CSE312 Homework3 Part1 Report

Yeşim Yalçın-200104004094

File System Structure

The filesystem is in a form which is like a tree. This tree starts from the root directory. All other directories are connected to this root directory so any directory can and should be reached from here.

The filesystem structure with the sizes of each part can be seen below. The sizes are calculated according to the given arguments. **My program in part2 needs 3 arguments. Example: makeFileSystem 4 400 mySystem.dat**

- The block size
- The I-Node Count
- The filesystem dat file name

Superblock 14B+16KB	**I-Nodes** INodeCount*55B	**Blocks** (16MB-14B-INodeCount*55B-16KB)/BlockSize

16MB File System

Example: If blocksize is given as 4KB and INodeCount is given as 400:

Superblock 14B+16KB	**I-Nodes** 400*55B=22KB	**Blocks** (16MB-14B-22KB-16KB)/4KB=4086

16MB File System

Note that:

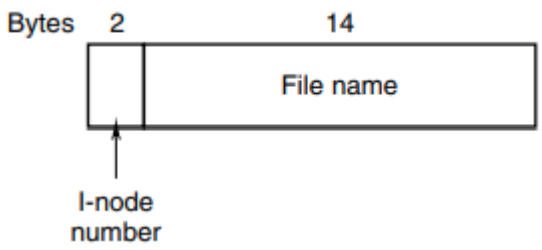
- The blocksize should not be smaller than 1KB and shoud not be bigger than 16MB.
- The iNodeCount must be at least 2. If it is 0 then there cannot be a root. If it is 1 then there can be only a root which would be pointless.
- When the sizes of iNodes and blocks are calculated, it should not be bigger than 16MB.

I will be explaning the file system structure in 5 categories.

1-) Directory Structure

A directory entry contains only 2 fields. These entries are:

- The file name
- I-Node Number connected to the file



The file name must be at most 13 characters. These characters can be anything ASCII except the character "/". Because "/" is used to represent the directory hierarchy. **Do not forget that I use "/" for root and regular separations like UNIX not "\".**

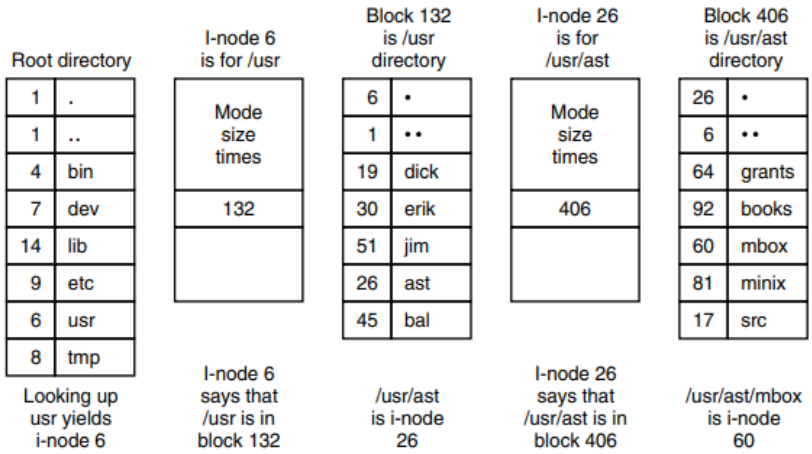
```
struct file{
    uint16_t iNodeNo;
    char fileName[14];
};
```

Because the file name has a fixed size, it is 14 bytes. This might lead to storage wasting for short named files however, it reduces the complexity of the implementation.

Total Size of a File= 16B
The I-Node number is 2 bytes. It is just an integer to find the I-Node of the file. I-Node of a file includes the attributes and data blocks of a file. It will be explained in the "I-Node Structure" part of the report.
By looking at this directory, one can easily look up for a file name and find it's I-Node number. After getting the I-Node number, one can reach the I-Node of the file from I-Nodes list and do necessary operations on that file.

There is a different directory structure for each directory. For example if we are trying to reach a file named mbox which is in "/usr/ast" directory we need to do these steps:

- Find usr in "root" directory structure.
- With the i-node number of usr, reach the i-node and get the block address which has the directory structure.
- Find ast in "usr" directory structure
- With the i-node number of ast, reach the i-node and get the block address which has the directory structure.
- Find mbox in "asr" directory structure.



2-) Free Blocks & Free I-Nodes

The data inside files will be held in blocks. So whenever a new data enters a file, a block should be assigned to it and the data should be written onto it. For that matter, we need to keep track of the free blocks.
I will be using a bitmap kind of approach. I will have n bits for n blocks. If a block is free, it's value in the map will be 1. If the block is not free, it's value in the map will be 0.
Using this method is very efficient because setting a block as 1 or 0 whenever it's state is changed takes O(1) time. However, this method requires exactly n bytes allocation which could have been less with different approaches.

Like blocks, whenever a new file is created it needs a unique I-Node and I-Node number. For that matter, I keep a isFree variable inside I-Nodes. This is 1 if the I-Node is free. It is 0 if it is full. I could have used another bitmap like the one I used for free blocks for free Node numbers as well. However, I thought that my current solution is more simple. Because, it uses the same amount of space as a bitmap aproach but we do not need to store another address for the bitmap.

3-) Superblock Structure

Superblock is a data holder which includes general information about the filesystem. Here, I will hold information related to:

- Block size
- Number of blocks
- Block positions
- Free block map position
- Number of I-Node Numbers
- I-Node positions

```
#define KB_1 (1024)
#define MB_1 (1024*KB_1)
#define FREEMAPSIZE 16*MB_1/KB_1

struct superblock{
    uint16_t blockSize;     //will be X KB
    uint16_t blockCount;
    uint32_t firstBlockAdd;
    uint8_t freeBlocks[FREEMAPSIZE];
    uint16_t iNodeCount;
    uint32_t firstINodeAdd; //Root
};
```

Total Size of Superblock= 14B+16KB

-The block size will be given as a command line argument. It will be X KB.
-The number of blocks depends on the size of the file system and the block size. The max size of the file system will be 16 MB. Therefore the block count will be calculated according to this max size. It will be useful to iterate over blocks and free blocks map. Example calculation of it is shown in the first section of the report.
-The address of the first block will be held to reach block positions. This address will be the exact byte that the first block is stored in the filesystem. This byte will be calculated thanks to the given I-Node count and the block sizes. The blocks will be kept in an array like structure.
-The free block map will be held inside the superblock. It's size has to be predefined because it is in a struct. This size is calculated assuming the minimum block size can be 1KB and all 16MB can be used for free blocks. The structure of this map is explained in "Free Blocks & Free I-Nodes" part of the report.

-The number of total I-Node numbers will be held. It will be used to iterate over I-Nodes and free I-Node numbers map.
-The address of the first I-Node will be held to reach I-Nodes. This address will be the exact byte that the first I-Node is stored in the filesystem. This byte will be calculated thanks to the given I-Node count and the block sizes. The I-Nodes will be kept in an array like structure. The first I-Node will belong to the root.

4-) I-Node Structure

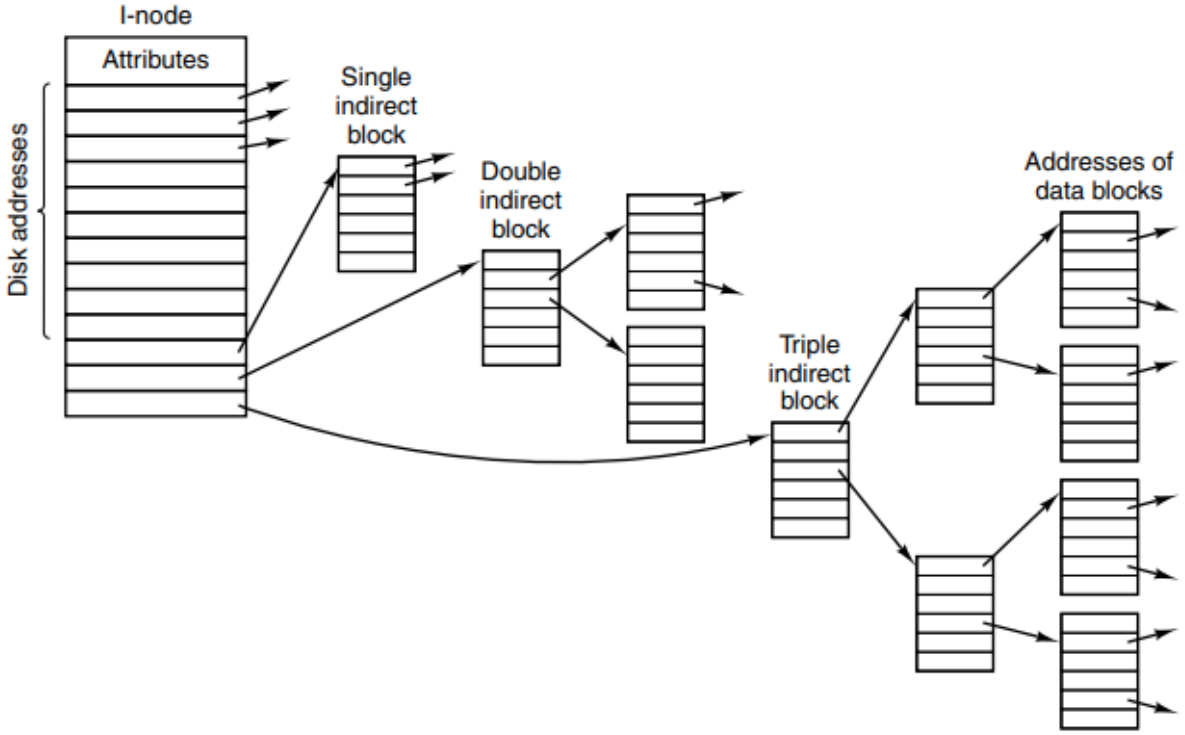
I-Nodes will be used to store attributes and the addresses of data blocks of files.

```
struct iNode{
    char name[14];
    uint32_t size;
    time_t creationDate;
    time_t modificationDate;
    uint8_t isFree;
    uint16_t directAdd[7];
    uint16_t singleAdd[1];
    uint16_t doubleAdd[1];
    uint16_t tripleAdd[1];
};
```

The attributes will include:

- Name of the file
- Size of the file
- File creation date
- Last modification date
- isFree to identify if the iNode is free

Total Size of an I-Node=55B



An I-Node will be able to store up to 10 addresses. First 7 of them will be direct block addresses. 8th one will link to an indirect node. This node will contain 100 block addresses. 9th one will link to another indirect node. However, this time this node includes 100 other indirect node links. These ones will have 100 block addresses etc.

To sum up:

- A single indirect link can link to 100 blocks
- A double indirect link can link to 10000 blocks
- A triple indirect link can link to 1000000 blocks

These indirectly linked blocks will be stored as regular free blocks. The reason why I chose 100 as the address count a block can store is not to waste many free blocks.

For example if a file needs 1000 blocks:

- 7 of them will be stored as direct links.
- 100 of them will be stored in the single indirect link = Meaning 1 free block used for links
- 893 of them will be stored in the double indirect link. = Meaning 1 block used to store 100 single indirect links + 9 block blocks used to store 893 links.
- In total for a file that needs 1000 blocks, only 11 blocks are needed for the indirect linking process. 1000 blocks for a file will not happen a lot considering our filesystem size. Moreover, if the 16MB limit is reached during this process, the filesystem can simply tell the user that the limit is reached. No errors will be caused.

5-) File System Operations

a-) void **printDirectoryInfo**(struct superblock* s, struct iNode* iN, char* directoryPath);
This function is used for "dir" command. It prints out all the information of a directory.
If **the given path is not valid or if it does not belong to a directory**, an error message is printed out.

b-) void **createDirectory**(char* directoryPath, struct superblock* s, struct iNode* iN);
This function is used for "mkdir" command. It creates a directory in the given path with the given name inside the path.
If **the path is not valid, a directory/file with the same name already exists in the given path or the directory name is too long** then an error message is printed.

c-) void **removeDirectory**(char* directoryPath, struct superblock* s, struct iNode* iN);
This function is used for "rmdir" command. It deletes the directory given in the path.
If **the path is not valid, does not belong to a directory or the directory is not empty** then an error message is printed.

d-) void **writeToFile**(char* path, char* fileName, struct superblock* s, struct iNode* iN);
This function is used for "write" command. It firstly creates a file in the given path then writes the content of a file with the given fileName to the created file.
If **the path is not valid, the given fileName does not belong to any file, there is already a file with the given path, there is not enough space to for the file or the new file name is too long** then an error message is given.

e-) void **readFromFile**(char* path, char* fileName, struct superblock* s, struct iNode* iN);
This function is used for "read" command. It reads the content of the file which is in the given path then writes the content to the file with the given fileName.
If **the path is not valid, the path does not belong to a file or the file with the given fileName can't be opened** then an error message is shown.

f-) void **removeFile**(char* path, struct superblock* s, struct iNode* iN);
This function is used for "del" command. It deletes the file which is in the given path.
If **the path is not valid or the given path does not belong to a file** then an error message is shown.

6-) Other Important Funtions

a-) int **checkPathValidity**(int size1, int size2, char array[size1][size2], int count, struct superblock* s, struct iNode* iN);
This is used in almost all command operations. It checks if the path given in an array format is valid also returns the iNode of the directory/file in the path.

b-) int **findParentINode**(int size1, int size2, char array[size1][size2], int count, int numberInArray, struct superblock* s, struct iNode* iN);
This function is used in many places. It retrieves any directory/file's parent's inode. A path is given in an array format.

c-) void **removeDirFromDir**(struct iNode* parent, int deletingInode, struct superblock* s);

This function is used to remove the file entry of a file/directory from it's parent. Moreover, if the file which will be removed is somewhere in the middle of a block then it finds the last file entry of the directory and puts it in the new free space. This is good to prevent fragmentation.

d-) int **getAvailableBlock**(struct iNode* iN, struct superblock* s);
This is used to get the first available space in a block which belongs to an iNode. It also assigns new blocks if all of the previous blocks of the iNode is full. **It gives an error message if there is no available blocks left.**

Tests

Testing for Exceptions:

```
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper dir "/a/SecondA/D/file2"
The command line arguments are invalid. You have entered 2 arguments. First one should be the system file. Second one should be dumpe2fs.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat dir "/a/SecondA/D/file2"
This is not a directory
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dar mkdir "/a"
Cannot open the system file: No such file or directory
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat mkdir "/a"
Given directory already exists.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat mkdir "/a/d/BB"
The given path does not exist.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat mkdir "/sdfsgsdghdfghdfghjdfgjdfgjdfghfdghfdgh"
Given filename is too long
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat rmdir "/x"
The given directory does not exist.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat rmdir "/a/SecondA/D/file2"
This is not a directory
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat rmdir "/a"
The directory is not empty.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat write "/a/d/hello" data.txt
The given path does not exist.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat write "/a" data.txt
Given file already exists.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat write "/a/newFile" sdfdsfgsd
Failed to open the file: No such file or directory
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat read "/a" result
This is not a file.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat read "/a/d/file" result
The given file does not exist.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat del "/a"
This is not a file
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat del "/a/d/file"
The given file does not exist.
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSyetmOper data.dat rmdir "/"
-bash: ./fileSyetmOper: No such file or directory
jade@DESKTOP-423HS14:/mnt/d/Yeni klasör/GTÜ/Year 3/Term 2/OS/hw3$ ./fileSystemOper data.dat rmdir "/"
You cannot delete root directory.
```

Creating Directories:

```
cse312@ubuntu:~/Music$ make run
./makeFileSystem 4 400 data.dat
cse312@ubuntu:~/Music$ make run1
./fileSystemOper data.dat mkdir "/a"
cse312@ubuntu:~/Music$ make run2
./fileSystemOper data.dat mkdir "/b"
cse312@ubuntu:~/Music$ make run3
./fileSystemOper data.dat mkdir "/c"
```

```
*****Information of the File System*****
Block Related Information
Total Block Count: 4087
Block Size: 4KB
Total Block Space: 15MB

Other Space Related Information
Total Superblock Space: 16KB
Total INode Count: 400
INode Size: 47B
Total INode Space: 18KB
Total Size: 16MB

File Related Information
Total Free Block Count: 4083
Total Free Inode Count: 396
Total Directory Count: 4
Total File Count: 0

Directory: "/", INode: 0
Parent Directory: ".", Parent Directory INode: 0, Size: 80B
Occupying blocks: 0

Directory: "a", INode: 1
Parent Directory: "/", Parent Directory INode: 0, Size: 32B
Occupying blocks: 1

Directory: "b", INode: 2
Parent Directory: "/", Parent Directory INode: 0, Size: 32B
Occupying blocks: 2

Directory: "c", INode: 3
Parent Directory: "/", Parent Directory INode: 0, Size: 32B
Occupying blocks: 3

cse312@ubuntu:~/Music$
```

Deleting a File:

```
cse312@ubuntu:~/Music$ make run45
./fileSystemOper data.dat del "/a/SecondA/D/file10"
cse312@ubuntu:~/Music$ make D
./fileSystemOper data.dat dir "/a/SecondA/D"
*****Directory Information*****
Directory Name: D
Creation Date: Fri May 27 19:11:09 2022
Last Modification Date: Fri May 27 19:12:16 2022

Directory Size: 176
Total Files & Directories: 11
Parent Directory Name: SecondA, INode: 6
Parent-Parent Directory Name: a, INode: 1
File Name: file1, INode: 35
File Name: file2, INode: 36
File Name: file3, INode: 37
File Name: file4, INode: 38
File Name: file5, INode: 39
File Name: file6, INode: 40
File Name: file7, INode: 41
File Name: file8, INode: 42
File Name: file9, INode: 43
cse312@ubuntu:~/Music$
```

Deleting a Directory:

```
cse312@ubuntu:~/Music$ make run49
./fileSystemOper data.dat rmdir "/a/FirstA"
cse312@ubuntu:~/Music$ make a
./fileSystemOper data.dat dir "/a"
*****Directory Information*****
Directory Name: a
Creation Date: Fri May 27 19:08:37 2022
Last Modification Date: Fri May 27 19:13:38 2022

Directory Size: 48
Total Files & Directories: 3
Parent Directory Name: /, INode: 0
Parent-Parent Directory Name: .., INode: 0
Directory Name: SecondA, INode: 6
cse312@ubuntu:~/Music$
```

Write to a File then Read from it:

```
cse312@ubuntu:~/Music$ make run70
./fileSystemOper data.dat write "/c/fileC4" data100.txt
cse312@ubuntu:~/Music$ make run71
./fileSystemOper data.dat write "/c/fileC5" data100.txt
cse312@ubuntu:~/Music$ make run72
./fileSystemOper data.dat write "/c/fileC6" data10000.txt
cse312@ubuntu:~/Music$ make run73
./fileSystemOper data.dat del "/c/fileC4"
cse312@ubuntu:~/Music$ make run74
./fileSystemOper data.dat read "/c/fileC6" Last.txt
cse312@ubuntu:~/Music$ make run74
./fileSystemOper data.dat read "/c/fileC6" Last.txt
cse312@ubuntu:~/Music$ make run75
make: *** No rule to make target `run75'.  Stop.
cse312@ubuntu:~/Music$ make cmp1
cmp data100.txt dResult100.txt
cse312@ubuntu:~/Music$ make cmp2
cmp data1000000.txt DAResult1000000.txt
cse312@ubuntu:~/Music$ make cmp3
cmp data10000.txt Last.txt
cse312@ubuntu:~/Music$
```