2020

CSE 312/504 Part1

151044026
BATUHAN TOPALOĞLU

1. General information

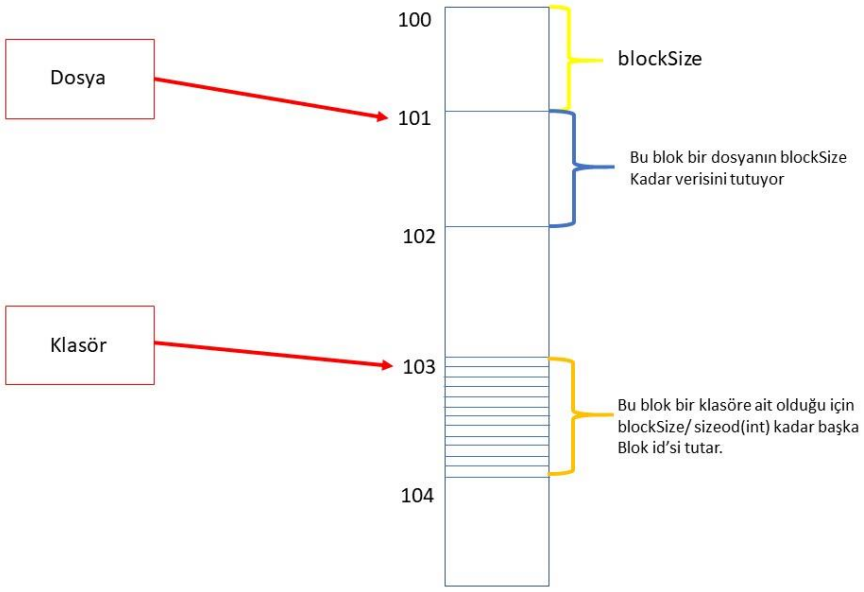
* The root folder is created by MakeFileSytem. I node belongs to zero root. Data block belongs to root in the first empty block.

* The smallest block can be 1kb. I limited these values to 1,2,4,8,16 and 32. This can be changed to be the force of 2. If more nodes are entered than the capacity of the system, an error is given.

* The file / folder name restriction is 14 characters. "." and ".." is not accepted as a name.

* There can be a maximum of 10 * (blockSize /16) sub folder or directory under a directory. For sample inputs 4 and 400 this value corresponds to 2560.

* How file and folder data is kept is shown below with an example drawing. (arrows represent the direct access arms of the node.)



2. Data types

2.a “superBlock”

The super block holds the basic parameters of the file system. "MakeFileSytem" calculates and writes this information to disk. "fileSystemOper" provides the operation of the system using this information.

```
struct superBlock
{
    int blockSize;
    int iNodeNumber;
    int numOfTotalBlocks;
    int numOfBlocksForINode;
    int numOfBlocksForSuperAndFSMG;
    int startBlockNumOfINodes;
    int startBlockNumOfDataBlocks;
};
```

If we examine the variables inside, *blockSize* is the byte equivalent of the value entered in the kb unit by the user. The 1Mb disk consists of these basic parts. *InodeNumber* keeps how many i nodes in the system. *NumOfTotalBlocks* keeps how many blocks are in the system. NumOfBlocksForInode keeps track of how many blocks are used to store i nodes. Likewise, the NumOfBlocksForSuperAndFSMG variable also keeps track of how many blocks are used to hold the superblock and free space management information. The StartBlockNumOfINodes variable holds from which block the list the nodes are kept from. Likewise, StartBlockNumOfDataBlocks is the first empty block number available on the system.

2.b “freeSpaceManagement”

Space management in the file system is provided through this structure. Whether the i node is used in the system or whether a disk block is in use is determined by this variable.

FreeBlockNum is the variable that holds how many empty blocks are in the disk.

```
struct freeSpaceManagement
{
    int freeBlockNum;
    char * freeINodeBlocks;
    char * freeDataBlocks;
};
```

The FreeINodeBlocks list is used for multiple purposes. If a i node is not used, there is an 'f' character in the corresponding index of the list. If an i node holds a place that is not linked, it will be marked as 'u'. If the node is hard-linked, it will be marked as "h" in the first link, and in each new link, it will change to the next ascii character. If the file is soft linked, it will be marked as 'a'.

In the FreeDataBlocks list, the index corresponding to the blocks used is 'u', the unused ones are marked as 'f'.

2.c “iNode”

I nodes hold the address (block id) of disk space containing the basic information and contents of a file or folder. I nodes hold 10 data blocks id with direct access. It can hold 1 single, secondary and tertiary indirect access data block id. “IsDir” is the variable that holds whether the node belongs to a folder or file. The variable that holds the size of the data is “size”. The last change date for folders is updated in case of adding / deleting a folder or adding / deleting files. For these files, it changes as write to the file.

```
struct iNode
{
    bool isDir;
    int size;
    int blogs[13];
    char lastModifiedTime[20];
};
```

2.d “dataBlockEntry”

There are "blockSize / 16" dataBlockEntry in a block. In this way, we can keep the files / folders under the folder. The file and folder name is limited to 14 characters.

```
struct dataBlockEntry
{
    short  nodeId;      // 2 BYTE
    char fileName[14];  // 14 byte dosy
};
```

3. Functions

(! The reasons for the difference in the names of the executable file of fileSystemOper when I took the screenshots were made at different times.)

3.a “list”

This function shows the folders and files in that folder if the path given as a parameter belongs to a folder. If path belongs to a file, like ls -l it shows only the information that belongs to that file. This information is as follows; file size (bytes), the last modification date (month name / day number / hour) and name. In addition, if the path belongs to a folder, the total size of the folder is printed as "total X" in kb as ls -l. It works compatible with files with hard and symbolic links.

Sample output for folders and files:

Incorrect path and parameter control was done in the function.

3.b “mkdir”

This function adds a folder to the file system according to the input given as a parameter. If the folder already exists or if the given path is invalid, it gives an error message and ends. With the command mkdir "." and “..” folder names have been checked a folder with this names cannot be created. Empty folder size is blockSize

Mkdir sample output with valid and incorrect entries:

```
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/"
mkdir: cannot create directory ‘/’: File exists
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/a"
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/b"
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/b"
mkdir: cannot create directory ‘/b’: File exists
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/a/abc"
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/ysa/abc"
mkdir: cannot create directory ‘/ysa/abc’: No such file or directory
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat mkdir  "/a/abc/g/g"
mkdir: cannot create directory ‘/a/abc/g/g’: No such file or directory
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/"
4096    May 29 06:30    a
4096    May 29 06:30    b

total : 8
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/a"
4096    May 29 06:30    abc

total : 4
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/b"

total : 0
cse312@ubuntu:~/Desktop/osVize$
```

3.c “rmdir”

This function deletes a folder from the file system according to the input given as a parameter. As in Linux, when non-empty folders are wanted to be deleted or the object to be deleted is not a folder, errors are given and deletions are not performed. Incorrect path and parameter checks were made.

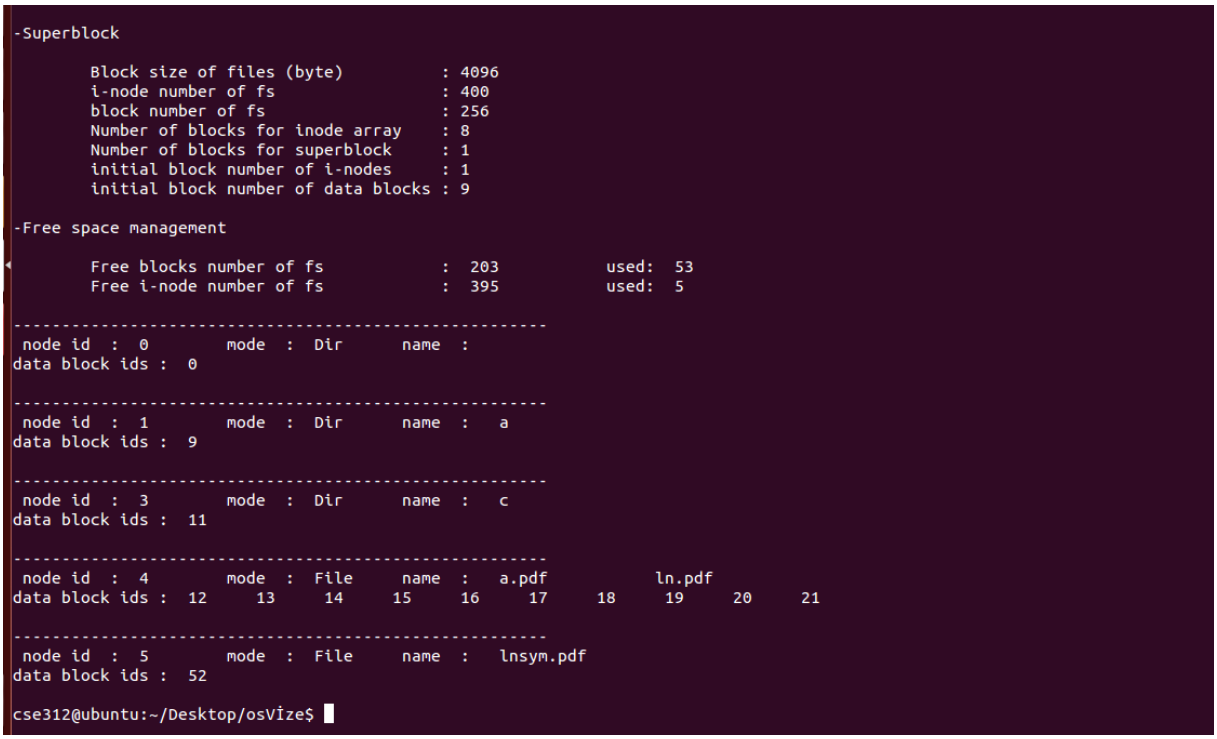
Sample output with incorrect and valid entries:

```
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat list "/"
Sublime Text
total : 0
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat mkdir "/test"
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat list "/"
4096    May 29 08:20    test

total : 4
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat rmdir "/test"
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat list "/"

total : 0
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat mkdir "/test"
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat mkdir "/test/abc"
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat rmdir "/test"
rmdir: failed to remove ‘/test’: Directory not empty
cse312@ubuntu:~/Desktop/osVize$ ./tst fs.dat rmdir "/tesTTT"
rmdir: '/tesTTT' no such file or directory
cse312@ubuntu:~/Desktop/osVize$
```

3.ç “dumpe2fs”

With Dumpe2fs function, information about the current status and general parameters of the file system can be obtained. As output, information about the blockSize, total number of blocks, number of nodes in the system and the number of free/used block and i node. After this information, the node id belonging to the nodes in use in the file system, the information about whether the node belongs to a folder or the file, the name of the file / folder that has the node and the ids of the data blocks used on that node are printed. "." and ".." is not added to the output. The name of the root folder appears blank.

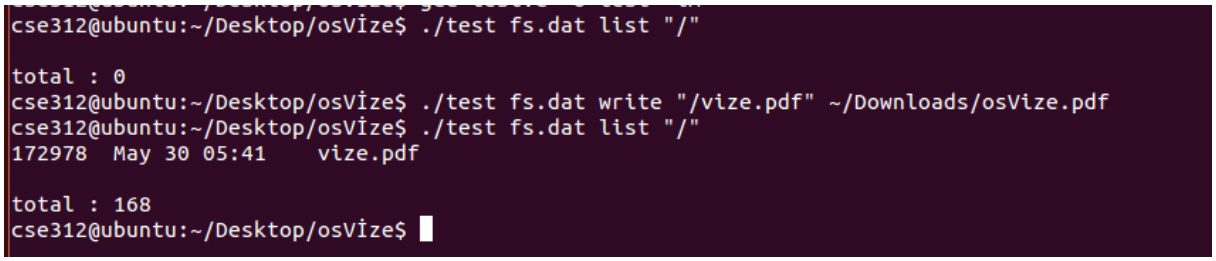
For example, there are folders a and c. The "dumpe2fs" function output of a system with hard-linked a.pdf and ln.pdf files, and lnsym.pdf file that is softly linked to a.pdf.



3.d “write”

With this function, we can import the files in the linux file system into our own file system. Incorrect path and filename check was done. The write function has the ability to overwrite. When it is desired to rewrite with the same name as the file in our own file system, it erases the old content and writes the new file in. If there is not enough space for the file to be included in the file system, the error message is not printed and copying is not performed.

Sample output before and after taking the homework pdf file into the empty file system:



3.e “read”

The read function allows us to extract the files included in our own file system to the linux file system with the write function. Incorrect path and parameter control was done. While designing the read operation, if the file to be created in linux already exists, it has been set to not take out. Attention should be paid to this during use. If the path you want to read with the read belongs to a folder instead of a file, it ends with an error message. It works compatible with files with hard and symbolic links.

Example output:

```
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/"

total : 0
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat write "/vize.pdf" ~/Downloads/osVize.pdf
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/"
172978  May 30 05:41    vize.pdf

total : 168
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat read "/vize.pdf" ~/Downloads/osVize2.pdf
cse312@ubuntu:~/Desktop/osVize$ cmp ~/Downloads/osVize.pdf ~/Downloads/osVize2.pdf
cse312@ubuntu:~/Desktop/osVize$
```

3.f “del”

With this function, the file is deleted from the system. Incorrect parameter and path control was done. If the object given path is a folder, an error message is printed. The next change date of the folder under the file is changed. The important point of this function is that it can work with “ln”. If a file is hard-linked with "ln", its deletion is different from other files. If any file showing the information about the linked file is available in the system, the del process is not deleted from the system until any file showing the content of that file is left in the system. The i node and blocks of the file are marked as empty / unused. It works compatible with files with hard and symbolic links.

Example output:

```
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/"
172978  May 30 07:13    vize.pdf

total : 168
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat del "/vize.pdf"
cse312@ubuntu:~/Desktop/osVize$ ./test fs.dat list "/"

total : 0
cse312@ubuntu:~/Desktop/osVize$
```

3.g “ln”

With the Ln function, the first file given in the system path is hard-linked to the second file. Function incorrect parameter and path controllers are made. One file can be linked to more than one file. No additional disk space is used as is required for the linked file. Situations such as the necessity for the object to be linked as a file should be checked. The same file could be linked to multiple locations.

Example output:

```
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat list "/"

total : 0
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat write "/vize.pdf" ~/Downloads/osVize.pdf
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat mkdir "/a"
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat ln "/vize.pdf"  "a/linkedF"
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat list "/"
172978  May 30 07:33    vize.pdf
4096    May 30 07:33    a

total : 172
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat list "/a"
172978  May 30 07:33    linkedF

total : 168
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat del "/vize.pdf"
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat list "/"
4096    May 30 07:33    a

total : 4
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat list "/a"
172978  May 30 07:33    linkedF

total : 168
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat del "/a/linkedF"
cse312@ubuntu:~/Desktop/osVİze$ ./test fs.dat list "/a"

total : 0
```

3.k “lnsym”

Write Sembolik linkleme işlemi yapan fonskyon için read, write, list, del gibi fonksiyonlar yeniden yapılandırılmıştır.

Example output:

```
cse312@ubuntu:~/Desktop/osVİze$ ./fileSystemOper fileSystem.data list "/"

total : 0
cse312@ubuntu:~/Desktop/osVİze$ ./fileSystemOper fileSystem.data write "abc.pdf" bt.pdf
cse312@ubuntu:~/Desktop/osVİze$ ./fileSystemOper fileSystem.data list "/"
159150  May 31 08:09    abc.pdf

total : 155
cse312@ubuntu:~/Desktop/osVİze$ ./fileSystemOper fileSystem.data lnsym "/abc.pdf" "/def.pdf"
cse312@ubuntu:~/Desktop/osVİze$ ./fileSystemOper fileSystem.data list "/"
159150  May 31 08:09    abc.pdf
8       May 31 08:23    def.pdf -> hiii
/abc.pdf

total : 155
cse312@ubuntu:~/Desktop/osVİze$
```

3.ı “fsck”

The fsck function file is for provisioning for the nodes and blocks of our file system. There are two tables for nodes and blocks. In the first table, the free inode list held by free space management is compared with the used node list calculated by the fsck function. In this way, the free node list kept on disk is provided correctly. Likewise, the fsck function also calculates the blocks used and compares this to the free block list maintained in fsmg. When there is no 0.0 or 1.1 status under the same id, it means there is no problem in the system.

(output is written line by line.)

The first line contains the information for which i node. List computed by fsck in line 2. The third line contains the list kept on disk.

Here are the first 25 nodes of an empty file system. Only the 0th node is used by root.Likewise, for empty mesh system, this time the blocks can be provided. The first 9 blocks look full because they are used by super block, fsmg and root folder.

```
>>>> i-nodes :

-------------------------------------------------------------------------------------------------------------------------
i-node ids     : 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
i-nodes in use : 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
free i-nodes   : 0   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
-------------------------------------------------------------------------------------------------------------------------
```

All nodes and blocks in the Full System can be controlled in this way.

```
>>>> blocks :

-------------------------------------------------------------------------------------------------------------------------
block ids     : 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
blocks in use : 1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
free blocks   : 0   0   0   0   0   0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
-------------------------------------------------------------------------------------------------------------------------
```