Edgardo Juan Gonzalez Design Document

The disk is laid out as follows

1. The boot
2. The fat
3. The copy of the fat
4. The root Directory
5. The data blocks reserved for files and other data

I handle allocating blocks by having a struct that keeps track of the number of free blocks and their indexes. By doing this I just have to find the first free block in the array so that I can allocate it. To free a data block I change the value to 1 to indicate that block belongs to a file. All blocks are initially freed so their value begins at -2. To free them, I have a 2d array that mimics the virtual disk (array[DATA_BLOCKS][BLOCK_SIZE]). I iterate through the block I want to free and zero out its value. I then update the array dedicated to keeping track of which blocks are free. The file descriptor associated with a file is the index of the file descriptor in the file descriptor table. From there one can use that to get the index of the file in the root directory. This is how I make the connection between the file descriptor and the actual file. The physical Space is the actual permanent file that mimics a disk while my 2d array is the logical space.

int make_fs(char *disk_name);
This function makes the disk, opens the disk, calls the initializer functions for boot, fat, directory, and fileSpace, and it writes all those to disk before closing the disk. It returns 0 if successful and -1 if it fails.

int mount_fs(char *disk_name);
This function opens the disk, reads all the data that was written in make and stores the values. It returns 0 if successful and -1 if it fails.

int umount_fs(char *disk_name);
This function writes the changes made to the boot, fat, directory, and fileSpace to the disk before closing the disk. It returns 0 if successful and -1 if it fails.

int fs_open(char *name);
This function opens a file by creating and initializing a free file descriptor and initializing some file data. It returns 0 if successful and -1 if it fails.

int fs_close(int fildes);
This function closes a file by freeing the file descriptor and resetting the file descriptor's information. It returns 0 if successful and -1 if it fails.

int fs_create(char *name);
This function creates a file by initializing some file data and assigning it a free block. It returns 0 if successful and -1 if it fails.

int fs_delete(char *name);
This function deletes a file by deleting all the file information in the fileSpace and in the rootDirectory. It returns 0 if successful and -1 if it fails.
int fs_read(int fildes, void *buf, size_t nbyte);
This function reads from a file based on the current position of the offset and stores those values in the parameter buf. It returns the number of bytes read if successful and -1 if it fails.

int fs_write(int fildes, void *buf, size_t nbyte);
This function writes the information in buf to a file based on the current position of the offset and it may or may not append the file. It returns the number of bytes written to the file if successful and -1 if it fails.

int fs_get_filesize(int fildes);
This function returns the size of the file pointed at by the file descriptor parameter.

int fs_lseek(int fildes, off_t offset);
This function sets the offset of the file pointed at by the file descriptor parameter to the parameter off_t offset. It returns 0 if successful and -1 if it fails.

int fs_truncate(int fildes, off_t length);
This function truncates the file to the length specified by the parameter off_t length. It deletes the data lost due to the truncation while also freeing any blocks no longer being used before returning 0 if successful and -1 if it fails.

void initialize_boot();
This function initializes the members of the boot struct.

void initialize_fat();
This function initializes the members of the boot fat struct and its copy.

void initialize_rootDir();
This function initializes the members of the boot Directory struct (called rootDir).

void initialize_file_space();
This function initializes the members of the fileSpace struct.

void initialize_data();
This function initializes the members of the data struct.

void initialize_fdt();
This function initializes the members of the fdt (file descriptor table) struct.

int getFileIndex(char *name);

This function returns the index of the file or -1 if it could not find the file.

int getFirstFreeFileDescriptor();
This function returns the index of the first free file descriptor or -1 if it fails to find one.

int getFirstFreeRootDirectoryIndex();
This function returns the first free index in the rootDirectory or -1 if it fails.

int getFirstFreeBlock();
This function returns the index of the first free block or -1 if it fails.

int getSizeOfPathOfBlocks(int fildes);
This returns the amount of blocks currently needed by the file or -1 if it fails.

int getFilePath(int fildes, short *array);
This function stores the indexes of the block path of the specific file or -1 if it fails.

void zero_out_data_block(int index);
This function zero outs-an entire block.

int findOffsetInFileSpace(int currentBlock, int offset, int *array);
This function takes the offset within a file and matches it to the location in the 2d array. It stores the block and offset within the block in the array parameter. Returns 0 (successful) or -1 (failure)

int getDifferenceBetweenBlockSizes(int fileSize, int newSize, int offset, int numberOfBytesBeforeOperation, int functionFlag);
This function determines the difference in block size between the fileSize before and after either truncation or writing. The functionFlag dictates which operation is calling it. It returns the difference.