Jordan Lu | jjl4mb

CS 4414 - Operating Systems

11/23/2016

Project 7: File System

The first step in completing the project was the create the make_fs() and mount_fs() functions write and read the super block, directory, and FAT blocks to and from the virtual disk. I have a global super block variable that updates every time a new disk is mounted.The super blocks contains information as to where the directory blocks, FAT blocks, and data blocks are stored. I then initialize everything else on the disk with new char[16], or empty blocks. I also make an array for FAT entries and for directory entries, both of which are custom structs. The fat entry struct contains the status of the block, as well as next FAT block of the file. The directory, on the other hand, contains the file name, the status of the file, the size of file, and the first FAT block the file uses. When mount_fs() is called, I load information from the disk, into these data structures, based on the location of their respective blocks. In mount_fs(), we must also set up the file descriptors that will be used to read and write open files. Each file descriptor is a struct object containing a status int, the file it is associated with, and the offset for that given file.

The next step was to implement all of the functions listed in the assignment. These included fs_close(), fs_create(), fs_delete(), and more. The first function, fs_open() essentially opens the file for reading and writing. Essentially, we look through our file descriptors, and using first-fit algorithm, pick a free file descriptor. If no file descriptors are open, then the function will return -1. Otherwise, we set the file descriptor to used, set the file equal to the file we opened, and the offset to 0, or where the file will begin to read / write. Next, we wrote close, which will

essentially check if a file is open, and if so, close it. We set the file descriptor to unused, and the file to 0.

For the functions create and delete, we needed to also update our directory and FAT table. When creating a file, we first use first fit algorithm to choose the starting directory and fat index. If both of these return valid indices, we update the directory entry to contain the file name, fat index, file size (0 when created) and set the status to 1. We next set the next fat index to be EOF, and the fat block status to used. When we delete the file, we can first update the directory, in which we simply set file size to 0 and the status to free. However, we also have to update our fat table, and in order to do so, propagate through every fat entry, and the fat block that they point to. For every fat block we get to, we set the status to free. Since we update the EOF when we create more data blocks for a certain file, we can repeat until we hit the EOF.

fs_get_filesize(), fs_lseek(), and fs_truncate() were a bit easier to write. Since we store the file size for the directory entry, we can simply go to file the file descriptor is pointing at, and return the file size. For lseek, we have to check whether or not the offset input is positive or negative. If the offset is positive, we can simply set the offset in our file descriptor to the given offset. Otherwise, we must add the offset, as it should subtract in the case of a negative offset. The truncate function is implemented by simply updating the status in the fat blocks. Since we save the size of the file in our directory, we can just set the status in the fat block to free. Anything that uses that fat and data block will just overwrite whatever was in there. This makes truncate much simpler.

Finally, we implemented read and write. Both of them follow a similar method, in which we much traverse through the fat blocks. For read, we can just go to the offset specified by the

file descriptor. We first traverse to the block of the offset, and them move to the data locations. From there, we read block by block using block_read until we reach the amount of bytes we are reading, or the end of the file. To write we, traverse to the offset specified by the file descriptors. We use block_write to write our data to the disk. When the block we are writing to runs out of space, we again, use first fit to find a fat block that is not used. We then update our fat table, and continue to write until we have written the correct amount.