CSE 312
OPERATING SYSTEMS

MIDTERM PROJECT

PART 1

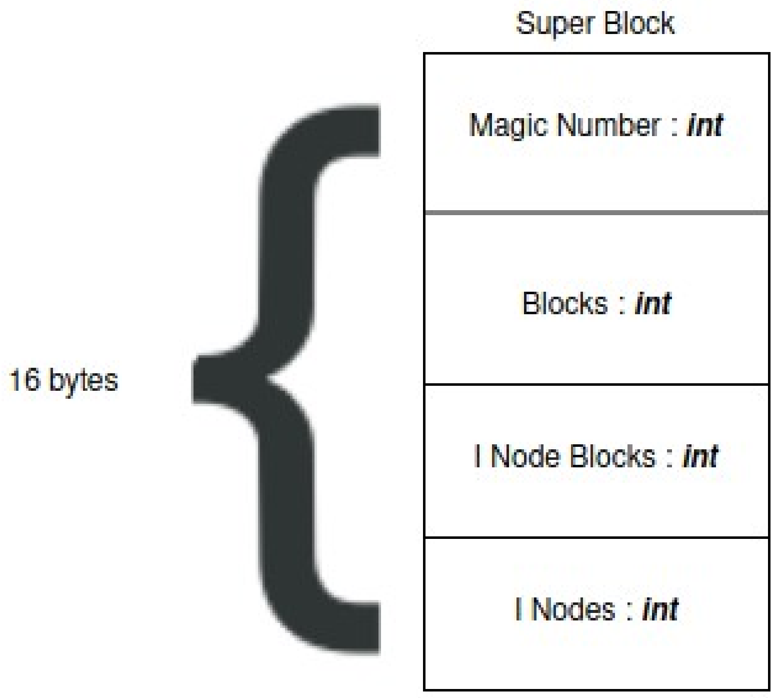
ÖMER ÇEVİK
161044004

1. <u>File System Architecture</u>

In my file system, there are i-node blocks and data blocks which are sized by program runner as KB. The i-node blocks keeps i-nodes that an i-node total size given by program runner. When the part2 run, it creates the empty file system as a file sized 1MB. In that file you can see the super block in the beginning then the i-nodes and data blocks.

Super Block

I have the super block which contains the *magic number* as integer that represents the validation of use the file system. If it is 1 then file system is available otherwise not. *Blocks* represents the how many blocks can be used in file system. That is evaluated using 1 MB – given total i-nodes sizes – super block size (16 byte). *I Node Blocks* represents the i-node blocks size evaluated by I Node size x 40 / 1024 / 4 (Block Size). 40 is the each size of I Node. *I Nodes* represents the size of I Nodes which is given by program runner. Super block is saving in the file system as like a JSON representation between two curly braces.

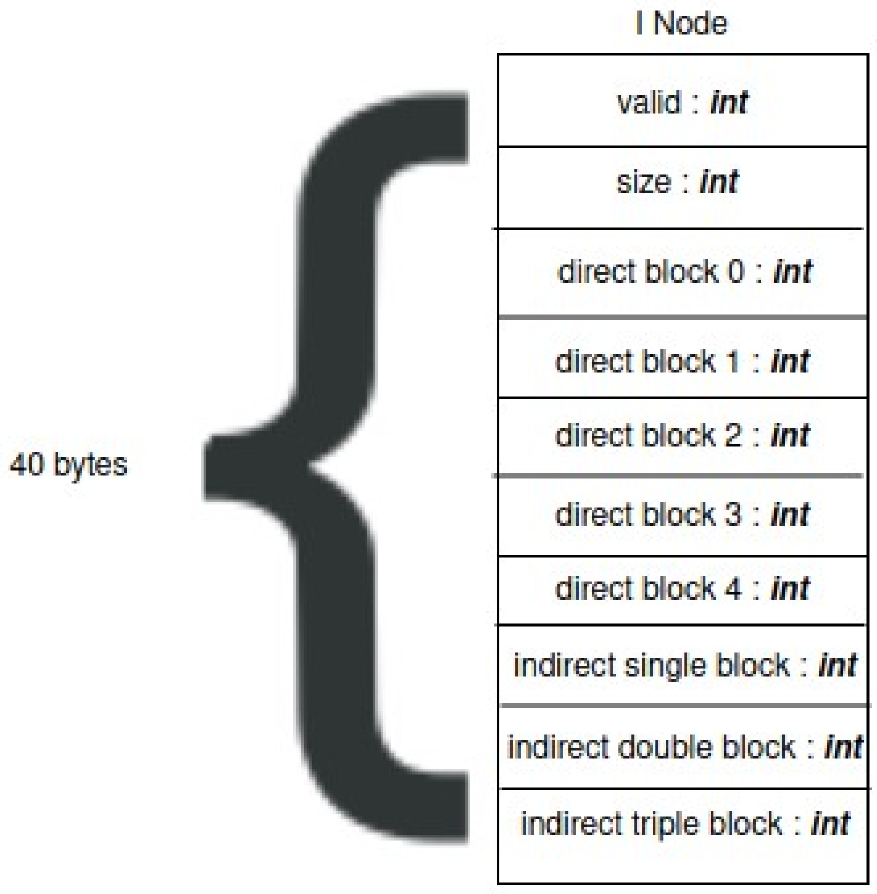


1. Super Block Structure

I Node

The I-Node contains the *valid* as integer if it is fully used then it is 0 otherwise 1. Each I-Node keeps the *size* of i-node blocks size as 4 KB (Block Size). Then it keeps *direct blocks* addresses which will be created and assigned. There are 5 direct blocks addresses in each i-node. In the beginning all empty direct block addresses represented by “NA”.

Also there are 3 *indirect data block* addresses: single, double and triple. If not in usage and empty, they are represented as “NA”.



2. I Node Structure

In the structure we can see that all the fields of i-node keeps integer and if we assume that integer is 4 byte then each i-node keeps 40 bytes in file system. I Node is saving in the file system as like a JSON representation between two curly braces.

Data Block

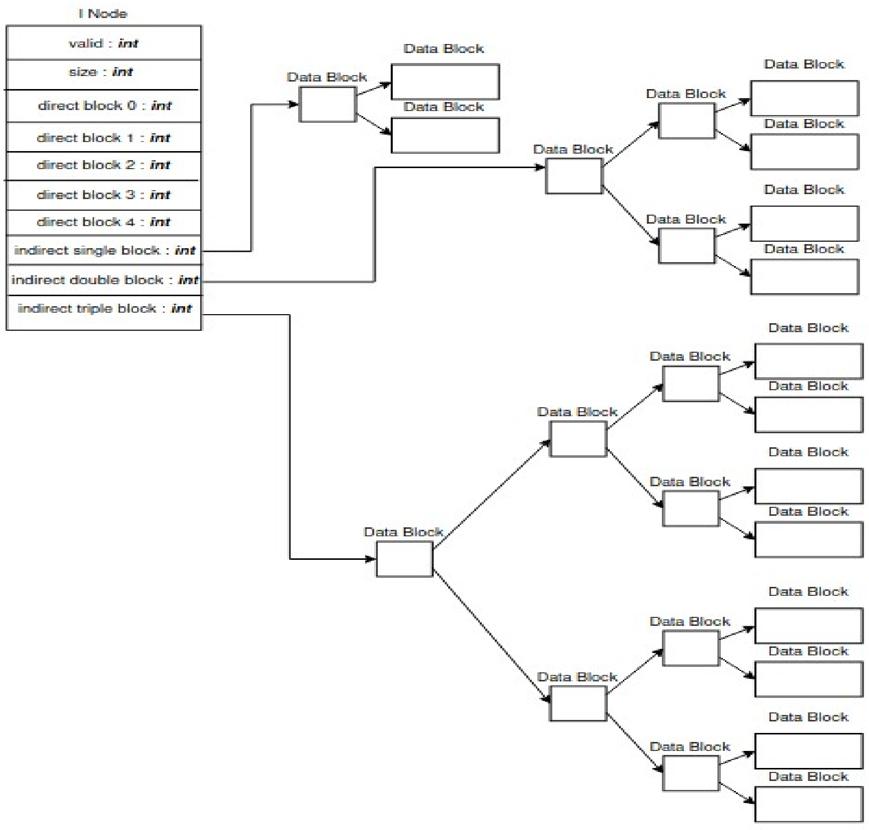
In the structure of data block the empty data block is represented as a JSON representation between two curly braces and it is empty between curly braces. Each data block can contain 4 KB (Block Size) datas.

Example:

```
Data-Block [0]
{
}
```

Indirect Data Block

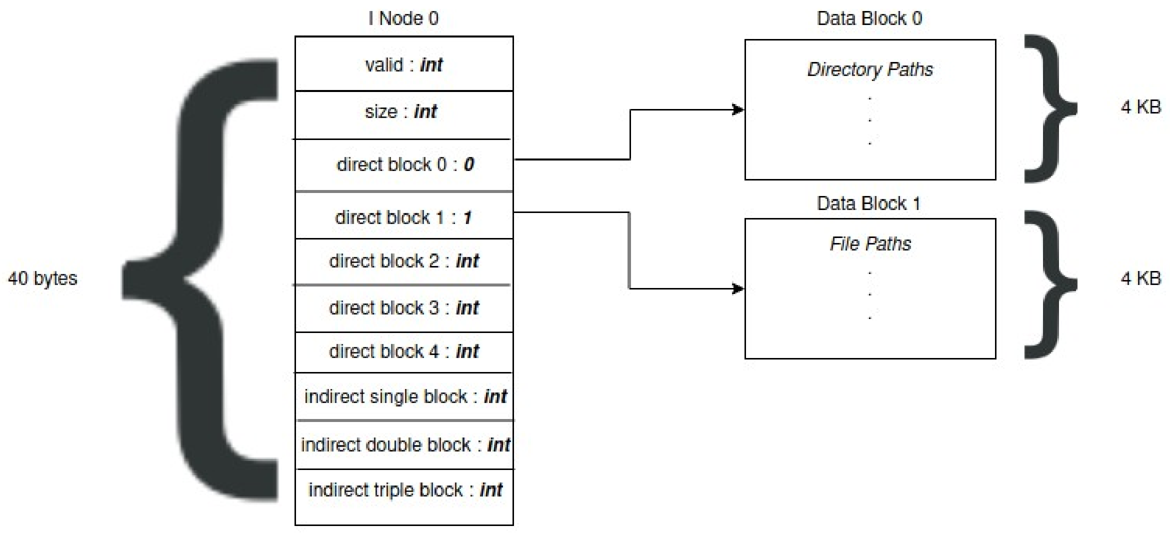
In the structure of indirect data block is about to keep multiple data block addresses in one address. A data block address is reserved to indirect data block address in a i-node as single or double or triple linked. The data block keeps multiple data block addresses. In a greater file systems data blocks can be use for each linked typed indirect data block addresses. But for 1 MB file system I used one data block for efficiency to keep data block addresses.



3. Indirect Block Address Structure

Directory & Files

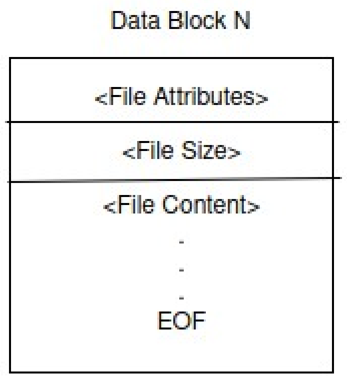
The directory keeps files with their attributes and sizes in data blocks. The files are separated by “EOF”. The address path of directories and some properties of directories are will be saved in a data block which is 0 indexed data block. The address path of files and some properties of files are will be saved in a data block which is 1 indexed data block.



4. Directory & Files Address Structure

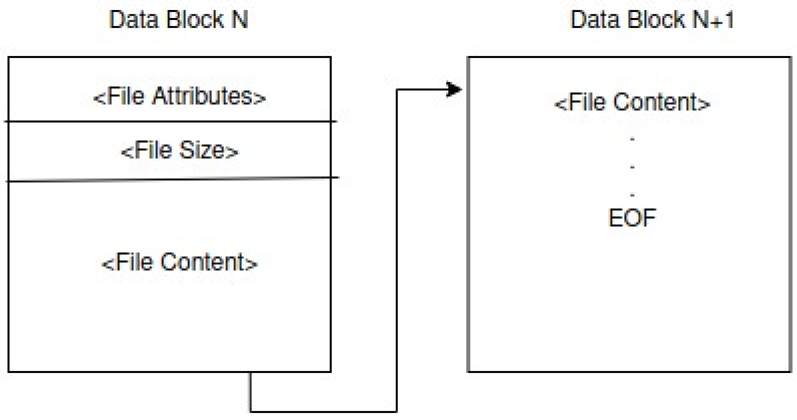
In this structure, the I-Node 0 represents the directories and files paths saved i-node. Direct block 0 and 1 are used for to keep paths of directories and files. Data block 0 initialized with root directory which is represented as “/”. If there is a root file then I-Node 0’s other direct blocks will be used. In the meaning, when a file is created in available directory, file gets the directory id and goes to that i-node with that index and sets the empty direct block address with which data block is used.

If the file’s size is smaller than 4KB (Block Size) then it writes its content to direct block with its attributes and size informations ended with EOF.



5. File Structure of Sized Under 4 KB (Block Size)

If the file size is greater than 4KB (Block Size) then it writes its each 4KB (Block Size) content to direct blocks which are next linked. For the last part of content is written the last direct block of links with EOF.



6. File Structure of Sized Over 4 KB

2. Part 2

In that part, in totally there are three functions created by me: *main()*, *getArguments()* and *createFileSystem()*. In main function, it calls the *getArguments()* and *createFileSystem()* functions. The *getArguments()* function gets the arguments in run arguments and saves. The *createFileSystem()* gets the saved arguments and creates a file and writes 1MB newline character inside in beginning to make a 1MB file system file. Then using total i-node size and block size finding the total data blocks, total i-node blocks and how many i-nodes will be in i-node blocks.

Then prints in the beginning of file the super block which contains the knowledge of *magic number*, *total data blocks size*, *total i-node blocks size* and *total i-node size*.
Then it appends to file the each i-node and their informations. After appended i-nodes, empty data blocks are going to be appended to file and closes file system in C++.

3. <u>Part 3</u>

In that part, there is a class named *FileSystem.* The main function gets arguments using *getArguments()* and saves the arguments of run. Then creates an object of FileSystem passing constructor parameter the file system name. In the constructor of FileSystem, there is a *readFile()* function call which reads the file system into *std::vector* of FileSystem object. Then gets the super block informations into FileSystem object using *getInitValues()* function. That functions are in *fileSystem.cpp* file. In the main, after getting the file system information calling the *executeCommand()* of FileSystem object passing necessary parameters. The *executeCommand()* gets command and parameters checks the which command is going to be executed: *mkdir, rmdir, write, read, del, list, dumpe2fs* etc.

mkdir

The mkdir command is executed by using *mkdirCommand()* function. It checks the root path is available and if not exist then sets it into i-node and data block. All directories keeps its owner index, previous directory index, directory path and creation time. Using previous directory index directories connects each other if nested directories are created.

rmdir

The rmdir command is executed by using *rmdirCommand()* function. It checks the path in directory paths and removes all directories which are connected to the given path. Before removing a directory it checks the file paths if there is a file inside of directory it removes firstly the file content in direct block then file path and finally the directory path. Before creating a directory it checks the available path.

write

The write command is executed by using *writeCommand()* function. It checks the path of directory if it is available then adds the file information into data block 1. Then gets an empty direct block and if the size of file content is smaller than 4KB it writes the content into one data block with EOF. If size of file content is greater than 4KB it writes content part by part nexted direct blocks. Then gets the id of directory and sets the direct block address in i-node which is directory id'th.

list

The list command is executed by using *listCommand()* function. It checks the paths of directory then adds the directory's paths into set. Then checks the path in file paths and then adds them into another set with size of files. Then prints them into screen with "directory/file-read-write-execute : d/-rwe". The total bytes are evaluated and printed as total bytes in the beginning.

del

The del command is executed by using *delCommand()* function. It checks the paths of directory about file then erases read content of file in data block. After erasing content of data block removes the path of file in data block 1 and resets the i-node address. If the file's size is less than block size then removes only one data block which contains the content of file, otherwise it removes next block size data blocks which are used by that file.

dumpe2fs

The dumpe2fs command is executed by using *dumpe2fsCommand()* function. It reads super block information about data block size, i-node size, i-node block size, begin of i-nodes, begin of data blocks; empty i-nodes and which are they, used i-nodes and which are they; empty data blocks and which are they, used data blocks and which are they; all directories in file system with path and dates; all files and properties about them. Then prints to console that all information.

read

The read command is executed by using *readCommand()* function. It creates a file which is passed as parameter by program runner and writes into it the file which is given the path by program runner. If the file content is greater than block size it reads linked, next, direct blocks until its content size and writes into the file.