ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 312E
Computer Operating Systems

Homework 3

Ali Emre Kaya
150210097
kayaemr21@itu.edu.tr

SPRING 2025

Contents

1 Introduction

In this assignment, the task was to implement a simple user-space file system. In order to implement a system that would meet the user's requests in accordance with the given disk layout, we had to prepare an empty disk, divide it into the necessary partitions, and run the appropriate operations within the system.

2 Questions

2.1 How are directory entries stored/searched?

I use mkdir while storing DirectoryEntry. First I parse the path and then I try to go to the parent node starting from root. What I need while doing this is the root's direct blocks, I need to pull the DirectoryEntrys from these direct blocks, determine the folder below according to their names and move to that folder and repeat this until I reach the parent folder. After reaching the parent folder in this way, I can create my new folder. When creating my new folder, I need to allocate an empty space from the blocks of the Parent folder and place my new DirectoryEntry in this space.

The logic of searching is actually similar. Again, first I need to reach the parent directory, and after reaching this directory, I need to find the folder I am actually looking for among the blocks of the parent directory. For example, I need to reach this directory in the rmdir function and I follow such a path.

2.2 What happens when the disk or inode table is full?

When the disk is full, new files cannot be created because the new block to be added cannot be found (it remains as -1 in the code, meaning that the code stops working and gives an error). Commands such as create_fs, mkdir_fs, write_fs cannot do what they are supposed to do and are run in vain.

Similarly, if no free inode is found in the system (checked with new_inode_index in the code), the system returns an error stating that no free inode can be found and the operation performed is thrown away. There is a total of 288 inode space on the disk and I calculated this as follows.

struct Inode = 32 bytes
block size = 1024 byte
inodes per block → 1024 / 32 = 32
inode blocks = 9
total inodes = 32 * 9 = 288

2.3 How is double allocation prevented?

Double allocation is protected by the bitmap structure. The bitmap block contains one bit for each block, 1 if the block is in use, 0 if not. In each block write operation, this data is directly marked and written to the inode, it works as atomically as possible, after this marking, the block write operations are performed. In this way, double allocation is prevented in a more secure way.

2.4 Describe one encountered error and your solution.

While trying to complete the ls_fs function, I had already prepared the find_parent_directory function, so that it would be easier and I would not have to write extra code. But the ls_fs function, by its nature, wants the files and directories directly under the written path. After running the find_parent_directory function, I thought that I had found the path I wanted correctly by getting the inode of the parent and the inode of the directory below. But then I realized that if the user ls_fs wanted root (files under root), since root did not have a parent, my function would give an error. I solved this error by opening an if else condition where I evaluated root specifically.

3 Commands

3.1 mkfs

- create a file with the given name and img extension.
- initialize all blocks to 0.
- initialize superblock.
- initialize bitmap block.
- initialize inode blocks.
- create root inode.

3.2 mkdir_fs

- check path and disk, and get superblock.
- find root inode, and find parent directory.
- check if folder already exists.
- allocate an inode for new folder. allocate data space (block) for new folder.
- write inode and new entries.
- add new directory to parent directory as new entry. find a free space to add new directory.

3.3 create_fs

It works very similarly to mkdir_fs, the difference being the is_directory in Inode and DirectoryEntry class arrangements. If it's a directory, DirectoryEntry class initalized. If it's a file, a block will be initalzied to the 0s.

3.4 ls_fs

- take the path, take superblock, take root inode.
- find target directory's inode.
- traverse and read all entries in this inode.

3.5 write_fs

- read the superblock and parse the path
- go to the parent directory containing the file, search for the file in the parent directory
- verify it's a file (not a directory) and read its inode
- calculate number of blocks needed for the data. free existing blocks if needed. allocate new blocks if necessary
- update the bitmap marking blocks as used
- write the data content to blocks. update the file's size and inode information

3.6 read_fs

- open disk, take supernode, parse path, find root inode
- go to the parent directory, and look for file
- check file and read it with a buffer

3.7 delete_fs

- open disk, read superblock, parse path
- go to the parent directory containing the target item
- search for the target item in the parent directory
- read the bitmap to track block usage
- free all blocks used by the target item
- update the bitmap to mark blocks as free
- mark the target's inode as invalid
- remove the target's entry from the parent directory
- write all changes back to disk

3.8 rmdir_fs

- open disk, read superblock, parse path
- read the root inode and navigate to the parent directory of the target. search for the target directory in the parent directory's entries
- read the target directory's inode and verify it's a directory, not a file
- check if the directory is empty by examining all its data blocks. if not empty, refuse to delete and return an error. if empty, read the bitmap to track block usage
- mark all blocks used by the directory as free in the bitmap
- remove the directory entry from the parent directory
- write all changes back to disk

4 Confusions

There was no example given on how to do the check. So I created my own outputs as appropriate (for example, create_fs does not return any output, I would like to return an output like 'File added successfully'). It would be great if an example command and an example expected output were given to solve this problem.

REFERENCES