Jay Patel – CIS 3207 – File System Documentation

This is how I plan to format the drive for my filesystem:

|  |  |  |
| --- | --- | --- |
| FAT | Root Directory | Data Blocks |

The File Allocation Table (FAT) will have an array of ints for each data block. The FAT will have (size of the drive/size of a block) entries and take up about (size of int \* (size of the drive/size of a block)) bytes in the drive. The first entry in the FAT will be initialized to -1 while the rest of the entries will be initialized to 0. The -1 entry in the FAT is there to indicate that it is the last block in the chain and a 0 in the FAT indicates that the block is not yet allocated. The block that is associated with the number x will be denoted as FAT[x]. If done this way the blocks can be made discontinuous in the drive. Also, because the root directory starts at block 0, we can assume that no block will ever link back to block 0.

Every block in the data region will contain data for a particular file or listing for a directory. If a block just contains a file, then the block will hold the data from that file. If the block has a directory listing, then the block will contain several *data* structs (one for each file) that will represent the contents of the directory. The name will refer to the file or directory name, while size will represent the size of the file. If it is a directory, then we will just set it to NULL or 0. Begin refers to the first block of the file/directory. Type will tell us if we are dealing with a file or a directory. Will just use a simple int value (1 for file and 2 for directory).

***Data struct:***

*typedef struct data {*

*char name[FILE\_LENGTH];*

*int size;*

*int begin;*

*int type;*

*} data;*

The block that begin points to will contain either a subdirectory or a file. The following struct will be used to represent a block.

***Block struct:***

*typedef struct block {*

*char file[SIZE\_OF\_BLOCK];*

*data dir[SIZE\_OF\_BLOCK];*

*} block;*

The following struct will be used to represent the drive.

***Drive struct:***

*typedef struct drive {*

*int FAT[NUM\_OF\_BLOCKS];*

*block info[NUM\_OF\_BLOCKS];*

*}drive;*

The following is a struct for a file pointer that will refer to a file that is already opened and that we will read and write to. Info will be a data \* that will represent the file’s metadata. The pointer and curr values will represent the current location of the file that we will be working with. The value for pointer will always be between 0 and SIZE\_OF\_BLOCK and will be an offset from the start curr.

***F\_pointer struct:***

*typedef struct f\_pointer {*

*data \* info;*

*int pointer;*

*int curr;*

*}f\_pointer;*

I will plan on memory mapping the drive into the program, so the program can see the see the drive as an entirely logical device. Also, the OS can handle low level file operations. The following code should map the drive to a drive struct that was defined earlier within the program. The PROT\_READ & PROT\_WRITE flags will allows us to read and write while the MAP\_SHARED flag will allow us to synch to drive in memory with the virtual drive. After all of this is done, we can just close it.

*int f = open("drive\_file\_name", O\_RDWR);*

*drive \*D = mmap(0, sizeof(drive), PROT\_READ | PROT\_WRITE, MAP\_SHARED, f, 0);*

*close(f);*

***Functions:***

Creates a new file or directory at a given location, returns nothing.

procedure create\_file(char \* filepath, int type, drive \* d) {

Check if the filepath is valid or already exists

Find the first empty space in the directory, and add a block to the directory if needed

Reserve a block for the new file

Write the new data into the directory

}

Returns a f\_pointer struct with that represents the requested file.

procedure open\_file(char \* filepath, drive \* d) {

Traverse the filepath to get the file’s metadata

Create the f\_pointer struct with the file’s metadata

Return the newly created struct

}

Writes the given data to a file, returns nothing.

procedure write\_file(f\_pointer \* file, char \* info, int len, drive \* d) {

Fill up the current block with the given info

Write additional blocks of data, reserving new blocks as needed

Update the current block and pointer of f\_pointer struct

}

Reads a given length of date from the specified file, returns nothing.

procedure read\_file(f\_pointer \* filename, int len, drive \* d) {

Copy the data up to the end of the block into a buffer

Copy the data from the next blocks into a buffer, until we reach the end of the file or hit the len limit

}

Closes the given file, returns nothing.

procedure close\_file(f\_pointer \* filename) {

frees the f\_pointer struct

}

Deletes a given file, returns nothing.

procedure delete\_file(char \* filename, drive \* d) {

Set all the blocks associated with this file in the FAT to 0

Remove the file’s metadata from the directory

}

***Testing:***

My current plans for testing are as follows:

1. Make sure we are able to create the file system.
   1. Check to see if the function call to mmap() was successful and that we were able to open the drive.
   2. Check to see if all of our structs were created successfully and contain the proper information.
2. Make sure that all of the filesystem functions work.
   1. Check to see if we can create a directory.
   2. Check to see if we can create a sub-directory.
   3. Check to see if we can create a file.
   4. Check to see if we can open a file.
   5. Check to see if we can read to a file.
   6. Check to see if we can write to a file.
   7. Check to see if we can delete a file.
   8. Check to see if we can close a file.
3. Test edge cases.
   1. Try to open a file that does not exist.
   2. Try to open an already open file.
   3. Interact with a file after we have already closed it.
   4. Interact with a file after we have deleted it.
   5. Try to delete a file that does not exist.
   6. Write to a closed file.
   7. Read from a closed file.
   8. Close an already closed file.
   9. Close a non-existent file.
   10. Interacting with a file in a sub-directory.
   11. etc…
4. Possible bugs/errors.
   1. I believe segmentation faults will be a big issue for me. The structs in my implementation uses a good number of arrays and it can be very easy to access something outside the specified area.
   2. Possibly creating files and storing them in the incorrect location in the file system.
   3. Not setting the proper values back to 0 in the FAT after we delete a file or directory.
   4. Having some issues interacting with files in sub-directories.