Assignment 1 - File System Implementation Report

Muhammad Youshay

September 30, 2023

1 Data Structures

The main data structures used are:

- **Inode**: This contains metadata about each file and directory like name, size, pointers to data blocks etc. This was provided to us in the tempelate and no changes have been made to it.
- **Dirent**: This represents a directory entry, containing filename, inode number etc.

These are the only two data structures that I have used throughout this implementation, however, we can also consider our myfs as a data structure as well because it itself is manipulated using file descriptors and standard file I/O functions. Data is written and read directly from this file to simulate the file system structure. However, I'll discuss how it works in the next section.

2 Algorithms/Functions

Algorithm 1 initialize_fileSystem()

```
1: Opens file myfs with permssion to read and write.
2: Initializes with null chars
3: Writes 'S' as Superblock marker to 0th block
4: Creates root Inode structure for root - '/'.
5: Creates '.' entry for root directory in data block
```

Algorithm 2 GetParentInode() - Helper

```
1: Tokenizes the path
2: Finds the parent inode using lseek and read and matches the enteries
3: Returns inode number of parent directory
```

Algorithm 3 GetFreeInode() - Helper

```
1: Reads inode table
2: Searches for inode with used = 0
3: Returns first unused inode number
```

Algorithm 4 GetFreeBlocks() - Helper

```
1: Searches data blocks
2: Finds unused data blocks
3: Returns array of free blocks
```

Algorithm 5 CD() - Create Directory

```
1: Gets parent inode using GetParentInode() - helper defined above in the doc
2: Checks directory doesn't exist already
3: Calls GetFreeInode(), GetFreeBlocks()
4: Initializes new directory Inode
5: Copies Inode to table, adds entry to parent
6: Copies '.' and '..' entries to directory
```

Algorithm 6 CR() - Create File

```
1: Similar logic as CD to allocate new inode and blocks
2: Initializes inode as file, copies to inode table
3: Updates parent directory with new entry
```

Algorithm 7 DL() - Delete File

```
1: Frees inode and blocks of file
2: Removes file entry from parent directory
```

Algorithm 8 DeleteDirectoryHelper() - Helper

```
1: Recursively deletes all directories and files inside this directory by calling
   itself when sub-dirs are found
2: Frees Inode and data blocks for each file/dir
3: Removes entries from directory when done deleting everything inside
```

Algorithm 9 DD() - Delete Directory

```
1: Calls DeleteDirectoryHelper()
2: Removes entry from parent directory
3: Frees Inode and blocks of directory passed to it
```

Algorithm 10 CP() - Copy File

```
1: Reads src Inode, copies blocks
2: Allocates free Inode and blocks for dest
3: Copies src data blocks to dest
4: Adds new dest entry to parent
```

Algorithm 11 MV() - Move File

```
1: Deletes src entry after copying inode
2: Adds that Inode to new dest entry
```

Algorithm 12 LL() - List Files

```
1: Iterates over all Inodes
2: Prints files and dirs that are in use
```

3 Conclusion and Approach

My approach was simple and broken down into steps as in most of the functions, we had to do the same things and then change them accordingly. Everything in my code revolves around the file system initialization. Which I would say is the main data structure of my code. It contains the superblock, inode blocks, data blocks and directory entries. In all the functions I directly manipulated this disk using read and write system calls and lseek (Had to research alot on the internet using how this works - chat gpt also helped a little here). As I mentioned before, many things were the same in all functions, like allocating free inodes and data blocks, Initializing those inodes, updating sizes and freeing inodes when deleting. One of the major problems that I had was when deleting directories. This was because initially, I was using a linear approach because of which I was having problems with sub directories. To solve this, I created a helper function that used recursive approach to delete it (details are mentioned in section 2 above). My main function reads the input file which was provided to us line by line and executes the commands accordingly. However, I have modified it a bit to test the MV command and have also added LL commands after each command to test my code. Updated file is available in the zip file.