CSE312
Operating Systems

Homework 3 - Part 1 | 2021 - 2022

Ahmet Tuğkan Ayhan

1901042692

File System

```
typedef struct fileSystem_s {
    superblock_t      superblock;
    rootDirectory_t   rootDirectoryBlock;
    freeSpaceBlock_t freeSpaceBlock;
    iNode_t           * iNodeBlocks;
    directoryBlock_t * directoryBlocks;
    char              ** dataBlocks;
} fileSystem_t;
```

To make it easier to read and implement, I created a struct for every block in file system. Finally, after finishing the creation process of structs, I collected them into a general fileSystem struct.

My file system starts with a superblock. Then continues with root directory block, free space block, i-node blocks, directory blocks and finally data blocks.

I will explain every struct that I have created and how many bytes it occupies in the following sub-titles. Let's start with superblock:

Superblock

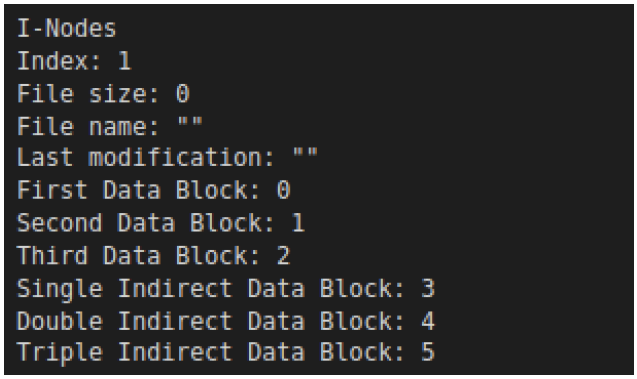
```
Superblock
Magic number: 0
Block count: 2048
Block size: 8192
Directory block count: 204
Directory count: 408
I-Node count: 408
Start position of root directory: 0
Start position of free space block: 128
Start position of data blocks: 416
Start position of I-Node blocks: 15081888
Start position of directory blocks: 15243864
```

```
typedef struct superblock_s {
    char magicNumber;
    int blockSize;
    int blockCount;
    int directoryCount;
    int directoryBlockCount;
    int iNodeBlockCount;
    int startOfiNodeBlock;
    int startOfRootDirectory;
    int startOfDirectoryBlock;
    int startOfDataBlock;
    int startOfFreeSpace;
} superblock_t;
```

The superblock contains metadata about our file system which is crucial to have. These metadatas located in the first block of our file system.

- **Magic Number(1 byte) :** It makes it easier for operating system to recognize the file format. It can be any integer value but I used 0 for my file system.
- **Block Count(4 byte) :** Block count is calculated according to block size which is given as parameter. We can have at most 16 MB file system. Because of that after converting 16 MB into Bytes, I divided block size(I converted it into bytes too) to 16 MB and found block count.
    - **KB :** 1024 Bytes
    - **MB :** KB * 1024 Bytes
    - **Max File System Size :** 16 * MB
    - **Block Size :** (Command Line Argument) * KB
    - **Block Count :** Max File System Size / Block Size
- **Block Size(4 byte) :** It shows how many bytes a block can contain. It is given by the user as a command line argument. If user gives incorrect size for block size, then user will be informed and program terminates without creating the file system.
- **Directory Block Count(4 byte) :** For directory block count, I gave %10 of the block count to the file system as directory block. Directory blocks starts after I-Node blocks.
- **Directory Count(4 byte) :** It is actually 2 times of directory block count. Because every directory block can contain 2 subdirectory.
- **I-Node Count(4 byte) :** For the i-node count, I gave %20 of the block count to the file system as an i-node block. I-Node block starts after free space block.
- **Start Position of Root Directory(4 byte) :** Shows which byte root directory starts from.
- **Start Position of Free Space Block(4 byte) :** Shows which byte root directory starts from.
- **Start Position of Data Blocks(4 byte) :** Shows which byte data blocks start from.
- **Start Position of I-Node Blocks(4 byte) :** Shows which byte i-node blocks start from.
- **Start Position of Directory Blocks(4 byte) :** Shows which byte directory blocks start from.

I-Nodes




```c
typedef struct iNode_s {
    int index;
    int fileSize;
    char fileName[NAME_LENGTH];
    char lastModification[DATE_LENGTH];
    int dataBlock1;
    int dataBlock2;
    int dataBlock3;
    int indirectDataSingle;
    int indirectDataDouble;
    int indirectDataTriple;
} iNode_t;
```

I-Node blocks contains the information about our files and stores file data. If 3 data block is not enough to store data inside the file, then indirect data blocks are used.

- **Index(4 byte) :** Shows which i-node is being used. For example if index is equals to 15, then it means this is the 15th i-node that being used.
- **File Size(4 byte) :** It shows how many bytes this file contains. It can be bigger than block size because each data blocks contain data with size of block size and indirect data blocks points to much more. Sum of them can be even GB of data. File size shows that property. (represented as KB)
- **File Name(41 byte) :** In my file system, name of the file can be at most 41 bytes. Any ASCII character can be used.
- **Last Modification(100 byte) :** Shows the last modification date of the file. It is represented as “dd/mm/yyyy hh:mm:ss” (d: day, m:month, y:year, h:hour, m:minute, s:second). Each time file is modified, this property will change.
- **First Data Block(4 byte) :** It shows byte index of the first free data block(at first use).
- **Second Data Block(4 byte) :** If first data block is not enough, then these extra blocks are used.
- **Third Data Block(4 byte) :** Final data block for the file data.
- **Single Indirect Data Block(4 byte) :** When the 3 data blocks are not enough, this index value is used. It indicates the first free indirect data block. These blocks contains a lot of data block indexes inside of them allocated for this i-node.
- **Double Indirect Data Block(4 byte) :** This time indirect double indirect data block indicates single indirect data blocks.

- **Triple Indirect Data Block(4 byte) :** Triple indirect data block points into double indirect data block.

Directory Blocks

```
Directory Blocks
Index: 1
Directory name: ""
Data: ""
Child Directory Block 1: 0
Child Directory Block 2: 1
```

```
typedef struct directoryBlock_s {
    int index;
    char directoryName[NAME_LENGTH];
    char data[DATA_LENGTH];
    int childDirectoryBlock1;
    int childDirectoryBlock2;
} directoryBlock_t;
```

We can think directories as files actually, but it differs from file in some ways. In my file system, a directory can have only 1 file inside, and 2 subdirectory. Number of blocks limited with %10 of my blocks.

- **Index :** Shows index of the directory block. If it is 90 for example, that means this is the 90th directory block.
- **Directory Name :** Stores name of the directory
- **Data :** A directory can only contain one data file inside of it.
- **Child Directory Block 1 :** It points into first free directory block. When this is used, it becomes child directory of this directory.
- **Child Directory Block 2 :** Second subdirectory(child directory).

Root Directory

```
Root Directory
Data block 1: 0
Data block 2: 0
Directory block 1: 0
Directory block 2: 0
```

```
typedef struct rootDirectory_s {
    int dataBlocks[2];
    int directoryBlocks[2];
} rootDirectory_t;
```

Different from other directory blocks, root directory has 2 data blocks and 2 directory blocks. It is located right after superblock in my file system.

- **Data Block 1 :** Points into an i-node address.
- **Data Block 2 :** Second data block for root directory

- **Directory Block 1 :** First subdirectory index for the root directory.(it points into a directory block)
- **Directory Block 2 :** Second subdirectory index for the root directory

Free Blocks

```
Free Space Block
File count at root: 0
Directory count at root: 0
Number of free blocks: 2048
Number of free I-nodes: 408
Number of free directory block: 204
Number of free data block: 1841
First free data block: 128
First free directory block: 15243864
First free I-node block: 15081888
```

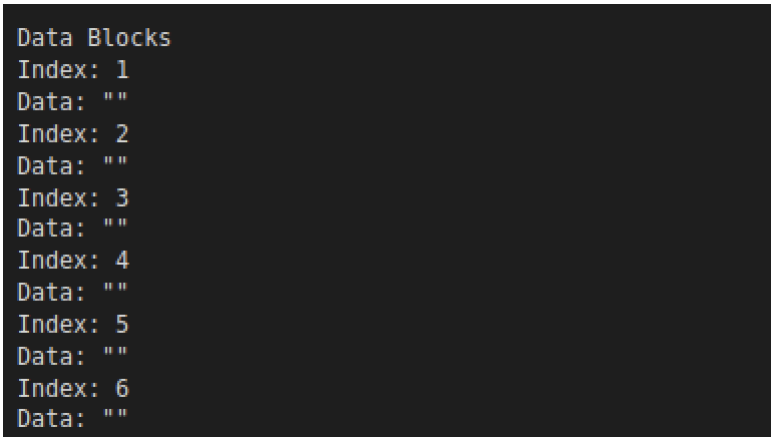
```
typedef struct freeSpaceBlock_s {
    int numberOfFreeBlocks;
    int numberOfFreeINodeBlocks;
    int numberOfFreeDirectoryBlocks;
    int numberOfFreeDataBlocks;
    int firstFreeINodeBlock;
    int firstFreeDirectoryBlock;
    int firstFreeDataBlock;
    int fileCountAtRootDirectory;
    int directoryCountAtRootDirectory;
} freeSpaceBlock_t;
```

Free block is crucial to have, because by using it we can see where is the next free spot for I-node block, directory block and free data block in our file system. It also shows how many blocks there are in the file system.

- **File Count at Root :** Shows how many file storage occupied inside the root directory. (in my file system it can me most 2)
- **Directory Count at Root :** Shows how many directory storage occupied inside the root directory. (it can be most 2 like files)
- **Number of Free Blocks :** At the first creation of file system, it is equal to number of total blocks. I explained how total block count is calculated under superblock subtitle. After every usage of any block type(i-node block, directory block, etc.)
- **Number of free I-Nodes :** At the first creation of file system, it is equal to number of total i-nodes. This number decreases with every new file and directory in the file system.
- **Number of free Directory Block :** At the first creation of file system, it is equal to number of total directory blocks. Then with every new directory block allocated, this number decreases

- **Number of free Data Block :** Data blocks are used to store data. Number of free data block count is calculated as:
    - **Total block count :** T
    - **I-Node Block Count :** I
    - **Directory Block Count :** D
    - **Root Directory(1 block) :** R
    - **Superblock(1 block) :** S
    - **Free space block :** F
    - **Number of data block :** T - (I + D + R + S + F)
- **First Free Data Block :** It points into first free data block location. This indexes shows byte location not the index(I mean it is not 1,2,3,4. Instead it is 0 2048 4096, etc - I assumed block size is 2 KB).
- **First Free Directory Block :** It points into first free directory block location. Same as first free data block.
- **First Free I-Node Block :** It points into first free i-node block location. Same as first free data block.

Data Blocks



Data block is the simplest block in the file system. It only stores index of the data block and data it need to store. Data size is given as command line argument.

- **Index :** It shows which data block is currently being used.
- **Data :** Data that is stored inside data block.

Part 3

Not implemented.