OPERATING SYSTEMS

ASSIGNMENT 1 REPORT

Fatima Tariq ft07200

Structures:

Apart from the given structures, I implemented two new: Datablock and Superblock.

Datablock: This contains the data in each block and size is 1024 bytes (1 kb).

Superblock: This represents the superblock of the file system. I have added objects of all the other structures in this. There's total blocks which is 127 (not including superblock), a list that tells us which blocks are free, an array of 16 inodes (inode structure), and 127 datablocks. Additionally, there is also directories (dirent structure) which keeps track of directories made.

Implementation:

Coming towards implementation, it's mostly an inode-based implementation and I have used block pointers to manage directories and files. All traversals are also done using inode tables and block pointers.

I have used fwrite and fread to write and read from myfs. Initially, since myfs does not exist, it will create the file and initialize it (call initialize function). Here, the inode table and free blocks list are getting initialized. The root directory is also initialized in this function (in the inode table and as a directory entry in dirent directories). If the file myfs already exists, it's simply loading data by reading it. After every command, I have also saved the current state by calling the save file system function.

In myfs, I mainly have the superblock structure since that contains everything.

The structure of the file system is as described in the hw file. However, for data blocks, if we create a file that's occupying more than one block (let's say the file's inode is 4 so block ptrs [4,5,6,7,8,-1,-1,-1,-1]), then the parent directory will just point to block 4.

I have separate functions for checking if directory exists and for updating sizes. In the updating sizes function, it will update the size of every parent directory and the root directory.

FUNCTIONS:

Create directory:

This function begins by tokenizing the input path to get each directory in the path. It also checks whether each directory in the path exists. It finds a free inode in the inode table to allocate to the new directory. If the parent directory exists, it updates its block pointers to include the new directory. The function then updates the inode table. It also provides error messages for cases where the directory already exists or no inodes are available. Ultimately, the function saves the updated filesystem.

Create File:

This function receives the path, file size, superblocmk, and a flag which is 1 if it is a move or copy operation. It begins by extracting the filename and parent directory from the path and tokenizes the path to ensure the parent directory's existence. If it's not a move operation, the function checks for an existing file with the same name. it allocates a free inode and the required blocks for the file based on the specified size. The parent directory's block pointers are updated to include the new file, and the allocated blocks of the file are filled with random characters to simulate file content. Then I have updated the inode table. It calls updateDirectorySizes to update directory sizes based on the new file. Error handling conditions are also included in the function.

Delete file:

This function extracts the filename and parent dir from the given path. It then checks if the parent directory exists through path tokenization. Then, it locates the file's inode index by searching for the filename. If the file exists, the function updates the parent directory's block pointers (sets them to -1), deallocates the file's blocks (-1), updates directory sizes through updateDirectorySizes, and marks the inode as unused. Error handling conditions are also included in case of incorrect paths, non-existent directories or files.

Move file:

This function first tokenizes to get source and dest filenames and paths. It then checks for the existence of source and destination dirs through the directoryExists function. The source file's inode index is obtained by traversing the inode table. After checking that the file exists, the function sets the destination file's size to match that of the source file. The file is then moved by calling deleteFile and createFile functions to delete the source file and create a duplicate at the destination.

Copy file:

This function tokenizes the source and dest paths to extract the filenames and paths. It then checks for the existence of the source and dest directories through the directoryExists function. The function then determines the inode index of the source file by traversing the inode table based on the extracted src filename. It sets the size of the destination file equal to the size of the source file and calls the createFile function to generate a duplicate file at the destination path.

Remove directory:

The function extracts the directory path and name through tokenizing. It then checks for the existence of the directories in the path through the directoryExists function. It then identifies the inode index of the target directory for deletion through traversal of inode table and checks if directory exists. Then the function starts a recursive process to delete all contents within the directory, whether they are subdirectories or files. This recursive process involves traversing the block pointers of the directory, recursively deleting subdirectories, and deleting files. Subsequently, it updates the parent directory's block pointers and size, along with the root directory's size. After this, it updates the inode table, block pointers, and the free block list. It marks the inode of the directory as unused,

I faced issues in this function. I have used a new C function sprintf(). It formats and stores a series of characters and values in the array buffer instead of printing it out like printf(). Let's consider /home/user/user2 and I want to delete /home/user (so that's the path that goes to removedirectory(). Using inode indices, I get the name of the subdirectory (user2) and using sprintf() we get the path /home/user/user2 which then goes to the remove function again through recursion.

Testing:

For testing, I used different paths with different sizes to check if it's working. In the end, I'm using it with the sampleinput file by parsing commands.

Memory:

I have dynamically allocated and freed memory of the data blocks used through malloc and free.

The disc is 128 KB in size and is divided into 127 data blocks and 1 superblock. My implementation follows the rule that only 16 files/directories may be supported. There are 16 inodes in the inode table, each of which represents a file or directory. It also enforces a cap of 8 blocks per file. The maximum length for file and directory names is eight characters, including the necessary NULL character.

For inode to be 56 bytes, although I have not dynamically allocated 56 bytes, the structure has 4 integers, 1 integer array of size 8, and name with max length 8. So [(4x4)+(4x8)+8]=56 bytes.

Problems faced:

A day before the assignment was due, my Ubuntu stopped working. I tried to install it multiple times on my laptop but it crashed every time. I even uninstalled and installed again but it kept crashing despite my laptop having enough storage. I primarily did all the coding and testing on vs code and it was working fine there. In the end, I confirmed from the RA that it's okay if it's working on vs code. In case it gives some issue on Ubuntu, kindly try the code on vs code.