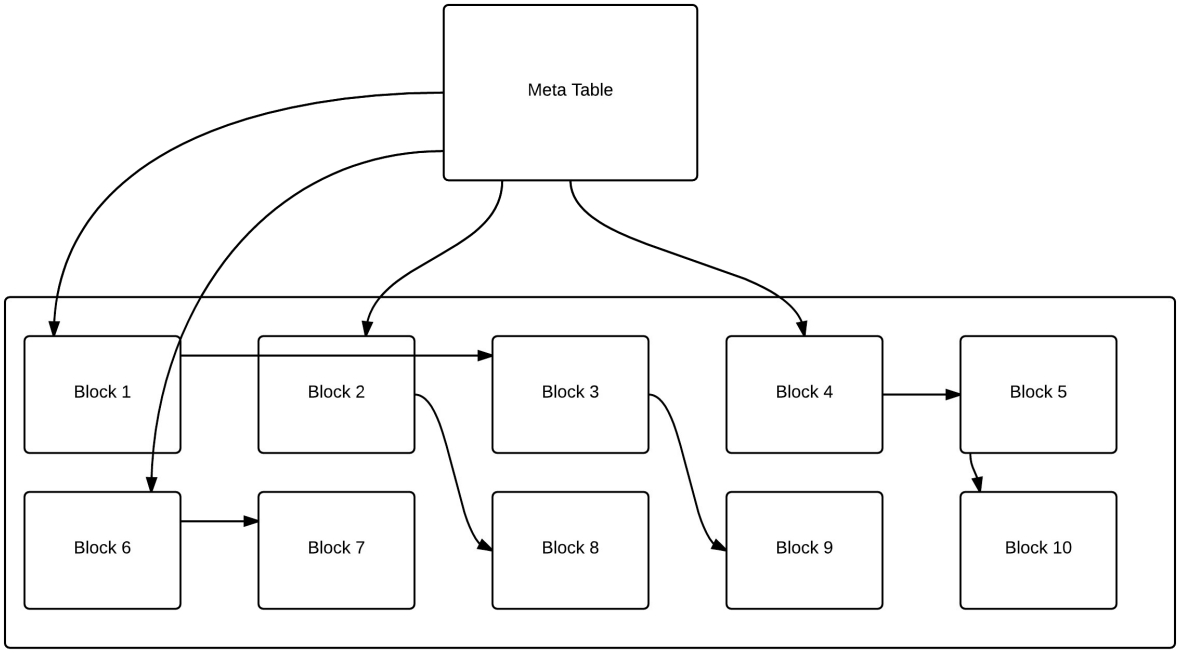
This structure contains a list of all open file descriptors, which includes the current cursor position of each of those files.

File Descriptor:

This is a structure that contains the name of the file, which block is currently being pointed to, and the offset within that block that the cursor is at. When a file is first opened, the first block of the file is being pointed to with no offset – that is, the

cursor is at position 0. As the user reads, writes, and seeks, the cursor will move through the blocks via the block number and the offset.

Diagrams and Explanations:



Above is a diagram of how our file system is setup. The meta table holds pointers to the beginning of each file, in this case block 1, 2, 4, and 6. The blocks then point to the next block of the file.

Creating a new file:

A new file starts with only one block allocated to it. This initial block is found by linearly searching for the first unused block. The reserved bytes in this block are set so that the block Is claimed (set to “0000”).

Opening/closing a file:

To open a file, our program finds the file in the list stored by the meta block, and follows the block pointer to the first block of the file. This is what the file descriptor returned contains. Using the Close operation frees the file descriptor and places a null in the open file descriptor table so the descriptor can no longer be used. It is still contained in the meta table, as the file is persistent.

Reading and writing:

The program uses the file descriptor to find where the cursor is pointing (this is done initially by using the meta table to find the first block of the file). For reading, it then reads all the data, following the next pointers as needed until the requested number of bytes are read. In the case of writing, it writes until the block is full, and if

needed finds another empty block to write to, pointing the first block to the new block in the process.

Seek
Seek is very simple, it changes the position of the cursor in the table of open file descriptors.

Deleting/truncating files:
This is simple, it removes the entry from the meta table, and walks down the list of blocks, setting the reserved bytes back to “-100”. When truncating, the meta block is not affected. During deletion, the key-value pair is deleted from the meta block itself.

Command Explanations:
Our tests are done by running the command “make run”, it runs our tests with the given file. You can also run the program with “./out filename” and replacing the filename with the name of the text file to be used.

Testing Plan and Results:
Plan:
Our plan was to do simple unit tests on each command being implemented. We checked the output of our tests manually instead of writing code to check, but thoroughly checked all cases we could come up with for each command. We began with commands like create, open, and close, moving onto write, read, seek and delete. We also tested the interaction of each command, like seek with read/write, opening and closing files with reading and writing, etc. We have placed detailed notes on the test cases in the main.c file.

Results:
Our results were very good. After much testing and debugging we were able to get most functions to pass our testing.

Known Bugs:
The primary problems with this project have to do with the initial design of the system. There are two primary issues

First, some of the initial functions we created assumed that the data being written to the files were strings. So, our program assumes that a byte with value 0 in the byte array means that there is nothing else to write. This means that using our file system to store anything but strings would be very difficult.

Second, in the initial design stage, we did not realize that it was necessary to store how much of each block contained actual data. This means that when we tried to write fs_get_filesize and fs_truncate, we could only provide block-size granularity. If we had stored how much of each block was actually being used, we would have been able to improve this to byte-size granularity.