Gebze Technical University
Department of Computer Engineering
CSE312/CSE504
Operating Systems
Spring 2020

Midterm Exam Project

Instructor : Yusuf Sinan AKGÜL

Student Name: Can BEYAZNAR
Student No: 161044038

Can BEYAZNAR
161044038

Definition of directory structure and directory entries

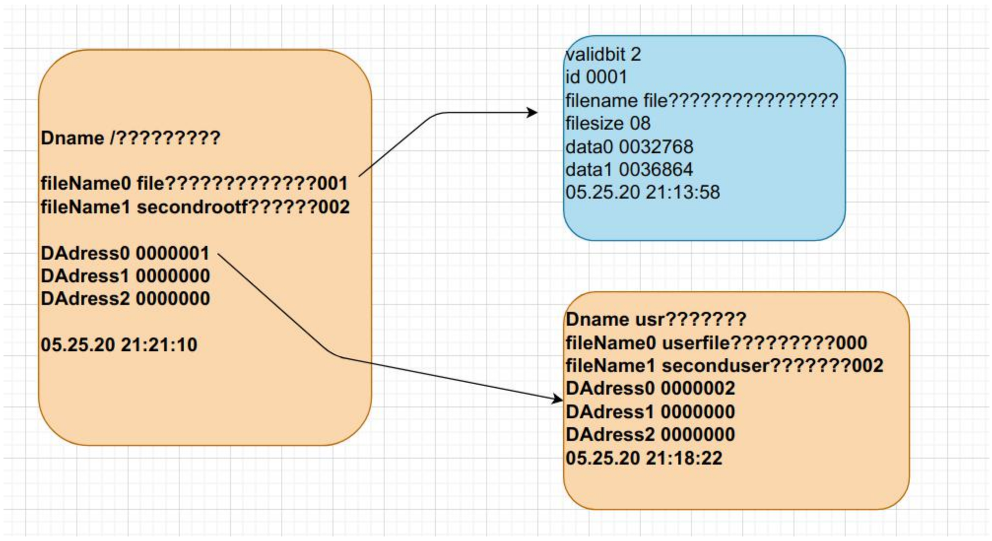
In my directory entries, I kept a total of 7 parameters. I will explain these parameters later. Each directory entry occupies 148 bytes for these parameters. And I am allocating 1 block for my directory structure. According to the options of 1 KB, 2 KB, 4 KB, 8 KB and 16 KB, the total number of directories I keep varies. Information I keep in my directory entry:

1- Directory name(Maximum 10 bytes): This will keep my directory name. Directory name of the first directory entry is “/”. Because the first index must be root directory

2- 2 File names (Maximum 16 bytes): These will keep my file names.

3- 3 Directory address markers (maximum 7 bytes each as it is a 1Mb file system): These will keep my pointed directory addresses.

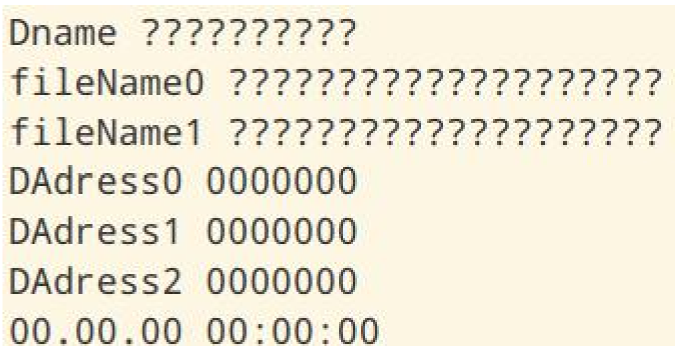
4- Date (month.day.year hour:minute:seconds) (17 bytes in total): This will keep last change date of my directory entry



Note: The file names take up 20 bytes in total. I am using the last 4 bytes to hold the inode number, as there should be an inode pointing to each file. For example, "file ????????????? 001" filename "file" with the inode number 001.

Blank bytes in filenames in my directory structure "?" I put my character and fill it. So the user "?" File names with the mark cannot be entered. Also, after running Part2 part, root directory is defined in the first index of the folder structure. And it is absolutely necessary for the file system to work in Part3.

My directory entry looks like the following:



```
Dname ??????????
fileName0 ????????????????????
fileName1 ????????????????????
DAdress0 0000000
DAdress1 0000000
DAdress2 0000000
00.00.00 00:00:00
```

Definition of directory keeping free blocks and free i-nodes

I keep my empty blocks and empty inodes thanks to my inode entries. I keep 2 parameters in each inode entry to solve this problem. One is validbit, the other is fileize. validbit indicates if my inode is full. If 0 is empty, if it is greater than 0 it indicates that it is full. filesize shows how many KB the file takes up. So I can find out how many datablocks are used in total. I will explain a more detailed explanation in the inode section.

Definition of i-node structure

In my file system, I reserved %5-6 of my entire file system to keep my inode entries. Each i-node entry occupies a total of 107 bytes. The information of the parameters in my i-node entry is as follows:

1- **Validbit :** Indicates whether i-node is full. If 0 is empty, 1 is full, if it is equal to 2 or greater than 2, it has a hard link. By holding this information, I can learn the number of empty inodes. Maximum 1 bytes.

2- **Id :** I keep this parameter because each inode must have a special id. In this way, I can find out which inode I am making changes while the user enters. Maximum 4 bytes.

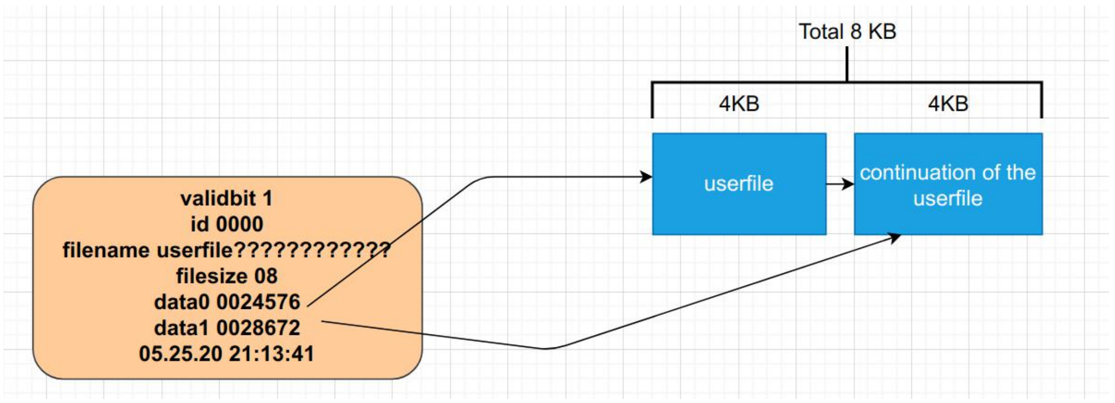
3- **Filename:** This parameter holds the file name. It can take a maximum of 20 bytes.

4- Filesize: The filesize parameter shows how many kb of the data blocks it points to occupies in total. Each inode entry points to 2 data block addresses in total. So each of my inodes holds 2xblock size. For example, if the user gives 4 kb as a parameter, each inode entry can point to a 8kb data block address. Also, if the data it writes to the file is less than 4kb, it will be filesize 4 (for this example only). I can learn my empty blocks with this parameter.

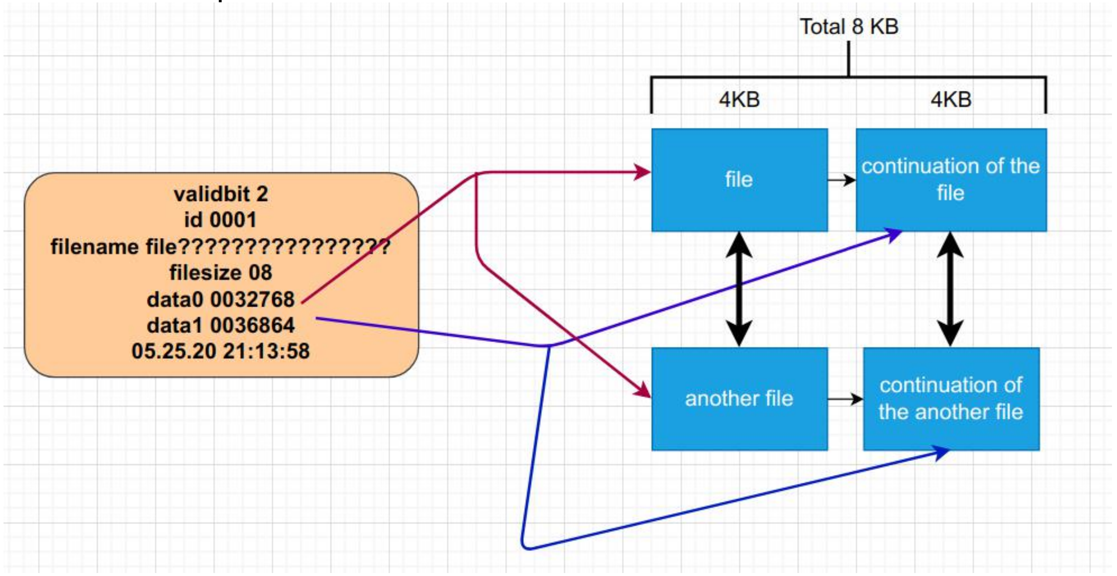
5- Data0 and data1 : data0 and data1 point to the address of the data blocks they have. The space in the data blocks is reserved for the file in the inode entry, and these two parameters point to the address of those blocks. In total, a maximum of 2 * blocks can point to data blocks. In fact, the 2 information is interconnected. For example, if each block is 4 kb for you, if the user input data is 4 kb, one data parameter is used. If Data0 = data address, data1 = data0 + is equal to blocksize. Each parameter can take a maximum of 7 bytes.

6- Date : Keeps the last modification date of the inode entry. takes a maximum of 17 bytes.

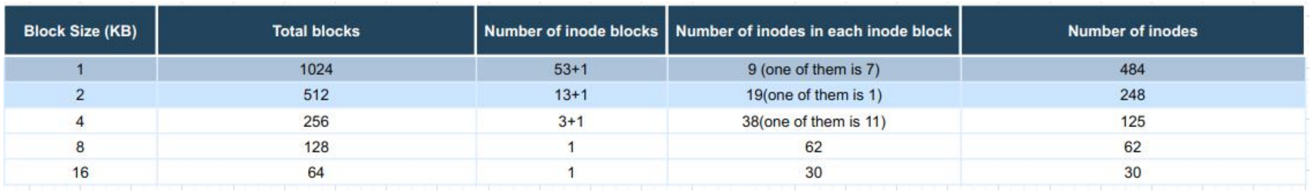
Example:



Hard link example:



According to the block size entered by the user, information such as the maximum number of inodes, the number of wasted blocks are as in the table below.



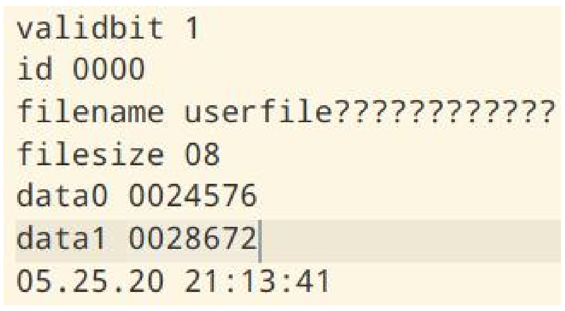
Block Size (KB)	Total blocks	Number of inode blocks	Number of inodes in each inode block	Number of inodes
1	1024	53+1	9 (one of them is 7)	484
2	512	13+1	19(one of them is 1)	248
4	256	3+1	38(one of them is 11)	125
8	128	1	62	62
16	64	1	30	30

Note: The table shown has a special condition for 8 KB and 16 KB. These two block sizes cannot fully fill their capacities for calculations. Normally for 8KB, 1 inode block can hold 76 inode entries in total. However, since we have a limit of 1MB, we should not exceed this limit. Therefore, the maximum number of inodes it can receive is 62. For the 16KB block size, this number has decreased to 30 while it can receive a maximum of 153 inode entries.

In addition, it is necessary to explain for 53 + 1 in "Number of inode blocks" for 1KB. In 53 inode blocks, a total of 9 inode entries are kept inside each one. However, less inode entries are kept in the +1 section, the extra part. This is because we have Limited space. And it is a method created to avoid wasting the idle block. The extra inode block of +1 holds 7 inode entries.

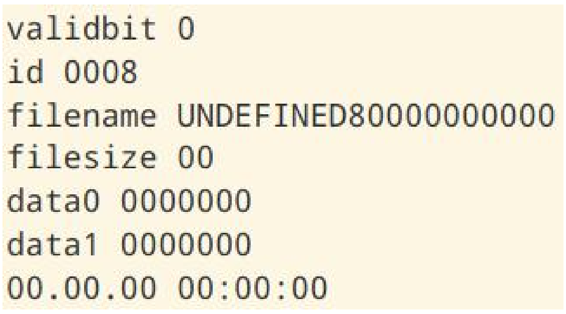
Important note: In Part2, if the user writes a number higher than the maximum inode number given as input, the program ignores this input and accepts the maximum number of inodes it can receive. And if the user enters 0 or a negative number, he will not accept it.

My inode entry looks like the following:



```
validbit 1
id 0000
filename userfile????????????
filesize 08
data0 0024576
data1 0028672
05.25.20 21:13:41
```

For undefined inodes:



```
validbit 0
id 0008
filename UNDEFINED80000000000
filesize 00
data0 0000000
data1 0000000
00.00.00 00:00:00
```

Definition of Superblock

I have reserved 1 block for my Superblock, and it is in the first place in my file system. I kept a total of 7 pieces of information in Superblock. This information is as follows:

1- **File name:** This parameter holds the name entered by the user for the file system in part2. In order for part3 to work, the same name must be entered for the file system. Maximum 20 bytes.

2- **End of the file:** This will point the address of end of file. Maximum 7 bytes.

3- **Number of blocks:** This parameter holds the total number of blocks in the file system.

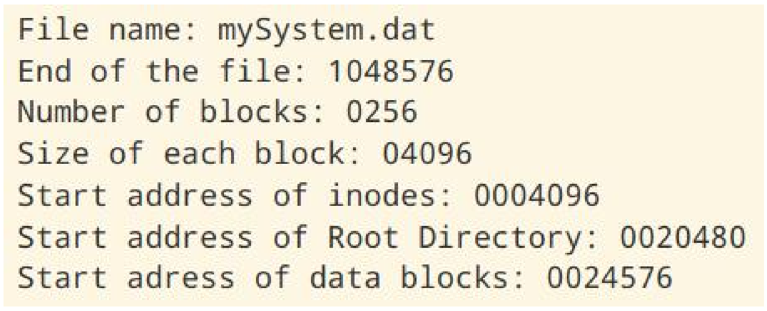
4- **Size of each block:** This parameter is the block size that the user entered as input in part 2. It takes a maximum of 5 bytes.

5- **Start address of inodes:** This parameter points to the address where the inode entries start. Maximum 7 bytes.

6- **Start address of Root Directory:** This parameter points to the address where the root directory start. Maximum 7 bytes.

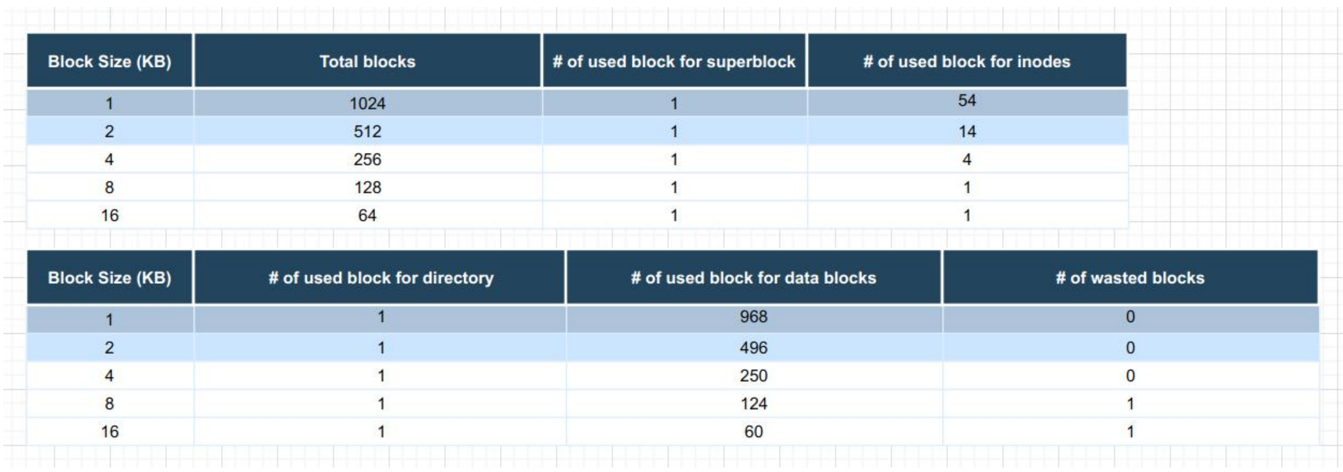
7- **Start address of data blocks:** This parameter points to the address where the data blocks start. Maximum 7 bytes.

My superblock entry looks like the following:



```
File name: mySystem.dat
End of the file: 1048576
Number of blocks: 0256
Size of each block: 04096
Start address of inodes: 0004096
Start address of Root Directory: 0020480
Start adress of data blocks: 0024576
```

I have indicated the number of blocks that I allocate to my structures according to the block size entered by the user in the table below. At 8 and 16KB, 1 block is wasted. This is because I can not use it as a result of the mathematical calculations I have made.



Block Size (KB)	Total blocks	# of used block for superblock	# of used block for inodes
1	1024	1	54
2	512	1	14
4	256	1	4
8	128	1	1
16	64	1	1

Block Size (KB)	# of used block for directory	# of used block for data blocks	# of wasted blocks
1	1	968	0
2	1	496	0
4	1	250	0
8	1	124	1
16	1	60	1

Definition of Functions

1- Main functions

```
int main(int argc, char *argv[])

int read_superblock(FILE *file_read, my_superblock &superblock)

void read_INodes(FILE *file_read, my_superblock superblock, my_INode INode_Arr[], int
maxInode)

void read_Directories(FILE *file_read, my_superblock superblock, my_Directory
Directory_Arr[], int maxDirectory)
```

```
void make_operation(char *argv[], string fileName, my_superblock superblock, my_INode
INode_arr[], my_Directory Directory_arr[], int maxINode, int maxDirectory)

void list_op(string fileName, vector<string> dirNames, my_superblock superblock,
my_Directory Directory_arr[], my_INode INode_arr[], int maxINode)

void mkdir_op(string fileName, vector<string> dirNames, my_superblock superblock,
my_Directory Directory_arr[], int maxDirectory)

void rmdir_op(string fileName, vector<string> dirNames, my_superblock superblock,
my_Directory Directory_arr[], int maxDirectory)

void dumpe2fs_op(my_superblock superblock,my_INode INode_arr[],
my_Directory Directory_arr[],int maxINode, int maxDirectory)


void write_op(char *argv[], string fileName, vector<string> dirNames, my_superblock
superblock, my_Directory Directory_arr[],my_INode INode_arr[], int maxINode)


void read_op(char *argv[], string fileName, vector<string> dirNames, my_superblock
superblock, my_Directory Directory_arr[],my_INode INode_arr[], int maxINode)

void del_op(char *argv[], string fileName, vector<string> dirNames, my_superblock
superblock, my_Directory Directory_arr[],my_INode INode_arr[], int maxINode,int
maxDirectory)

void ln_op(string fileName,vector<string> dirNames1,vector<string>
dirNames2,my_superblock superblock, my_INode INode_arr[],my_Directory
Directory_arr[],int maxINode,int maxDirectory)

void fsck_op(my_superblock superblock, my_INode INode_arr[], int maxINode)
```

2- Helper functions

```
int getNumberOfMaxINode(int blockSize);

string returnTime()

int return_numberOfBytes(int val, int max_size)
int findCharecterinStr(string _str, char character )
string appendZeroToStr(string input, int numberOfZero)
void intToStringWithZero(int value, string &myStr, int maxByteSize)
string appendMarkToStr(string input, int numberOfNull)
int getNumberOfEachInode(int blockSize);
int getMaxBigINode(int blockSize);

void write_Datas_To_SystemFile(string filesystem_name, string user_fileinput, int
user_fileSize, int fseek_start)

void write_Datas_To_InputFile(string filesystem_name, string user_fileinput, int
byte_size_write, int fseek_start)

void fill_zero_filesystem(string filesystem_name, int eachTime, int totalByte, int
fseek_start)

void update_inode(string fileName, my_superblock superblock, my_INode INode_arr[], int
index)
```

```
void update_directory(string fileName, my_superblock superblock, my_Directory
Directory_arr[], int index)

int add_directory_recursive(string fileName, my_superblock superblock, int
currentDirectory, my_Directory directoryArr[], int directorySize, vector<string> dirNames,
int pathCount, int maxDirectory)

int remove_directory_recursive(string fileName, my_superblock superblock, int
currentDirectory, my_Directory directoryArr[], int directorySize,
vector<string> dirNames, int pathCount, int maxDirectory)

vector<int> returnDirIndexes(vector<string> dirNames, my_superblock superblock,
my_Directory directoryArr[])


vector<int> returnDirIndexes_listop(vector<string> dirNames, my_superblock superblock,
my_Directory directoryArr[])

vector<string> parse_string(char *char_Arr, const char *parseCharacter)

int getEmptyINode(my_INode INode_Arr[], int inode_size)

int getFileSizeOfDatablock(string filesystem_name, int start, int end)

string get_inode_newfileName_in_directory(my_Directory directory[],int maxDirectory ,int
INode_index)
```

3- Classes

```
class my_superblock

class my_INode

class my_Directory
```